A Tinderbox Reference File: v9.6.1 - Table of Contents

A Tinderbox Reference File

Update Status: aTbRef source TBX last updated: Sun, 8 Oct 2023 13:34:59 +0100 :: RSS 2.0 Feed :: Atom 1.0 Feed :: JSON 1.0 Feed.

"**aTbRef**" (**A T**inder**b**ox **Ref**erence) is a *reference* file aboutTinderbox's objects and functions, from which this website is generated since 2004. It has been written and updated by Mark Anderson a long-time user of Tinderbox (see more) an is offered to the Tinderbox community as a free public service under a Creative Commons BY-NC-SA licence (licence details); voluntary donations to help defray costs are welcomed—see the link at the bottom of all pages.

Current Version: v9.6.1b638 (15 Aug 2023)

Version Baseline: this aTbRef95 website is baselined on v9.5.0b597 (released 9 December 2022) and includes changes *up to and including* v9.6.1b638 (15 Aug 2023). To look at changes to the app over time, see other baselines, based on older versions. Also see Eastgate's version release pages and release dates of versions prior to the current baseline.

Change Log: see this section for significant per-release changes *in and since* the baseline release (above) for this re-release of aTbRef. In-text references to per-release changes prior to the baseline have been removed as they are now baseline features; some may remain if still pertinent to understanding the app's functioning. If a page is listed as created a version not in the Change Log, it is simply because it has been exported using a (non-public) beta build.

Quicklinks: a set of 'quicklinks' to key listings within aTbRef are at the top and bottom of every page, including to the sitemap which is the closest there is to an index for aTbRef.

Contents:

REMARKS AND CAVEATS

Is there an index for aTbRef? An auto-index would be too noisy and a human-created one would be too much effort, especially as aTbRef is updated constantly. But the aTbRef Site Map page, used with your web browser's in page Find (⌘+F) is a pretty good substitute.

Can't find a subject? Please visit the Tinderbox User-to-User forums and ask there. If doing so it helps to know how you phrase your question and where you thought you might find the answer. Sometimes the issue is simply one of terminology/process but sometimes we find something that should and is added to aTbRef (note: 'how-to' for specific tasks is not and will not be in aTbRef: it is a reference not a tutorial). This file is supposed to go along with the existing manual & release notes (available via the app's Help menu) as well as the Tinderbox forum.

This is not a how-to/tutorial. Although aTbRef contains some examples of use, it is not, by design, a 'how-to' resource. 'how-to' questions are best addressed by the Tinderbox forum. Its main aim is to serve as an "on the go" (online) as wel as a personal (TBX file) reference to Tinderbox use.

Install, Uninstall, Support and Registration

To what does 'TB' refer?

The word Tinderbox when referring to the application is, by common usage, often shortened to 'TB' in discussion in places like the user-to-user forum. This use of 'TB' is not an acronym. Rather, it is just a simple shorthand for those who find they need to type the word Tinderbox a lot!

It is also worth noting that the correct spelling of the application name is 'Tinderbox' and not 'TinderBox'. The latter is sometimes assumed from seeing the TB 'short' name.

'TBX', the file extension used by Tinderbox file, is also used as a shorthand for a Tinderbox document(s).

Technical Requirements

Tinderbox runs on all modern Macintosh computers.

From v9.5.0, Tinderbox requires macOS 11 (Big Sur) or later.

Tinderbox v9.0.0 runs on macOS Big Sur (11.x), Catalina (10.15) and Mojave (10.14) and High Sierra (10.13).

For Sierra (10.12), El Capitan (10.11) and Yosemite (10.10), use Tinderbox 8.5.

Note that not all features run old the older supported OS versions, especially those using AI and Machine Learning techniques. There is a list of such features.

Features with OS limit higher than app base specification

The following features require an OS later than the current baseline of macOS 10.13. See the individual notes for details of the OS-based limitations:

- macOS Dark or Light modes
- Exploding Notes
- Built-in Hints container
- Sentiment Analysis
- Taggers
- Find results stand-alone dialog
- Find results pop-over
- Find toolbar (view pane)
- Window menu
- Colors
- String.paragraphList()
- String.wordList()
- String.nounList()
- wordsRelatedTo(dataStr[, wordsNum])
- favorites folder

Licence & Registration

The Tinderbox Licence

The licence (EULA) can also be view in the Tinderbox Help file.

Eastgate Systems Inc.

134 Main Street

WatertownMA 02472 USA

Tel: (800) 562-1638 (617) 924-9051

Fax: (617) 924-9051

Email: info@eastgate.com

Web: https://www.eastgate.com/

Registering Tinderbox for the first time

When first purchasing Tinderbox, a registration code will be emailed. The code is applied via the Register tab of Preferences, accessed from the Tinderbox menu.

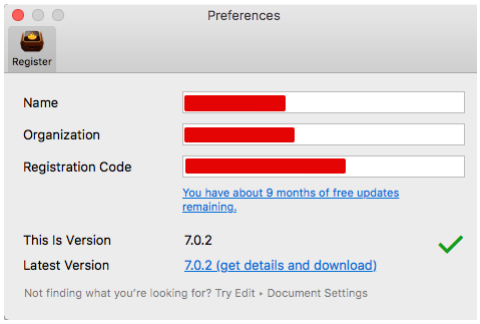
Once registered, the Register tab of Preferences will show how many months' entitlement of free updates are left.

Until a registration code is applied, the application runs in demo mode.

Possible problems applying a licence

The most common registration problems are:

- Mixing up old and new registration codes, e.g. when applying an upgrade licence but using a previous code
- Using a different name (or middle initial) in the registration window from the name on the order. To Tinderbox, J. Doe, joe and J. Q. Doe are all different names.
- omitting the optional organisation name if one was given during purchase.
- mistaking the letter "l" (lowercase L) and the numeral "1" in the registration code.



Licence renewal and upgrade requirements

One year from first registering, the right to use new major and minor versions Tinderbox ceases unless you purchase an upgrade licence. An 'upgrade' licence renews free use of new releases for a further year, i.e. for 12 months from the issue date of upgrade licence. If your current cover has not quite expired the new period is added to the end of the existing one.

Therefore, those wanting continued full access to all new releases should purchase upgrade licences on a yearly basis, but this is not a requirement for continued use.

An upgrade licence allows the user to move from any past Tinderbox version to the latest release. As the licence code controls access to all releases, the terms upgrade and update can be regarded as interchangeable.

If a user is outside the free update cover period, new releases may still be downloaded and tried, though use a separate install location and note the the newer version will run in demo mode.

The user is not prompted for upgrades, but it is possible to check the number of remaining months access to new versions and the current licence code via the Register tab of Preferences.

It is not a requirement that upgrades be purchased before expiry of the current free update period. This offers flexibility, especially for those with limited budget for upgrades. Thus it is allowable for the original or a renewal upgrade licence's free upgrade period to elapse and then purchase a new upgrade at some later date. Simply register the upgrade, download the current version and carry on installing new versions for a further year.

Put another way, an existing user can run any version of Tinderbox. If the release date of that version post-dates the end of their free upgrade access, the app will run in demo mode. If within the current access period, the app will run normally the upgrade licence ensures continued unlocking of new releases.

Registering an upgrade

The process is the same as with initially registering Tinderbox. When purchasing an upgrade licence, a new registration code will be emailed. The code is applied via the Register tab of Preferences, accessed via the Tinderbox 8 menu.

Once registered, the Preferences Register tab will show how many months' entitlement of free updates is left.

Possible problems applying a licence

See Registering Tinderbox for the first time.

Installing a licence on more than one computer

Note licence conditions about installing the same licence on more than one computer (see manual for complete licence), in that you may only:

- Install the program on a second computer you use provided all copies of the program are used directly and exclusively by yourself.
- Allow anyone else to use the program, so long as there is never more than one user per licensed program at any time;

Put in simple terms, you can install it as many times as you like on different computers as long as you are the only user. For those who work on multiple computers at once, the licence does not check concurrent use so you can, for example, use your Tinderbox on you desktop and laptop at the same time.

If your licence for free upgrades has expired, install your current version on the new computer.

Separately, the 'Household' version of the licence allows:

- You and up to four (4) other persons who occupy the same household may use the program on their respective computers. "Household" means a person or persons who share the same home, apartment, condominium or dormitory suite, but shall also extend to student household members who are primary residents of the household but who reside at a separate on-campus location.

In essence, the latter is a 5-seat concurrent access licence.

Installation

- Installing Tinderbox
- Installing a new version
- Updating an existing installation
- Installing more than one version
- Migrating Tinderbox to a new computer

Installing Tinderbox

To install Tinderbox, download the disk image (DMG) and double-click it to mount the image. Drag the application icon onto the shortcut to your Applications folder to install the app. If installing over an existing version you will be asked if you wish to replace it or to keep both versions. For normal use, replace any existing version.

If the mounted image contains no shortcut to Applications (e.g. with beta builds), simply open a Finder window showing the Applications folder and drag the drag the new app icon onto it.

Once installed, open the Applications folder, find the Tinderbox icon and double-click it to open Tinderbox. On very first use you will be asked for permission to open the app, as part of standard Mac security procedures.

Installing on macOS 10.8+ with default permissions

From v10.8, Apple has changed the defaults for installing applications as part of improving security for the general user. As a result, after installation, on a default 10.8.x system the user may receive a warning dialog along the lines of "Tinderbox" can not be opened because it is from an unidentified developer. The error is slightly misleading, as Eastgate is a long-standing Mac developer. The warning relates to the lack of macOS 10.8-specific code signing. However, including such causes problems on older supported OS.

Solution (required once only):

- When/if given the above warning, locate the Tinderbox program in Finder (in the /Applications/ folder).
- Right-click the program and select "open".
- A warning dialog will now state: Tinderbox" it is from an unidentified developer. Are you sure you want to open it? The dialog will offer 'Open' and 'Cancel' buttons.
- Click 'Open' and the app will run.
- Subsequently, the app will run as normal with no warning dialog.
- If you over-install with a newer release of Tinderbox, you may need to repeat the process (again, once only until the app is changed).

Multiple versions

For users with more specialist needs or those just wishing to test a new version before over-writing their existing version, Tinderbox can be installed pretty much anywhere (for which your account has permissions). Note that if several versions are installed, they will share the same application support folder.

Versions 6.x or later and v5.x can be co-installed but it is not recommended to open both versions at the same time.

Installing a new version

The process is as simple as:

- Close Tinderbox (if open)
- Download the DMG of the latest version from the Eastgate website: https://www.eastgate.com/Tinderbox/download.html
- Double-click the DMG to mount it. Depending on system settings this may happen automatically on download. A new Finder window opens.
- Select the 'Tinderbox.app' icon and drag it:
  - onto the the alias icon next to it to install to the default location (in /Applications). Accept the prompt to overwrite the existing version.
  - into any other location, e.g. the Desktop if installing a second version to test.
- Unmount the DMG.
- Use the new version.

It can also be useful to review the Document Settings and change any defaults that do not suit. It is certainly worth setting the user name value that is stored as the name of the creator of the note. The latter is very useful if sharing a file with other users who may edit it.

Updating an existing installation

To replace an existing version, first ensure you have closed the existing version. If required, back up the current app (or ensure you still have its installer). Then:

- mount the new version's disk image by double-clicking it
  - default install: drag the application icon onto the Applications folder alias in the disk image
  - custom location: drag the application icon onto the folder holding the current version
- Finder will ask if you wish to replace the current application; click the 'replace' button
- Tinderbox is now installed

If you wish to run more than one version, install the new version to a different folder from your existing one. However, do note that both versions will use the same preferences. If the later version has changed significantly, i.e. has differing preferences, this may cause the older version to behave unexpectedly. Versions 6 and 5 can co-exist but it is not recommended to attempt to work in both concurrently as this scenario is not tested.

Updating the application will inherit existing app preferences so if a preference or attribute default changes in a new version this may not be reflected as the app assumes the existing preference is the users choice. A change to preferences on new version install generally only occurs when completely new preferences are introduced or when the new default choice(s) no longer fit with the pre-existing ones. The user can always manually apply changes to bring preferences back in line with current defaults (assuming these are known!). For those using custom configuration files, also bear these in mind when reviewing changes to preferences.

Although you may have more than one version installed, and whilst they may both open if started you should not open more than one version/copy of the app at a time or you may get unexpected results. In other words, unless you are a beta tester and/or are specifically instructed to do so, only run one version at a time.

Installing more than one version

NOTE: Nearly all users should only ever need to have one version installed—either the current one or the last one available when their free upgrades expired.

However, occasionally it can be necessary to want to have two versions installed at once. A good example is assessing a new version to decide on purchasing an upgrade after having let cover lapse. This approach is also needed for beta testers.

So, it is possible, but do so with care. It is not a supported form of general use. Both versions will share the same app settings.

Do not try to open both versions at the same time—you will not get a warning (or a crash) but you may get unexpected results, so do not do it!

It is suggested to leave the current version of the application installed in Applications. This user then places secondary versions on the Desktop, or whatever location is chosen. If wishing to keep multiple version in the same folder, one will need renaming before being added to the folder.

As all installed versions share settings file, if using a much newer version for test, there is a chance the settings may not then work in the older version. This has not been seen, but if it does old app setting may be lost. Plan accordingly.

As aTbRef's author is also a beta tester, this document has been used editing on multiple versions of Tinderbox (albeit one version at a time) including most betas and without harm.

Note: it is worth repeating that very few users will ever need to use this configuration, but it is worth documenting for those who do.

Migrating Tinderbox to a new computer

With a current original or upgrade licence, simply download the latest version of Tinderbox onto your new compute and register as normally.

If your licence for free updates has expired the process is the same except you should install the last version that your licence allows. If you no longer have the installer for that version, email Tinderbox support and ask for the download URL for your version.

Note the limitations on using the same licence on multiple computers.

Other aspects to consider:

- Preferences file. This is in `~/Library/Preferences`. That folder will contain a preferences file and (if used) a Tinderbox user dictionary.
- Configuration files. If you have customised badges, colour menus, etc., or have a set of favourite files, check your configuration files which are at `~/Library/Application Support/Tinderbox/`. The latter folder can also be opened from the Tinderbox Help menu.

Uninstalling Tinderbox

Close Tinderbox, open the folder where it is stored (default is /Applications/) and drag the app to the Trash. Supporting files will be found in the user account Library, i.e. `~/Library/Preferences/` at:

- `com.eastgate.TinderboxSix.plist` (Tinderbox v6+)
- `com.eastgate.TinderboxSix.LSSharedFileList.plist` (Tinderbox v6+)
- `com.eastgate.Tinderbox.plist` (Tinderbox pre v6)
- `Tinderbox user dictionary`
- `Tinderbox™ Preferences`

Check, especially If you have installed custom configuration files, and remove the folder and its contents at:

- `~/Library/Application Support/Tinderbox/`

If you have installed the app at a global level (for all accounts) you may need to check:

- `/Library/Preferences/`
- `/Library/Application Support/Tinderbox`

Any support files in these locations may deleted (or archived off elsewhere).

Demo & Viewer mode

If Tinderbox is installed and not registered it runs a demo. In demo mode the app is fully functional except for the abilities to add more than 24 new notes.

This means the demo can be used as a viewer for TBX files as there is no cost to downloading and using the demo.

The demo can open and edit TBXs with more than 24 notes—it just cannot add more than 24 new ones. Thus, if trying out the demo it can be useful to download various demo TBXs just to see how larger documents look and to play around with their arrangement of notes.

The demo does not expire and on application of a registration code becomes the full application.

Tinderbox documents

- TBX Documents
- What does a Tinderbox document contain?
- TBX file sizes

TBX Documents

A Tinderbox document is a saved as a single file, with a '.tbx' extension, and thus are often referred to as 'TBX' documents.

Tinderbox can create and save multiple TBX documents, limited only by storage space available. It can open multiple TBX documents at once, limited only by the host computer's resources. When Tinderbox is started, it will open any documents open when the app was last closed; this behaviour follows the host OS's System Preference/General 'Close windows when quitting an app', so may vary if that preference is changed.

Tinderbox also tracks recently opened documents on a sub-menu of the File menu. If multiple documents are open, the application's Window menu list all open windows from all documents. The latter listing does not indicate the document to which each document belongs.

Any non-default Document Settings and other customisations (link types, custom colours, user attributes, etc.) are stored in the TBX.

A Tinderbox TBX document stores its data in XML format; for some users with XML skills this can provide another way to interface with, or view, the data. Users altering TBX documents by editing the XML source should note that such work is not formally supported as an editing method, i.e. do not expect that Tech Support will be able to fix user mistakes editing source XML.

An open document can display one or more windows and tabs.

See more on the XML structure of TBX documents.

What does a Tinderbox document contain?

A Tinderbox document is a self-contained storage of your notes and information about them including:

- every note (including any embedded images)
- every agent
- every adornment (including picture adornments)
- any non-default Document Settings
- user attributes
- any user added/customised link types, defined colours, stamps and macros
- information about open document windows and tabs

Note that embedding images inside a document—whether as map adornments or in notes—will likely add noticeably to the overall file size of the document.

The document data is stored as XML. Those whose understand XML can (at their own risk). Use XML tools and techniques to read, extract or even modify data. Do not attempt to edit XML data for Tinderbox files that are also currently open in Tinderbox but rather close them first. Users altering TBX documents by editing the XML source should not this is not formally supported as an editing method; i.e. do not expect Tech Support to clear up after user mistakes. See more on the XML structure of TBX documents.

Either to save space for long term storage or for safe transmission (e.g. as email attachments), Tinderbox files compress quite well. To make a ZIP format archive of one or more documents, select the Tinderbox file(s) in a Finder window, right-click the selection and choose the 'Compress' option. A Zip file will be created alongside the selection. The name of the file is either the filename plus a '.zip' extension or 'archive.zip' if multiple files are compressed. The name of the compressed file can be altered without affecting the contents (though leave the .zip' extension!).

TBX file sizes

Generally most TBX files are pretty small—a few MB or less. There is no hard limit to size to a practical limit will be resources getting maxed out either in the app or the host OS.

Some of the ideas behind Tinderbox do have limitations at large scale. Spatial hypertext works well with dozens or hundreds of notes. If you have millions of notes, it's harder to make any sense of the map, and it's hard to know where a ne note ought to go. Hyperbolic view can likewise become very busy with large numbers of notes.

Adding images to notes (i.e. to their $Text) or as image adornments on maps will have far greater effect on document size than ordinary textual content (in $Text or as attribute values). Occasionally, users will discover hard limits by building

documents larger than anyone had tried to build before but there is no fixed limit.

As a general guideline, Tinderbox is comfortable with hundreds or a few thousand notes. If needing millions of notes, a database application is really the correct. In between those sizes is a grey area. If TBXs are operating smoothly, size shoul not be a cause for concern.

Support

Reporting Problems

In the event you have problems with the application or with particular TBX(s), formal tech support is offered via email at info@eastgate.com.

Bear in mind that the online forums are user-to-user (i.e. not formal Eastgate support), so people helping there are fellow users who do not have access to the internals of the app. They can however be a useful source of advice on the how an why of using Tinderbox and specific techniques.

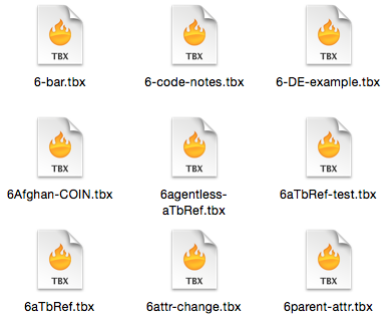
Tinderbox files have no '.tbx' extension

First check you have not set Finder's preferences to hide file extensions: see how. If you have, you will not see any file extensions!

If a Tinderbox file has no '.tbx' extension, even when extensions are shown, then there are some possible causes:

- If you used Tinderbox before July 2005 (v2.5), the application did not add extensions and relied on old-style Mac OS resource fork info. All such legacy Tinderbox files should now be given a '.tbx' extension.
- You somehow deleted the extension. If you are certain the file is a Tinderbox document, then give it a '.tbx' extension and try opening it
- The file is not really a Tinderbox file. Either—if confident to look at XML—look at the source code in a text editor. A Tinderbox file will mention the app name on line #2 of the code. Or, send a copy of the file to support (info@eastgate.com).

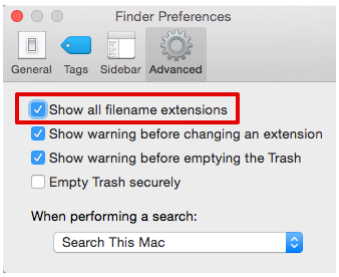
Now your files have the correct extension they should show a Tinderbox file icon and you should be able to open then simply by double-clicking their file icon. If not, see here.



Showing or hiding filename extensions via Finder

The setting is normally on the Advanced tab of Finder Preference. To do this:

- Select Finder
- Open the Finder menu from the main menubar (top of screen)
- Click the 'Preferences…' menu option
- The Finder Preferences dialog will open (illustrated here)
- If not already selected, click the 'Advanced' button in the dialog's toolbar.
- Ensure the top tick-box 'Show all filename extensions' is ticked (as shown)
- Close the Finder Preferences dialog.



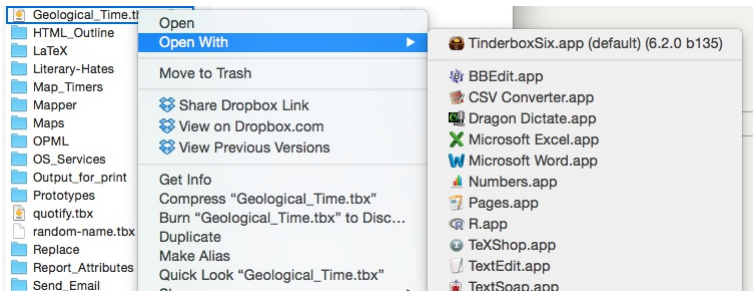
Your Tinderbox documents now show a '.tbx' extension.

Is desired, you can toggle this option back of but all extensions will be hidden. It is certainly advisable to have file extensions visible whilst troubleshooting issues with Tinderbox files being opened correctly on double-click.

Note that Finder itself can still 'see' any file extensions even if you have told it to hide the file extensions.

Beyond the scope of this article is that depending on how much fiddling you have done in this area, some extensions may not be hidden, even when Finder has been told to hide extensions. This is because unwittingly you will have told Finder that this particular file's extension or this extension in general must always be shown. That is fixable; look in Finder's Help for how.

Tinderbox files do not have a Tinderbox icon



Tinderbox documents should show the file icon illustrated here and open in Tinderbox if double-clicked. If the icon is for some other application or just a plain white page, Finder believes the file is not associated with Tinderbox. In such instances double-clicking the file will cause the other app to try and open the file or finder will ask the user to choose and application to use. This can be fixed.

In such a case, first check that the file has a '.tbx' file extension.

If there is an extension but still the wrong icon it is likely that, due to some unintended error, you have told Finder to open this document in some other application and/or the later has now assumed 'ownership' of the document. Such and error can happen for a single file or all files of a given file extension.

To fix, locate your file in Finder:

- Use Finder's Get Info on the file (Cmd+i).
- Click the 'Open With' disclosure triangle to expand an application list.
- Choose Tinderbox as the application you want all files of this type to open with. If Tinderbox is not already listed as suggested choice, use the "Choose…" option and find Tinderbox in your /Applications folder.
- Click "Change All" and then "Continue" when the confirmation dialog appears. This last step tells Finder to open not just this file, but every file of this type with Tinderbox.

The icons for your Tinderbox file(s) should now regain the expected icon type and open correctly in Tinderbox when double-clicked.

Tinderbox files don't open on double-click

If you definitely know the file is a Tinderbox document and are in a hurry:

- Open Tinderbox
- In Finder, drag the document's file icon onto the Tinderbox application icon in the Dock.
- The file should open

If this does not work, or when you have more time to fix things properly, follow this procedure to (re-)inform Finder how to handle files with a '.tbx' extension.

Read what to do if you get a " Tinderbox was unable to parse this file " message.

Checking or resetting old System attribute defaults

Over time, in successive Tinderbox releases, some System attributes' defaults have changed. These changes will not show up for existing users if their Tinderbox Preference file already defines the values, i.e. some but not all values are masked by existing Preferences.

If it is desired to reset/review attribute defaults:

- Create a new Preferences file (see how).
- Create and save a new Tinderbox document.
- Save the new Preferences file to a new location.
- Close the Tinderbox and restore the old Preferences file.

The old and new Preferences file can now be compared for differences. Attribute defaults set directly via the app can be viewed by opening the saved new TBX in a text editor (the source is XML code).

For long-term users with TBXs originally created in older versions, the issue is the latter are set and fixed at doc creation. New app defaults are not applied. Comparing the attributes section of the new TBX with one or more existing files may show up changes that can then be applied, if required, to the existing document(s) by resetting the document attribute defaults. Note that some attributes, being set via Document Settings, should be altered by that mechanism.

Message: Tinderbox was unable to parse this file

Very occasionally, on opening a file you may get a dialog with a message like:

`Tinderbox was unable to parse this file. It may be damaged, or you may need a newer version of Tinderbox. The XML parser said : not-well formed (invalid token) (line 1232).`

Whilst the exact line number stated at then end will vary, Tinderbox is telling you that the TBX file's data includes something it can not understand at the line (as referred to) in the source XML of the document. When Tinderbox opens a file, it has to read in all the XML data stored in the TBX document file. If this data has been corrupted in some way such that it is no longer in valid XML format, Tinderbox cannot read past that point and gives this error message.

How might such a thing occur? The error is unusual but causes include copying/pasting text from web pages that mis-declare their encoding, such as with web pages where quote marks show as question marks (e.g. character) or less common accented characters as pairs of random characters. It is not always possible for the Tinderbox to detect that the data it is passed is not what it declares itself to be. This can cause the data to be stored inappropriately in Tinderbox, although the effect does not tend to surface until the document is next opened—which for some users can be hours or days after the triggering event. More technically, data is saved in a form that's not intelligible to the XML parser used to read the data when opening a TBX file. Similarly, AutoFetch of data from badly-encoded pages/feeds/sources can ingest data that has the same effect as above.

The solution is to send the file (or better, a zip of it) to Tinderbox support (info@eastgate.com) and they should be able to fix it and return the document. At times the fix may result in the loss of all or part of $Text (or the affect attribute(s) of th affected note(s), but generally just the offending characters can be excised.

If you are doing some task where this happens more than once (perhaps you have very 'dirty' source material) then ensure you have reviewed your settings for back-ups and autosave. You can always change back to your defaults once the problem is resolved.

For the more technically minded…

If you are confident using a text editor and looking at XML source code you can have a go at fixing this yourself. If you do not have a text (code) editor—do not use TextEdit—a good free option is BBEdit. Now:

- Make a copy of your broken TBX and give it a new name indicating it is (will be) the 'fixed' version.
- In your text editor turn on line numbering (see your editor's manual if unsure how). It is also a good idea to turn on 'invisibles' so you see an on screen character for spaces, tabs, control characters, etc. Turn line wrap 'off', at least until you have located the problem.
- Open the TBX in the text editor app
- Scroll to the line mentioned in the original Tinderbox error message.
- Examine that line in the code for anything untoward (use the window's line numbering to find the right place). If your editor has syntax colouring it should stop working around where the error occurs.
- If it is not self evident as to the exact error, try removing the whole attribute value containing the bad line. Most commonly these errors occur in $Text. In such a case you might delete the value of that $Text, i.e. all code from the last tag preceding the error position and the first tag following it. You can always cut/paste the excised data to a text file for temporary safe keeping.
- Save the file close and re-open in Tinderbox. You may get another error (likely caused by the same thing in a different note). Rinse and repeat the previous steps until the file opens. You can then review what's been lost and the likely probable cause. If you think you know the latter, it is probably worth letting Tinderbox support know in case it is a cause they can predict and guard against in the future.
- If you have no joy at all delete your edited file and sent a copy of the original TBX to support.

For TextWrangler and BBEdit users a slightly more hands-off approach is offered via those apps' "zap gremlins" option. It should prune non-XML-safe characters from the file though you will not know exactly what that is. You will just have to look at the resulting TBX file and guess.

Where are crash and hang logs found?

Tinderbox is generally very stable but that does not mean you may not find some data from the app can not ingest from drag/drop or find some configuration next really designed for.

Should the unfortunate occur, and the app crashes, macOS will create a crash log which it can be helpful to Tech Support in resolving the cause. Such logs are found inside your home folder at:

`~/Library/Logs/CrashReporter/`

or

`~/Library/Logs/DiagnosticReports/`

Within that folder look for files with names in the format " `Tinderbox_yyyy-mm-dd-hhmmss_macbookpro.crash`". Basically, the filename retains the date/time of the crash.

If reporting a crash, attach a copy of the log file(s) to your email. It is also very helpful to support is you can describe the events lading to the crash and/or a step-by-step method to trigger the crash.

Occasionally an app may not crash but become very busy and have to be stopped (by the user or the system). In such cases a hang report may be created. These are stored in in the main, system-wide Library:

`/Library/Logs/DiagnosticReports`

The logs use a similar naming method to the crash logs but use a '.spin' or '.hang' extension.

A more detailed tutorial on crash/hang logs for Tinderbox is available elsewhere.

Resetting Preferences

It is most unusual to get a problem with Tinderbox's preferences. The file contains the app-installed preference defaults, as subsequently modified. However, if a problem is suspected, a number of things may be tried.

For the non-tech user…

…who does not like looking at under-the-hood stuff:

- Locate the Preferences file (for location see below).
- Ensure you have your Tinderbox licence code to hand. NB: the licence is needed only if you cannot roll back to existing Preferences after testing.
- Close Tinderbox, if open.
- Drag the "Tinderbox™ Preferences" file to another place such as the Desktop.
- Restart Tinderbox, which causes a new Preferences file to be created. If demo mode does not enable you to test as needed, you will need to re-apply your licence info via the.
- Test Tinderbox and if no change:
  - Close the app, and copy the old Preferences back over the new set.
  - Contact technical support.
- Or, if the problem goes away:
  - Continue using the new preferences.
  - Re-apply your licence key if you have not already.
  - Check the Tinderbox Preferences dialog in the app and make any customisations to restore your preferred settings.
  - Archive or delete the old preferences file.

For the more technical user…

…the Preferences are a text file so you can open it in a proper text editing (not a word processor!) and look for any obvious corruption. The file includes the Recently Used Files list and in earlier v5 versions some more exotic filenames/paths caused problems. Deleting the corrupt path listing could normally fix Preferences (assuming no other errors). The latter problem seems to have been resolved in later v5 releases.

If confident using a text editor and if forced to use a new Preferences file, the old file can be viewed/compared and used as a reference to re-apply any desired non-default settings via the Tinderbox Preferences dialog.

Locating the Tinderbox Preferences file

Tinderbox's Preferences files normally normal located at:

`~/Library/Preferences/Tinderbox™ Preferences`

Users wanting to share Preferences between several users of the same Mac may elect to copy/move their Preferences file to:

`/Library/Preferences/Tinderbox™ Preferences`

Tinderbox checks the latter folder for a Preferences file and if found, uses it in preference to any local version.

Image formats for embedding

Generally, Tinderbox expects bitmapped images of the JPG (JPEG), GIF, or PNG format. Tinderbox will likely accept other older formats such as BMP, PICT but these are best avoided if possible.

Formats used as wrappers for bitmap content, are not supported directly, but can be used indirectly as discussed below. For instance, more complex bitmap formats such as TIF/ (TIFF) which can support a wide of complex internal formats (hidden from the ordinary user) may work depending on the internal structure of the file.

In general, if you can open an image with Preview (or some image editor such as Graphic Converter, Lightroom, Aperture or Photoshop), you can select a portion of that image, copy it to the Mac clipboard, and paste it into Tinderbox as an image adornment (though not as an image note). The latter applies to PDF and TIFF files, among others.

Apple Preview and image editors can also convert TIFF files to JPG or other simple bitmap formats files that can be dragged into Tinderbox as image notes.

Preferences & Document Settings

Preferences currently only relate to registration information and Tinderbox versions.

The old Preferences have now become Document Settings, applying only to the current document. Many Document settings act as default values for System attributes. Whilst a few System attributes are calculated on the fly and thus read-only, all other attributes can be manually set a doc-level default.

Preferences have been significantly revised and reduced from pre-v6 designs.

- Preferences
- Document Settings
- Attribute inheritance of preferences and settings

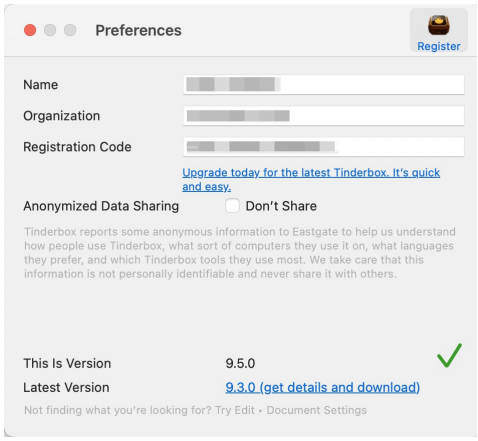
Preferences

Preferences has the following tabs:

- Register

Register

The Register tab holds the following information:



- **Name**. The user's name as per the registration details (user's details greyed out in screen-grab).
- **Organization**. Optional organisation information as per the registration details (user's details greyed out in screen-grab).
- **Registration Code**. The licence code as supplied by Eastgate (user's details greyed out in screen-grab).
- (Remaining months of free updates.) The number of months remaining of access to free updates or a reminder. Or, if expired there is a click-able link to an Eastgate webpage for obtaining upgrades.
- **Anonymized Data Sharing**. Tinderbox anonymously reports some simple usage statistics to Eastgate make features easier to discover (see longer discussion). The default is to allow sharing. Un-tick the box to prevent sharingTbRef's author recommends accepting the default, of sharing.

This is Version. The version of Tinderbox currently being used.

Latest Version. The latest public release of Tinderbox.

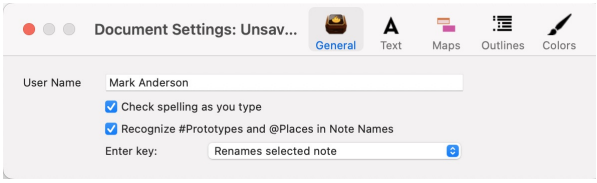
Tick/Cross. The tick symbol indicates the supplied registration code has been successfully applied.

A text prompt indicates where to find Document Settings.

Document Settings

Document Setting allows the customisation of various application defaults. Most of the setting are also used as defaults for System attributes. The title of the Document Settings dialog is the filename of the current Tinderbox Document when it is opened. If multiple TBXs are open, it is possible to open a Document Setting dialog for each open TBX.

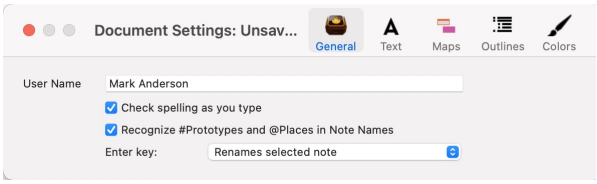
The controls for settings are divided into a series of tabs:



- General
- Text
- Maps
- Outlines
- Colors

General

The General tab has the following controls:

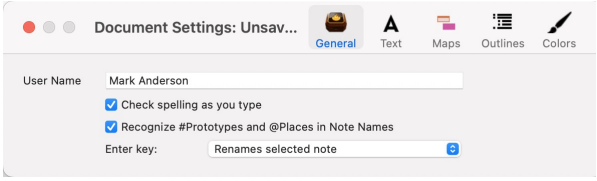


- User Name
- Check spelling as you type
- Recognize #Prototypes and @Places in Note Names
- Enter key

User Name

User Name. Default value: 'system'. The name entered here is used by the $Creator attribute. It is useful to alter this from the default if sharing a Tinderbox file or it is being edited by more than one person.

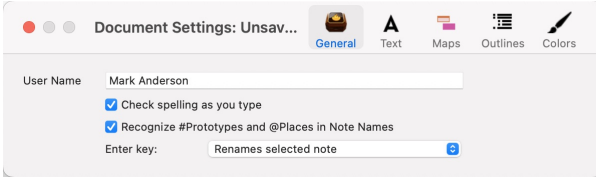
In the latter case each user would start their edit session by setting their name (and possibly resetting this preference when done).



Check spelling as you type

This toggles the underlining of (possible) spelling mistakes. Default: ticked (on).

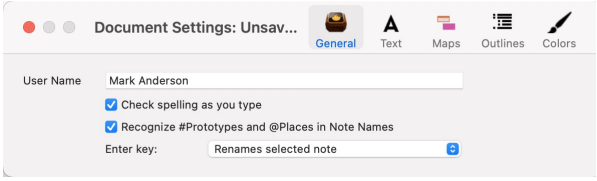
On low-powered systems and/or with documents with large amounts of $Text leaving this off may help with performance. Otherwise, leaving the setting turned off simply avoids the red underline spelling prompts. The latter are not helpful if showing the app in a presentation or if using multiple languages in $Text.



Recognize #Prototypes and @Places in Note Names

This toggles parsing of new noted names for special handling of # and @ syntax, used for setting prototypes/locations.

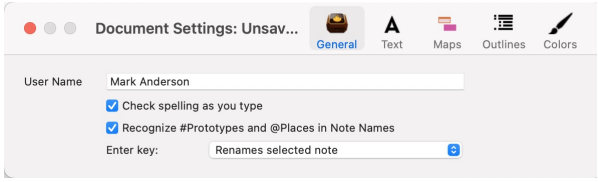
This toggle is 'on' (ticked) by default, thus literal use of # and @ in new/edited note titles may be problematic unless the feature is toggled off.



Enter key

Enter key (^↖). There is a choice of outcome when pressing the Enter key (or Fn+Return (Fn+↩) on modern keyboards):

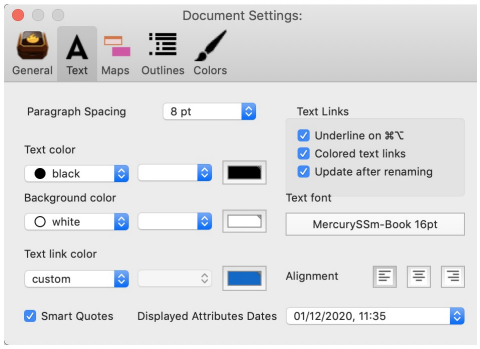
- **Renames selected note**. (default) This reflects pre-v6 behaviour when Renaming occurred in a separate dialog.
- **Opens Action Inspector**. This opens the Action Inspector, with the Action tab selected. It is probably a more useful choice for most users.



Text

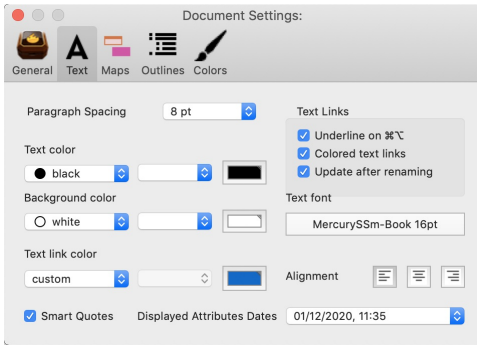
The Text tab controls various defaults relating to note text ($Text):

- Paragraph Spacing
- Text color
- Background color
- Text link color
- Smart quotes
- Underline on ⌘⌥
- Colored text links
- Update after renaming
- Text Font
- Alignment
- Displayed Attributes date format



Paragraph Spacing

Paragraph Spacing. Default = 8pt (was 2 pt in older TBXs). Choice of preset values: none/2/4/6/8/10/12/14pt. This controls the amount of white space between paragraphs and sets the default for $ParagraphSpacing.

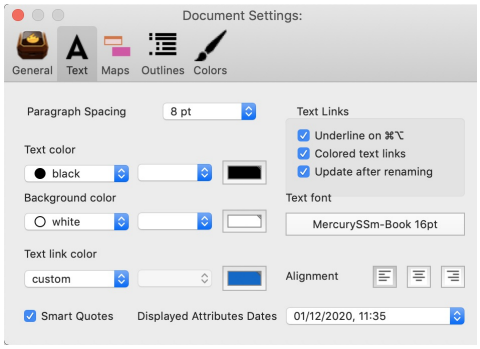


Text color

Text color. The *default* colour of the text used in (all) note text. Default is: #000000 (black). Inherited by $TextColor attribute.

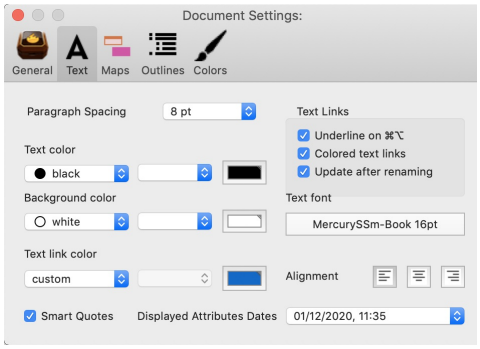
Once text is added to $Text, its style—at time of entry—is saved in the TDF data and so it is unaffected by subsequent changes to $TextColor. Re reset a note's $Text to use a (new) $Text colour, select all $Text—or that to be repaired—and use menu Format ▸ Style ▸ **Standard Font**. (N.B.there must be a $Text selection for this to work)

To re-colour (selections of) existing $Text, i.e. to a colour *different from* $TextColor, use either the Format ▸ Style or Format ▸ Font menus.



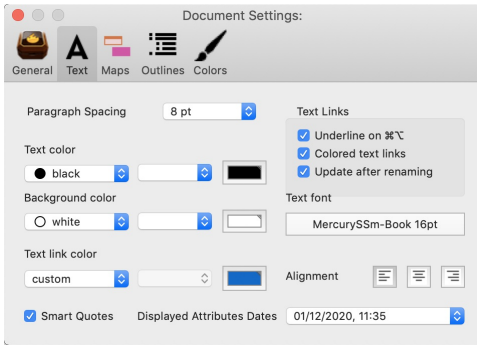
Background color

Background color. The background colour of (all) note text windows. Default is: #ffffff (white). Inherited by $TextBackgroundColor attribute.



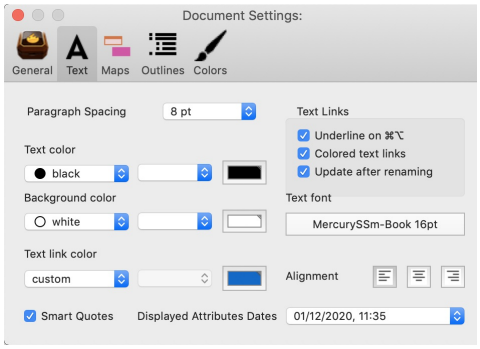
Text link color

Text link color. The colour used (optionally) to mark link anchors in $Text. Default is #0066CC (blue).



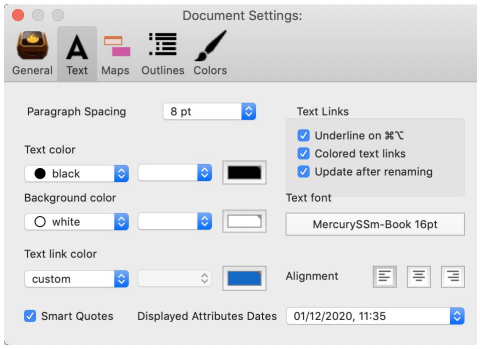
Smart quotes

Smart Quotes. This controls whether straight single or double quotes are automatically replaced by 'curly' typographical quote symbols; it also controls the automatic conversion of two hyphens (--) to an en-dash (–). This setting forms the document default for $SmartQuotes. Default: ticked.



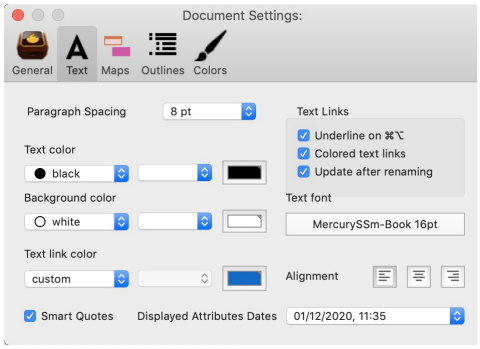
Underline on ⌘⌥

Underline on ⌘⌥. In notes, when this option is ticked, holding down both the Command & Option (⌘+⌥) keys together causes any links in current note window's text to be underlined (default= ticked). This is useful for discovering the exact extent of a text link when there is no distinguishing text colour for the note or try to establish is the link area has opening/trailing white space included.



Colored text links

Colored text links. This controls whether link anchors in $Text are coloured using the text link colour. Default: ticked.

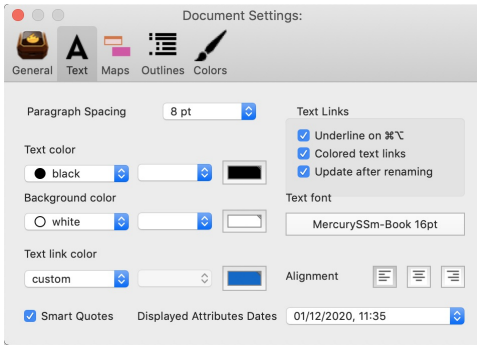


Update after renaming

There is an option to disable automatic updates to all note $Text text link anchors when note titles ($Name) are edited. This is useful for those using *zettelkasten* or wiki type links where link anchors are exact target note names.

Default is ticked (`true`). The setting is inherited by $UpdateTextLinksAfterRename, so can be set differently per-note (or via a prototype).

In large, well-linked documents that do not use note titles as link anchor text, this option is of less use as it is essentially unneeded so nugatory background work for Tinderbox to be doing.



Text Font

Text Font. The name and size of the default font for note text ($Text). This sets the default for $TextFont and $TextFontSize. Text size is set directly in Points.

Default is MercurySSm-Book at 16pt. Previously it has been variously: HoeflerText-Regular, Cochin, Lucida Grande, and originally Geneva.

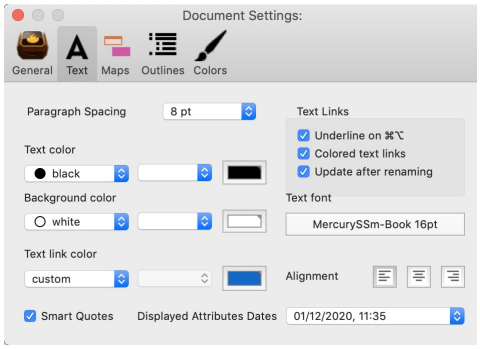
As note default text size is also set via the OS Fonts dialog, a wider range of default sizes is possible than in early versions of Tinderbox.

When the default text font is changed, Tinderbox scans the text of every note in the document and changes each usage of the former text font to adopt the new font family while retaining the current size (see more).

Changing the default text font in Document Settings now updates both the text font and font size in notes using the former default font and font size.

Edge case issues:

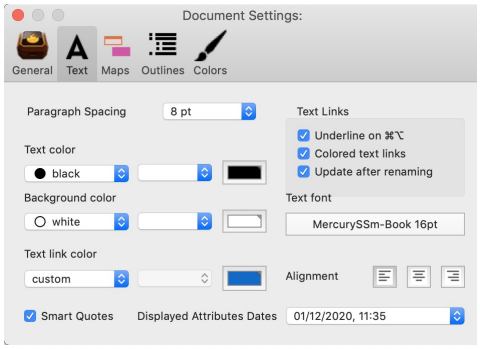
- what happens to text styled to other than the default font face/size? Tinderbox does its best to handle bold and italic styles consistently.
- what happens to text already at a different size? Tinderbox does not resize text at different sizes., i.e. non-normal sizes
- if the first character of an existing note is not using the document' default Font/Size, do other parts of $Text using the default still update? Tinderbox scans the entire text, and update text that matches the old default.



Alignment

Alignment. The justification style for text in note text windows. Sets the default for $TextAlign. Choices are icons representing:

- **left**. (default).
- **center**.
- **right**.



Displayed Attributes date format

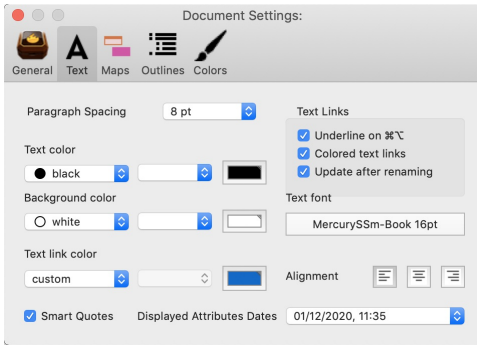
A pop-up list offers a choice of 3 doc-wide default formats for the values of Date-type attributes shown in a note's Displayed Attributes table.

The default is to show the local OS' short date format, then a comma, and then the hours:minutes.

This choice can be overridden at document or note level by setting a date format string in $DisplayedAttributesDateFormat:

- To set a document-level default that is not one of the 3 built-in defaults listed here, use the document Inspector's system tab, select $DisplayedAttributesDateFormat and replace the **default** with the desired format string (without enclosing quotes). This will now be the format inherited in the document and the control in Doc Settings will be greyed out. The attribute's doc-level default is " " so as normally to inherit the format set in the above pop-up.
- To set at prototype or note level simply set that note's $DisplayedAttributesDateFormat.

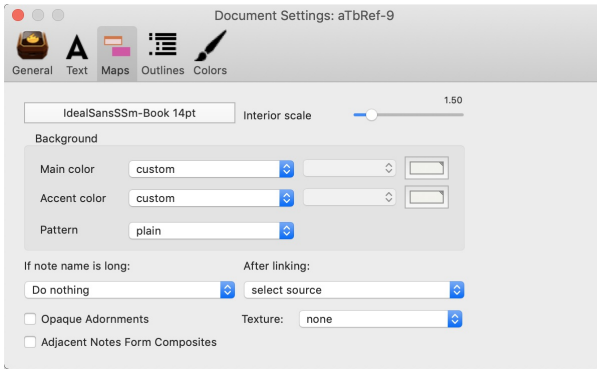
The design intent here assumes a customised format will more likely be set in a prototype than at doc level, but the latter is possible (as described above).



Maps

The Maps tab controls various defaults relating to Map view (although some may be used in other views too):

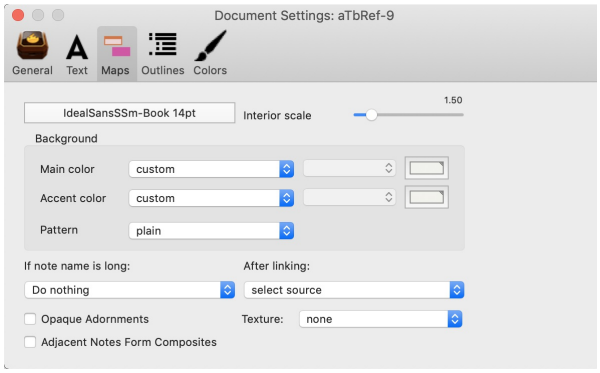
- Map Font
- Map interior scale
- Background Main Color
- Background Accent Color
- Background Pattern
- If note name is too long
- After linking
- Opaque Adornments
- Adjacent Notes Form Composites
- Texture



Map Font

Map Font. This sets the font to be used for titles in main views and sets the default for $NameFont (font) and both $MapTextSize and $OutlineTextSize (font size).

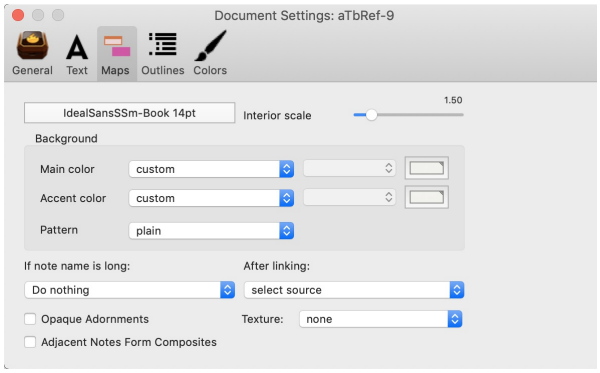
The default is IdealSansSSm-Book (at 14pt).



Map interior scale

Map interior scale sets the default for the $InteriorScale attribute. This number is the factor by which notes inside a container are smaller than notes elsewhere in a map. The default value is 1.5. If InteriorScale is 2, then interior (child) notes are half their normal map size.

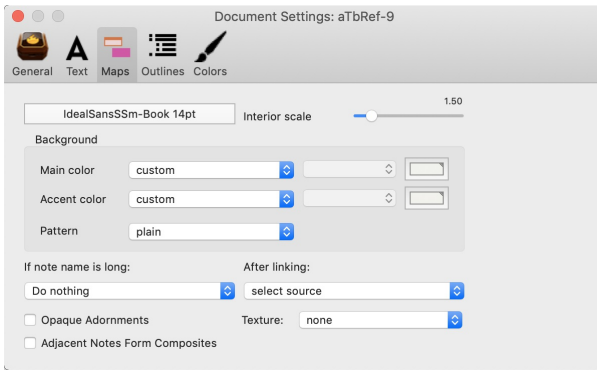
This scaling helps with viewing the content of a map container's viewport.



Background Main Color

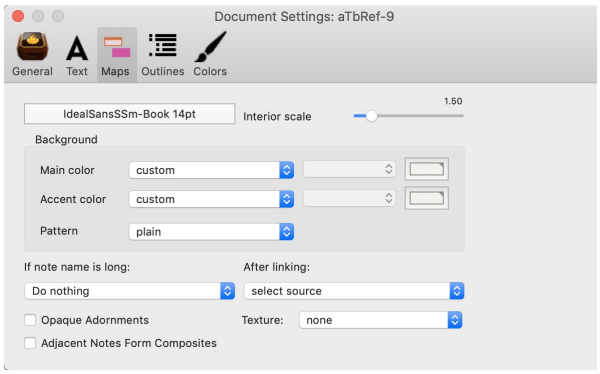
Background Main Color. Specifies the background colour to be used for Map views and all other view windows, i.e. all windows except Note windows and Tinderbox dialogs/palettes. Inherited by the $MapBackgroundColor attribute. Default colour value is 'gray'.

This also sets the primary colour for map background patterns (in Map view only, not in other main views).



Background Accent Color

Background Accent Color. Specifies the accent colour used for patterns in Map view backgrounds. Inherited by the $MapBackgroundAccentColor attribute. Default colour value is 'gray' (same as $MapBackgroundColor).

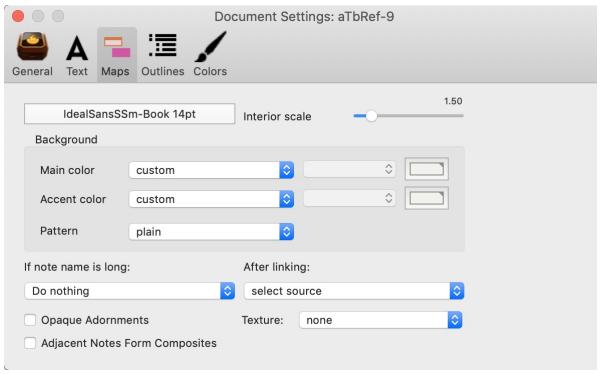


Background Pattern

Pattern. This allows a map icon style pattern to be applied to he background of map views, using the main and accent map background colours. The allowable pattern values are:

- plain (i.e. no pattern. Default value)
- gradient
- diagonal
- radial
- cylinder
- lines

The value is stored in $MapBackgroundPattern.



If note name is too long

If note name is too long. This option lets you ask Tinderbox to automatically expand notes that have been newly-created or renamed in the map view, if their titles are too large to fit in the current note rectangle.

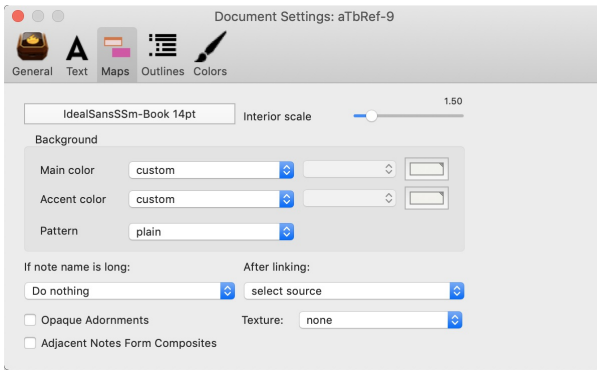
This new options is especially nice for quickly making notes in meetings. The choices are:

- **Do nothing** (default)
- **Expand horizontally**
- **Expand vertically**
- **Expand proportionately**
- **Use smaller type**.

The **Expand proportionately** option tries to use a smaller font size in order to fit in the available space. Manually resizing the note will increase the font size again, up to (but not exceeding) the preferred font size.

The **Use smaller type** option uses smaller type but it does not resize the title text if the map icon is later re-sized. This option will not make the type larger than the default size if the note is later made larger. Adornments are not affected by this document setting.

If this setting is used and a note is added in a view other than map view the appropriate map view is updated appropriately even though not currently open.



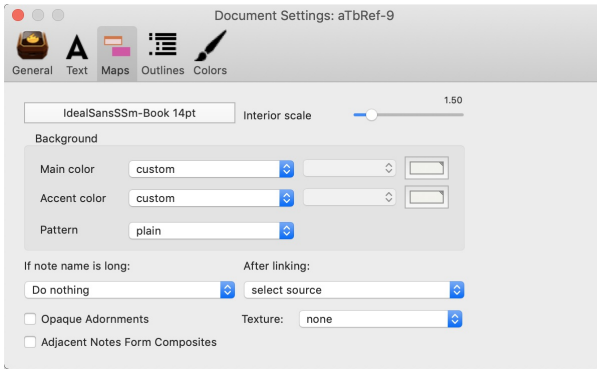
After linking

After linking. This controls whether focus shifts after creating a link. Values:

- **source**
- **destination** (default)

The default setting is best for data input. Making a link to a note, selects that note as a result of completing the link, allowing raid traversal of a new link chain or entry of data into the newly linked note.

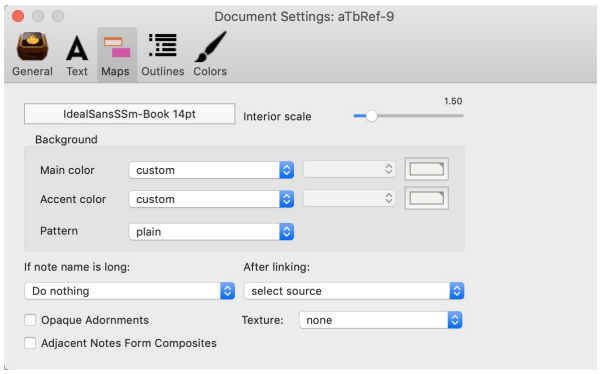
The 'source' option is more useful when reviewing a new document. In this mode it is often desirable that in creating, or correcting links, that the focus does not shift from the current note.



Opaque Adornments

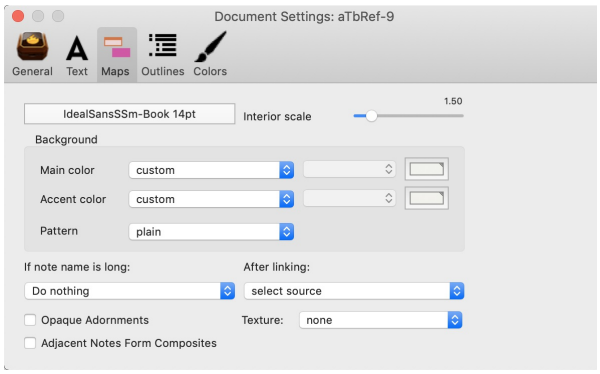
Opaque Adornments. If ticked, adornments are completely opaque. Normally, they are drawn slightly translucent as this helps when adornment need to overlap. Default: un-ticked.

This setting applies to all adornments in the document and cannot be set per adornment.



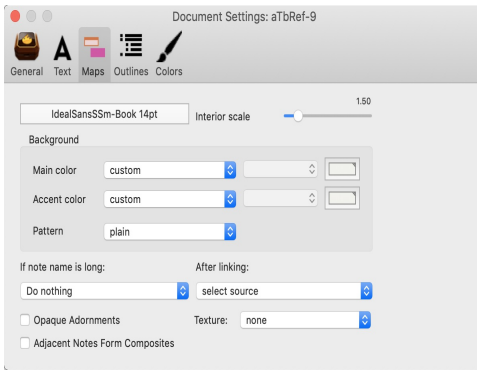
Adjacent Notes Form Composites

This allows the default document setting for $NeverComposite to be altered. By default is unticked meaning notes in new will not form accidental composites.



Texture

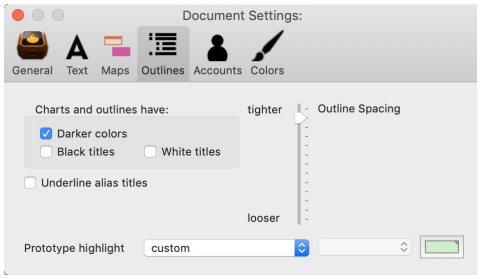
This pop-up allows pre-setting the default texture for new notes, i.e. the document default for $Fill. The choice is also pre-selected in the Appearance Inspector's Interior tab's Texture control.



Outlines

The Outlines tab controls various defaults relating to Map view (although some may be used in other views such as Chart):

- Charts and outlines have:
- Underline alias titles
- Prototype highlight color
- Outline Spacing



Charts and outlines have:

Charts and outlines have:

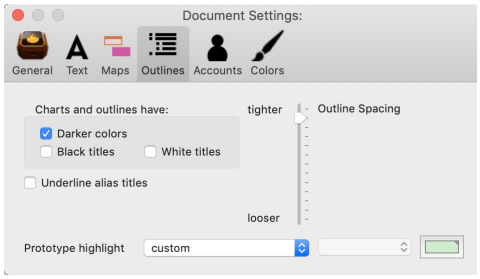
Two features to assist with legibility when using lighter colours and colour schemes. Some colours (e.g. yellow, pastels) are hard to see against a light background. The effects can be applied singly or together.

- **Darker colors**. This darkens the colours used for drawing outline text & icons. Default is 'on' (ticked, previously it was 'off'). However, this does make most colours harder to tell apart, and it is worth trying out documents with this setting off. In documents that use 'Darker colors' in outlines, the selection highlight also uses the darkened colour.
- **Black titles**. This enforces the display of all note titles in charts and outlines in black, without regard to the note's $Color value. Default is 'off' (un-ticked).
- **White titles**. This enforces the display of all note titles in charts and outlines in black, without regard to the note's $Color value. This can help with use in OS dark mode. Default is 'off' (un-ticked).

NOTE: although both black and white title options my be un-ticked, only one may be ticked at a time, i.e. you can choose to have:

- no black/white title option set
- black
- white

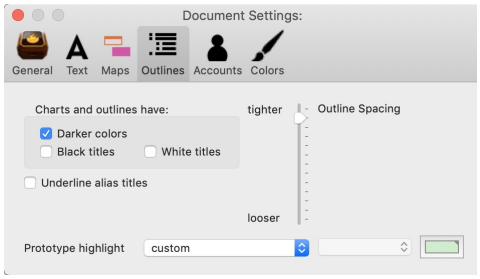
… but not both black and white (which would not make sense anyway!).



Underline alias titles

Underline alias titles. This allows aliases in Outlines (in fact all major views) to have their titles underlined in order to make them more conspicuous as aliases. Default: un-ticked.

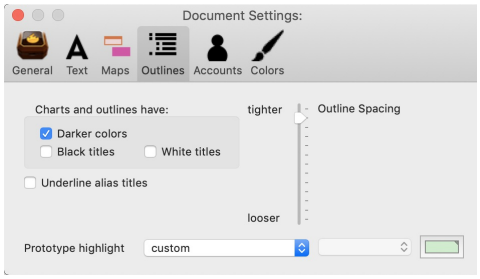
It is also useful for pre-v6 files where the specified $NameFont may lack an italic font variant: see more.



Prototype highlight color

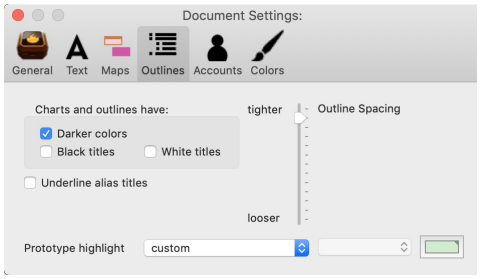
Prototype highlight color. This preference sets the colour of the circle shown behind the Outline view note icons of prototype notes. The default is very light. If prototypes do not show a mark behind them try a darker value for this attribute. To avoid the mark showing, set the value the same as the view's background colour.

The default value is #DDEEDD and it is inherited by the attribute $PrototypeHighlightColor.



Outline Spacing

Outline Spacing. A control for leading in Outline view, i.e. controlling additional space between outline items.

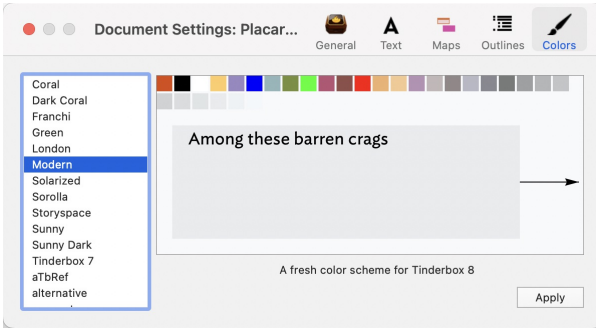


Colors

The Colors tab controls selection of the current colour scheme in use in the document. The default is 'Modern'. The list shows both built-in scheme and any stored in the Tinderbox support folders. Read more on colour scheme files.

The left list box lists available scheme. When selected, a sample bitmap (if supplied) illustrating the scheme is displayed on the right.

Note that from v8.0, Tinderbox's colour schemes have been given an overhaul. In support of these improvements, several old colour schemes have been dropped. This means new documents will have a different set of pre-defined colour schemes. Note that for *new* documents created in v8+ on OS versions (10.14+) with dark/light modes the document is given a *different* default colour scheme. The current built-in schemes (and defaults)



- Coral
- Dark Coral (v8+ default for new TBX files in OS dark mode)
- Franchi
- Green
- London (add v9.5.2+)
- Modern (v8+ default for new TBX files in OS light mode)
- Solarized
- Sorolla
- Standard
- Storyspace
- Sunny
- Sunny Dark
- Tinderbox 7 (default v6-v7)
- [user colour schemes in the user's 'color schemes' app support folder, if any, list at the end]

The colour scheme format has been extended to accommodate a wider range of appearances. Experimental support has been added for colour schemes that use dark background colours, in part to determine how difficult this may prove. Tinderbox has always assumed that map and text panes had fairly light backgrounds, but there has clearly some interest in dark colour schemes, especially since the macOS 10.14 introduced a dark mode.

'Modern' is the light mode default, replacing 'Standard' in older versions; in this scheme, the map background colour, and the colours 0-9, are slightly bluer. "Dark Coral' is the dark mode default.

For those with TBXs pre-dating v8 and who work in dark mode, applying 'Tinderbox 7' should give a better result. The issue this solves is that older colour schemes were not aware of the colours changed when toggling OS dark/light modes and so did not (re-)set some colours accordingly. All the schemes above have been updated to allow for this issue.

When applying a new colour scheme, if the scheme changes $TextFont, notes that use the old (scheme's) text font will be changed to use the new scheme's text font.

Attribute inheritance of preferences and settings

Settings controlling the look or behaviour can be set at various levels. Consider something controlled at note level via an attribute. Working back up the inheritance chain, this is how the value may be derived, if not explicitly set at a given level The lowest level (and the higher in the list below) at which a value is set dictates the value used at note level:

- within the TBX

- Attribute value for this note
- (Attribute) value inherited from a prototype
- (Attribute) value inherited from a doc-level user-set default.
- Value from Document Settings (i.e. this TBX only)

- at the TBX creation

- Value from Tinderbox built-in document defaults

The latter can, in some cases, be also be modified by the user editing configuration files, or at the extreme by modifying the app package's configuration files though this is not suggested!

Objects in Tinderbox documents

Tinderbox can open multiple documents. Each discrete document is saved as an individual TBX file.

Within a Tinderbox document everything essentially consists of notes and links.

At the document storage level, 'notes' are of 4 different types:

- item (the basic note)
- alias
- adornment
- agents

In addition, the above may have other discrete functional characteristics such as an outline separator note, that are set via the object's attributes.

More on the various special forms of note object:

- TBX Documents
- Notes & Containers
- Notes as agents
- Creating notes and agents
- Deleting notes and agents
- Aliases
- Composites

TBX Documents

A Tinderbox document is a saved as a single file, with a '.tbx' extension, and thus are often referred to as 'TBX' documents.

Tinderbox can create and save multiple TBX documents, limited only by storage space available. It can open multiple TBX documents at once, limited only by the host computer's resources. When Tinderbox is started, it will open any documents open when the app was last closed; this behaviour follows the host OS's System Preference/General 'Close windows when quitting an app', so may vary if that preference is changed.

Tinderbox also tracks recently opened documents on a sub-menu of the File menu. If multiple documents are open, the application's Window menu list all open windows from all documents. The latter listing does not indicate the document to which each document belongs.

Any non-default Document Settings and other customisations (link types, custom colours, user attributes, etc.) are stored in the TBX.

A Tinderbox TBX document stores its data in XML format; for some users with XML skills this can provide another way to interface with, or view, the data. Users altering TBX documents by editing the XML source should note that such work is not formally supported as an editing method, i.e. do not expect that Tech Support will be able to fix user mistakes editing source XML.

An open document can display one or more windows and tabs.

See more on the XML structure of TBX documents.

Notes & Containers

A note is an element of a Tinderbox file and might be thought of as the 'atomic' entity of information within a Tinderbox file. However, a note is really the sum of its attributes. Most obvious of these are the title ($Name) and text ($Text) of the note, but in fact the state of virtually every aspect of a note is stored in an attribute.

Every note (and agent, etc.) possesses every attribute defined in the document, much as a spreadsheet row possesses every column, or a pre-printed index card has a labelled box for each possible discrete input—regardless of whether the attribute/spreadsheet cell/index box is actually used. Indeed a note may be considered as a set of attributes, the majority of which can be viewed, and most edited.

In initial use, the note's title and text are the most obvious (attribute-based) affordances you will alter but deeper use will introduce the user to all manner of attributes both system and user types.

Notes are the basic units of writing and information in Tinderbox document. There is no fixed limit to the number of notes a Tinderbox document can handle; it can easily manage thousands, even tens of thousands of notes. At some point performance will become a limiting factor, but use of agents and rules is a bigger factor in performance than the sheer number of notes.

A note may also act as a prototype for other notes.

A note is edited via the text pane, which fills the right side of the default window layout when a note is selected in the (left-side) view pane. The text pane thus displays information about the currently selected note.

Many other aspects of Tinderbox such as agents, separators and adornments are effectively just specialised forms of the basic note object.

Notes are often referred to, or their location is described by, their Path within the canonical data outline used to store the document. That is also the outline used in Outline view.

Any note containing other notes is also regarded as a 'container'; put another way, a container is any note that has children.

Agents are a special class of container, as they create/control the aliases within themselves. Agents may not contain other notes, with the exception of adding adornments in map view

- Note Text
- Note Attributes
- Special types of notes
- Containers

Note Text

Each note (including agents and adornments) has:

- a title, i.e. the name of the note. This is stored as the $Name attribute of the note.
- body text: stored as $Text. It is not necessary for a note to have any body text.
- attributes: other pieces of information about the note. For instance the note's colour, size, position in the map, and whether it is exported or not.

The equivalent of many pages of styled text and images can be fitted in a note's $Text, but notes are best kept short. Using less text per note also makes it easy to find exactly the desired content, and helps agents find specifically what they are seeking.

A note is primarily text: it can contain many paragraphs, with colour, bold, italic and underline mark-ups, a choice of font as well. This content can also contain links to other notes or link out to web URLs. Embedded graphics in the text are also supported.

Under the hood, Tinderbox files are XML allowing a good range of options when it comes to exporting text. HTML is especially easy as an output format, but the HTML export method can be used for pretty much any mark-up format (XML, PHP, OPML, etc.) as well as plain text and including web support text formats such as CSS.

Note Attributes

An Attribute has a name, a data type, a value (which may be preset to an optional default value). Some attributes are read-only, meaning they cannot be modified by the user; this is normally because such attributes are calculated on the fly (lik the current number of child notes). If no default value is set the attribute is empty/zero/`false`/etc., according to data type. There are two primary types of attributes:

- System attributes: information built into all Tinderbox documents, such as the colour of the note, or its width and height. Tinderbox keeps this information about every note, and you may view, use, and change it.
- User attributes: you may add your own attributes that are then available to every note. For instance, you could add a user attribute "Priority" and give every note a priority level from one to five. Or in a bibliography, create the attribute "Type" and tag the note for each source as either "Primary" or "Secondary".

Additionally, system attributes may be:

- Read-only attributes: information such as the date and time the note was created, or the date and time it was last modified, or its position within the overall document outline structure. Tinderbox keeps this information about every note and you may view and use it, but you cannot change it.

In Tinderbox all attributes are global in the sense that they exist and can be given a value for any note or agent, even if it might not be useful in a particular case. Just ignore attributes when they do not make sense. Put in database terms, Tinderbox implements one big table (flat file style), rather than a set of related tables.

Note that whilst the document itself does not have an (accessible) $Text property, it does have attributes. Thus you can set some document level preferences. For instance, you can set the Map view's background colour through out the document by opening the Document Inspector's System tab and changing the $MapBackgroundColor attribute's default value.

See here for a deeper discussion of the concept of note attributes.

Special types of notes

Although essentially notes, there are a few objects in a document that by design behave differently from normal notes.

- Adornments for Maps and Timelines
- Separators
- Prototype notes
- Export Template notes

Adornments for Maps and Timelines

An adornment is seen only in Map and Timeline views and is used to provide a means of adding visual elements to the background of the Map view. In a Timeline, adornments only draw if they have both a $StartDate and $EndDate. Note th Outline view has a rough equivalent in separator notes which are shown *only* in outlines and not other view types.

As an adornment cannot be listed in an Outline, an agent cannot match or act on it.

Although adornments are not listed in most other views like Outline, these objects do have attributes. To control details like the border width, select an adornment and either use the [Cmd]+[Opt]+[I] shortcut or select Get Info from its contex menu to open the Info dialog. Most attributes are irrelevant for an adornment except those in the Map section.

Note that adornments do get counted in the outline order numbering, even if not visible there; the are counted as the last sibling children of the note on whose Map view they appear (see a deeper explanation of this mechanism). This ensur the adornment always sit behind any notes on the map. Using 'send to back' on a note in Map view will not place it behind any adornments. However, if the map has more than one adornment it is still possible to use the sent forward/back commands to re-arrange the stacking order of adornments alone.

As adornments (can—as ' smart' adornments) act as a form of container, they have support for Actions just like agents and container notes. As with a container note, the action string is stored in the $OnAdd attribute, and is triggered when a note or agent is dragged onto the adornment. A note being 'on' an adornment occurs if all or part of the note lies within the adornment's boundaries.

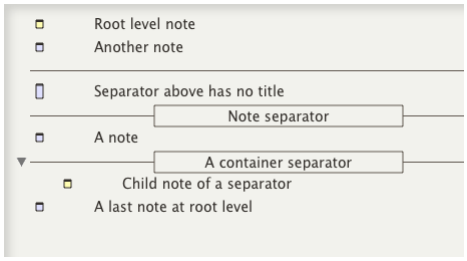
Given their role in a map, adornments show their name but not $Text. Indeed, it is not generally useful to use adornment $Text, but it can be used as a form of hidden note.

Adornments show their locked/sticky states with lock and push-pin icons (though only when the adornment is selected). The push-pin and lock icons are packaged separately in the application allowing custom icons to be added.

A prototype can hold adornments—at its sub (container) level—though they will only be visible in map view of the prototype container.

Adornments are discussed further under discussion of the Map view and Timeline view.

Separators



A note may be designated as a separator. A separator is drawn in outline view as a horizontal line. The note's name, if any, is shown centred in a box-out in the middle of the separator line. If the note has no Name value or it is "untitled' the separator shows as a solid line. For separators no duplicate name warning is given if the Name value is left blank. A note's status as a separator is held in $IsSeparator.

Just as adornments are only visible in Map view, separators are only shown as separators in Outline view (and exceptionally in Treemaps). However, in all other views except Map view, a separator is visible as if a normal note. By the same token children of a separator are hidden in Map view, as their parent container is not drawn on the map. The latter can be a useful way to 'hide' content, either notes or containers, in a map without removing the content from the overall document.

Note that separators were not originally designed to be able to be containers but rather like a (non-smart) adornment for separating groups of siblings in outline views. However, they can be containers; even agents may be a separator. Because of their possible container role, treemap views always show separators (as if they are normal notes). No note icon is shown for a separator note so the age/level of content/age cannot be judged although the separator is a fully functional note. In major views other than Outline and Map views, a separator is shown as a normal note.

A separator can still have normal note features like a Rule and OnAdd action and become a container by having notes added to it.

If data columns are enabled for the Outline view, separators still draw normally. Thus the data columns are not displayed for separator notes, unlike all other items in the outline.

Even if not needed as a separator for Outline purposes, making a Map item into a separator can be a way of hiding it (temporarily) from Map view; note that when hidden like this it cannot be selected from the Map either. To make the item visible to the map it is necessary to find and select it in a view other than map and remove the separator status.

Separators, but not adornments, are included in $SiblingOrder wheres both are included in $OutlineOrder. This makes sense when considering export is where SiblingOrder values are most regularly used and where adornments do not appear. See more on separators in a map vs. outline context.

A separator can have inbound and outbound links. It can also have note text; in such cases Treemap views will show a 'dogear' symbol denoting text content.

To designate a note as a separator, check the Separator checkbox in the Prototype sub-tab of the Properties Inspector set the value of $IsSeparator to `true`.

Separator notes default to exporting as HTML (only) if either:

- they contain both a title **and** have note text (i.e. have $Name **and** $Text values)

…OR…

- they contain child notes

If exporting Separator notes with no text (i.e. with children) ensure either the template will link through to the child content or add a ^childLinks^ into the $Text of the separator note.

Making a note into a separator does not alter any of its font-size related attributes but when drawing an outline view Tinderbox draws separator titles very slightly smaller than for normal notes. Likely this is done so as not to expand line height i.e. allowing for the box around the separator title. This effect is more noticeable as zoom level is decreased.

Duplicated separators retain their original name rather than receive the name suffix " copy" making it easier to clone unnamed separators.

Separators do not appear in hyperbolic view or in the attribute browser.

Prototype notes

Any note can be made into a prototype. This includes container notes, adornments and agents.

Export Template notes

Formatted export from Tinderbox, such as to HTML, is controlled via templates containing export code. These code templates are stored in ordinary notes that have been marked as templates ($IsTemplate).

Template notes are stored in a 'Templates' container at root of the document. This container does not exist in a new Tinderbox document, as some users may never use export features. Adding a built-in template will result in the container being added and is the easiest way to set this up.

Containers

Any notes *containing* other notes, i.e. in outline terms having child notes, is considered a 'container.' At the most trivial level this allows notes to be nested inside other notes. For instance, a 'Section' container might contain a series of 'Chapter' notes.

Containers are most useful for structured projects that will mostly be viewed in outline or chart view. In map view, only sibling notes within a a single container are visible on the map. Containers on the current map draw a ' viewport' showin part of its child map.

Agents are a special class of container, as their children consist of an an alias to every note matching the query stored in their $AgentQuery attribute.

Adornments (map view only) are not strictly containers but act like containers in that they have OnAdd and OnRemove actions and support the inside() action query.

Notes as agents

An agent is a container note that is a persistent search query. It creates/holds an alias for every note/agent/etc. object that matches the agent's query.

Like other notes, agents can have actions like rules, edicts and On Add, though for agents the latter is called the Agent action.

Agents are discussed in more detail here.

Creating notes and agents

Normal notes, special types of notes (adornments, separators, composites) and agents can be created in a variety of ways:

- Creating via keystroke
- Creating via menus
- Other creation methods

Aliases are discussed here.

Creating via keystroke

Only normal notes or agents can be created via a keystroke (shortcut):

Note:

- **Return** key (↩). In most view pane view types except report type views (Attribute Browser, Crosstabs, etc.), pressing Return will add a new note to the document as the next item in sibling/outline order.
- **Shift**+**Return** (⇧+↩). In Outline and Chart views, this will result in the new note being created as a child of the currently selected note. If the selected note already has children, it is added as the **last** child of the container.
- **Option**+**Shift**+**Return** (⇧+↩). In Outline and Chart views, this will result in the new note being created as a child of the currently selected note. If the selected note already has children, it is added as the *first* child of the container.

Agent:

- **Command**+**Shift**+**A** (⌘+⇧+**A**). This adds a new agent to the document as the next item in sibling/outline order.

Creating via menus

Various menus can be used to make a variety of note types and agents.

Note menu:

- **Create Note**. (keyboard alternative)
- **Create Agent**. (keyboard alternative)
- **Create Agent As Child**. The agent is generated as a child of the current note/container. Note: agents cannot be nested.
- **Create Separator** (Outline view only) / **Create Adornment** (Map view only)
- **Add Composite** (Map view only). Adds a new composite from those available in the current document.

View context menu (i.e. right-click), when no note is selected:

- **Create Note**. (keyboard alternative)
- **Create Agent**. (keyboard alternative)
- **Create Agent As Child**. The agent is generated as a child of the current note/container. Note: agents cannot be nested.
- **Create Separator** (Outline view only) / **Create Adornment** (Map view only)
- **Create Composite** (Map view only). Adds a new composite from those available in the current document.

Other creation methods

Notes and agents can be generated in a number of other ways.

Drag-drop or **Paste** image adornments (Map view only). See more.

Attributes (via Displayed Attributes, Inspector, Get Info or an action)

- Separators. $IsSeparator toggles a note between normal and separator. Separators appear *in outline view only* and are hidden in all other views.
- Smart Adornments. Setting a query in an adornment's $AgentQuery makes it into a smart adornment; removing the query switches back to normal adornment behaviour.
- Timeline adornments. Setting $StartDate (and $EndDate) will make an adornment show in a relevantly-scoped Timeline view as well as in a Map.

Action code:

- create() will create a note.
- createAgent() will create an agent.

AppleScript. See more on Tinderbox AppleScript methods

Deleting notes and agents

The most normal methods of deletion are:

- **Keyboard**. Use the (backspace) **Delete** key (⌫) with one or more selected items.
- **Edit** menu. Use the **Delete** option.
- **Action code**. Use the delete() operator.
- **AppleScript**. See here.

Aliases

Making an alias to a note allows a note to be placed (i.e. appear) in more than one location in a Tinderbox document, just like an alias in macOS. The original note can be in one place in the hierarchy whilst the alias can be somewhere else entirely. Both the original note and the alias give access to the content of the note: the same text window, the same attributes, and so on; see notes below re intrinsic attributes. An original and its alias(es) can share the same container/map, though normally will be in different ones.

A note can have many aliases, or none. Aliases are a flexible way for organising notes in ways that a simple Outline-style hierarchy does not permit.

An alias will always have the same name as the original. Change the name of either the alias or the original, and both will change.

- The name of an alias will appear in italics in all views (more re this further below). Due to limitations in some fonts, there is a (non-default) Document Settings option to underline alias titles instead.
- A copy of any alias does not take the suffix ' copy', but retains the same $Name as its original note.
- An alias cannot contain children/descendants, but those of the original can be queried via its aliases (e.g. in an agent). In agent queries aliases assume their original's descendants for purposes of matching.
- Selecting an alias populates the text pane with the original note (except for intrinsic attributes)of the original note.
- Intrinsic attributes are those attributes for which an alias can have a discrete value differing from the original (more & listing), for example map position and icon size.
- Editing the attributes of an alias actually edits the attributes of the original note (except for intrinsic attributes).
- Deleting an alias removes just that alias: the original and any other aliases are unaffected.
- Deleting an alias in an agent will cause the alias to be recreated as long as the original (or another alias) continues to match the agent query.
- Moving an alias out of an agent effectively duplicates the alias as the agent will recreate its now-missing child alias (see above).
- Original and alias share outbound text links (outbound internal and web links within $Text). All other inbound/outbound basic links are individual to the original or alias.
- Deleting the original note automatically deletes all its aliases; deleting an alias has no effect on the original. Any in/outbound links affected are also deleted.
- in Map views, aliases of containers are drawn in normal container style (i.e. title bar at the top) and show the viewport map:
  - if in the same map as the original: same as seen in the original (i.e. with any child objects). Although the alias has intrinsic height and width, the size/scroll state of the originals map is reflected in the alias.
  - if on a different map from the original: the viewport is drawn but is blank.
- in Map views, regardless of the map on which the alias resides (same or different from the original), drilling down into an alias' map container results in the view shifting to the map of the *original container*—i.e. *shift of context*.

Creating an alias

If you are in a view window, first select the correct note with the arrow tool. The new alias is inserted immediately after ($OutlineOrder) the source item; noting both have the same title, it is the new alias that is now selected (the styling of the title should indicate this is the alias). There are several methods:

- choose Make Alias from the Edit menu
- press Cmd-L (⌘**L**).

Drag from Find. A further method is to drag an item from a Find results pop-up/tear-off into the view pane. In this method the alias is placed at the drop point (Map) or closest insertion point in other view types.

Newly-created aliases in map view take their height and width from that of their original note, and are placed on the map to the right of their source item (assuming there is suitable free space, the alias is otherwise placed to avoid overlapping/compositing with another note).

Once created, the alias can be moved anywhere in the current document.

Agents. Another method of making aliases is to create an agent as that will create aliases for any notes matching the agent query. Aliases can then be dragged elsewhere, in which case the agent will spawn a new child alias as the source not still matches the query.

Exporting

When exporting, aliases also behave as if their original note's children were their own children. Aliases are exported as separate pages in the appropriate location within the output. This makes it easier to use web links to alias content that point to the right place. It also helps when web output uses a hierarchical navigation system as with aTbRef.

When including children that are aliases such as when using ^children^ and ^descendants^, items that happen to be aliases are included as aliases (so reflecting intrinsic attribute values). Since for most purposes the alias and its original are interchangeable, this seldom affects export. However, intrinsic properties of the alias are exportable (and may differ from those of the original).

Reflecting the way aliases are treated as separate entities in an export context, basic links to or from aliases belong to those aliases and they support their own Roadmap view and Browse Links windows. However, the links() action code cannot currently address aliases as a destination object so analysis of links to/from an alias is best done visually or by means of Roadmap view. Originals and aliases export their own basic links (i.e. aliases can differ), but if an alias has no in/outbound basic links it will export those of the original. In the latter case the alias will show no basic links internally, e.g. in Roadmap view, but on export will inherit its original's basic links.

Text links as well as basic links are generated from exported aliases; the destination of the link is the same regardless of whether the source is the original or an alias. An alias can never have $Text/text links that differ from that of its original: that is fundamental of the point of it being an alias.

Copying aliases

Any alias can also be copied or aliased. However, do note that when copying/aliasing an existing alias the new alias is created as if being made from the alias's original though it uses the original's attributes rather than the source alias' intrir attributes. Likewise, the source alias' discrete basic links are not copied, i.e. the new alias will start with no basic links. In short you cannot make a copy of an alias that maintains the intrinsic differences of the source alias from the original. Aliases cannot be copied outside their current document, as another TBX will not hold the original of the (alias) note being copied. An alias needs to point to its original for many of its attribute values so the two must reside in the same document.

Identifying aliases in queries

The $IsAlias attribute can help with identifying whether a note is an alias or note. This is especially helpful within agents where all matches are aliases, regardless of the type of note being matched.

The $Container attribute is worthy of note as this means the ' parent' designator value of an alias does not equal that of the original note. This becomes important in agent actions where the acted-on note is an alias. It is thus often necessary t use syntax like `$Name(parent(original))`, i.e. the name of the original note's parent container, rather than `$Name(parent)`. In an agent action the latter code returns the name of the agent. Incidentally, in the last scenario `$Name(parent)` and `$Name(agent)` would both return the name of the Features needing more recent OS versions (more on the 'Link type honouring operators', 'original' and ' parent' designators).

Aliases render in italics

In any major view aliases are always drawn with their title ($DisplayName) in italics. There is no way to make an alias be titled in normal type; this is by design, mirroring how the Mac's OSs have shown aliases. One result of this is that when older, pre-v6, TBXs are opened in v6+ they may not render aliases in italics: see more. There is a document setting to also underline alias titles, although they still render in italics (if possible).

Link counts and aliases

$InboundLinkCount and $OutboundLinkCount are intrinsic and report separate values for the alias and original note, those they may often be the same (outbound text links are always the same for original and alias).

Word counts

Word counts in Get Info/info pane do not include the $Text word counts of alias notes, just the count of the original $Text.

Composites

A *composite* is a group of Tinderbox notes that work together to describe something larger than themselves. For example, when taking notes (in Map view) in a conference, an individual talk might be a composite of notes for the title, the speaker, the content of the talk, and action items requiring follow-up action. Each note has its own text and attributes, but it may sometimes be useful to treat the composite as an object – for example, to move all the notes in the composite tc a new map location. Further examples of usage, as envisaged by Eastgate, include the following (giving in parentheses the items the composite might contain):

- shopping lists (a store, followed by one or more things you want to buy)
- books (author, title, reading notes, editorial notes on the review we will write)
- lesson plans (date, topic, assignment, homework, special requirements)
- restaurant visits for a reviewer (name, phone, hours, credit card info, who dined with you, what you ate, when the review is due)

Composites are a feature primarily intended for map view use and Tinderbox notes form a composite when their map icon touches another note. Composites make it possible to make compound notes which have individual identities but also are linked to their collaborators in a new (visual) way.

Some built-in composites can be created from the File menu. This creates a root level 'Composites' container. User-created blank composites can be added to the container. Any composite in this container can be used to create a new copy by choosing Create Composite from the Note menu or the map background's contextual menu.

When selected, composites are outlined with a darker and thicker bounding box. The composite name and edit widget (a little pen icon at top left of the bounding box) are also displayed when the composite is selected. To (re-)name the composite, click the pen symbol: a pop-up naming box appears over the windows text pane. Enter the desired name for the composite and press the Return key to set and save the new name.

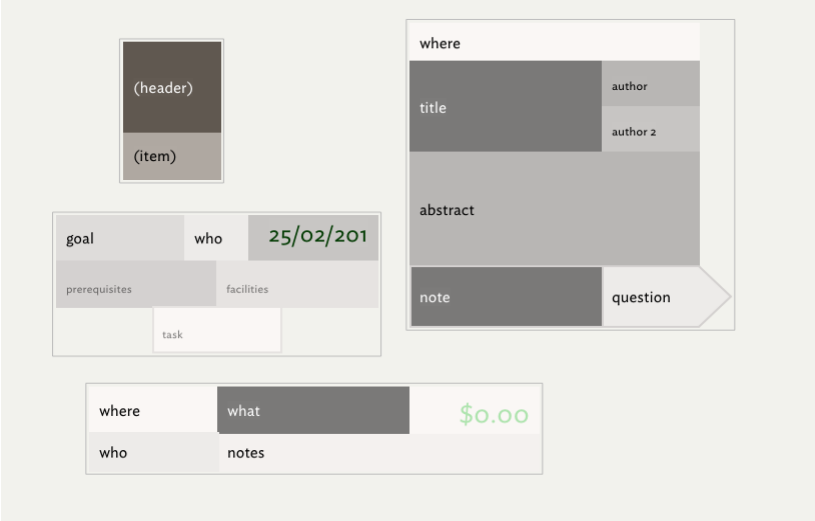
Clicking on any note within a composite selects the composite; clicking on non-note space within the composite's bounding box does not select the composite. Cmd+click to select individual note(s) within a composite. To edit an individual note, or to see only its text and Displayed Attributes, the note must be individually selected. When a composite is selected the resize handles are not drawn on the individual constituent notes.

Edit ▸ Break Composite can be used to break up a composite into individual components. Individual notes can be removed from a composite by (Cmd+click) selecting it and dragging it away from the other notes. When using ⌘-drag to remove a note from a composite, the note is removed from the composite immediately. (Previously, it was removed from the composite at mouse-up. The new approach makes the effect of ⌘-drag more clear.)

When a composite is selected, the text pane displays the text from each member of the composite, as for multiple selections. Composites may have a name. When any item in a composite is selected, the name of the composite is shown above the composite, along with a control widget that allows you to rename the composite.

When a note joins a composite, its moves in the outline to become the younger sibling of the last member of the composite. Thus, all members of a composite are adjacent in outline order. The relative order of items within a composite should remain unchanged.

Further composite features:
See here.



Automating Tinderbox

To get an idea how Tinderbox works, it is useful to have some idea as to the main sorts of objects found in a Tinderbox data file, and the concepts behind their use:

- Concepts
- Coding
- Coding conventions
- Deprecated Usage

Concepts

This section contains:

- Windows, Tabs & Views
- Hierarchy of Content
- Paths
- Uniqueness, duplicates and matching notes
- Attributes
- Prototypes
- Inheritance of attribute values
- Data export
- Data import
- Unicode
- Inline images
- Fonts: italics and bold

Windows, Tabs & Views

A Tinderbox (TBX) document has a single two-paned document window by default, with two tabs. Your work in Tinderbox takes place in one of the two main panes of a document window, the View pane and the Text pane:

- The View pane (left-hand pane) depicts some, or all, of the document's notes using one of a variety of view (visualisations) types. The View pane represents what were stand-alone 'main view' pre-v6. There are a number of types of view used by the view pane:
- Map.
- Outline.
- Chart.
- Timeline.
- Treemap.
- Attribute Browser.
- Hyperbolic.

The Text pane (right-hand) represents the pre-v6 text window for a note. It holds the note title, optional Displayed Attributes, and the note's text ($Text). Alternate sub-tabs of this pane show the raw mark-up of the note as exported via HTM (marked-up) Export and the same in web browser preview form.

Some View pane views show only a narrow context, e.g. Map view, whilst others (by default) show the document's hierarchical structure such as Outline, Chart or Treemap views.

All new document windows always have a Map view (tab #1, selected by default) and Outline (tab #2). However, users are free to use as many or as few view types as their work or particular projects demand. Not all users will need to use every view type.

A small number of secondary view and windows from earlier Tinderbox designs have become pop-over dialogs (and thus essentially modal), although a few such pop-overs 'torn off' text windows can be created, by dragging them to becom stand-alone windows. Pop-over offering this affordance do so with a double rectangle icon in their top right corner. Stand alone text windows show the contents of a note's Displayed Attributes and $Text, but lack some text pane affordances such as the link well, title bar and Preview/HTML tabs.

Multiple document windows may be open while working in Tinderbox as well as torn-off windows and the Inspector. They can show the same or different parts of your document. It is also possible have multiple windows open for more than one TBX document. Thus, if more than one document is open at a time, the windows from each/every document are be open at the same time. This can be useful when trying out new code techniques, to avoid doing potentially disruptive test on a primary work document.

Hierarchy of Content

A Tinderbox document has two basic kinds of structure: one is the structure formed by links between notes. The other is the hierarchical structure of the document. A Tinderbox note can contain other notes, which can contain other notes, and so on. The notes can thought of this as chapters containing sections containing divisions; or topics containing sub-topics; or in any way that fits the user's workflow.

In the image for this article, the 'MyString' column shows the relevant designator (in correct case-sensitive form). Note that designators are case-sensitive when used in export/action code; by convention designators are normally not quoted in action code, although they are strings. The OutlineDepth column shows the $OutlineDepth for each note (1 being root level).

IMPORTANT: inheritance of attribute values (e.g. via prototypes) is **not** a feature of hierarchy. Put another way, a child note does **not** automatically inherit all its parent's customisation: that *may* be made to happen but will require the user to add some action code.

Several terms are used to describe relationships within a hierarchy:

- parent: the item one level up. A parent note contains one or more child notes (children), and thus contains one or more levels of sub-notes. A parent note may also be referred to as a 'container' as it contains other notes. ('parent' designator)
- sibling: an item that has the same parent. ('firstSibling', 'prevSibling', 'nextSibling', 'lastSibling' designators). The group designator siblings means all items with the same parent as the current item.
- child: an item one level down. In Tinderbox, when a note contains other notes, those notes are its children. A child note is thus any note that has a parent note, i.e. it is not a top level note. ('child', 'lastChild', 'randomChild', 'previous' (child), 'next' (child), Nth child (zero-based) 'child[N]' designators).
- descendant: a child, or a child of a child, and so on, of the current note.
- ancestor: a parent, or a parent of a parent, and so on, of the current note.
- grandparent: specifically the parent of a note's parent. ('grandparent' designator)
- root: an item that has no parent. In a Tinderbox document, the document as a whole is the root: it contains the top-level notes, and nothing contains it. Root level notes have $OutlineDepth of 1. Root is not used as a designator but does have a role in export code: see ^root^.
- cover: the first root level sibling. Thus, if there is more than one root-level note, the cover is the oldest sibling. ('cover' designator). This is little used in Tinderbox and has more purpose in its sibling program Storyspace.
- original. The note upon which an alias is based. ('original' designator)
- this and current. These generally mean the same thing. 'this' applies to the currently selected item; where transclusion is used (e.g. during export, see ^include^) current applies to the note in current scope of evaluation.
- agent. This is a method for agent queries to refer to the agent's attribute values and for aliases to refer to their parent agent in agent actions. ('agent' designator)
- adornment. Not illustrated, as only used in map view. It fulfils the same role as 'agent' in adornments and smart adornments ('adornment' designator)
- ID. Not a hierarchical element, but an item's $ID value (a 10-digit number) can be used as a designator.

	MyString	OutlineDepth
Cover	cover	1
Note (when selected)	this	2
$ID value 1627576511	1627576511	2
Source of aliased note	original	2
Root	root	1
Descendant	descendant	2
Descendant	descendant	3
Ancestor	ancestor	2
Grandparent	grandparent	3
Parent	parent	4
Child	child	5
Descendant	descendant	6
Root-level container		1
Sibling	firstSibling	2
Sibling	prevSibling	2
Sibling		2
Sibling	nextSibling	2
Sibling	lastSibling	2
Root-level container		1
Child	child	2
Child[N]	child[1]	2
Child	previous	2
Current Child	current	2
Child	next	2
Child[N]	child[-2]	2
Last Child	lastChild	2
agent	agent	1
Source of aliased note		2

Not illustrated: item-scope designators: ' that' and ' find()'.

There are several group terms to describe Tinderbox notes:

- all: all notes in the current TBX file. ('all' designator)
- ancestors: all ancestors. ('ancestors' designator)
- children: all children. ('children' designator)
- descendants: all descendants, i.e. all children and their children. ('descendants' designator)
- sibling: all siblings. ('siblings' designator)

Any note containing other notes is a 'container'; put another way, a container is any note that has children/descendants. Agents are a special class of container, holding an alias to every note matching the query stored in their $AgentQuery attribute. Map adornments are not containers. Though not originally intended as such, outline separators can be used as containers.

Paths

Tinderbox allows you to identify notes by relative or absolute paths, not just name. If a tag's request matches neither a keyword nor a note, Tinderbox tries to interpret it as a path. For example:

`/news` ...is the top-level note named "news"

`/news/Iraq` ...is the note named "Iraq", inside the top-level note "news"

`../Chicago` ...is the note named "Chicago" that is a sibling of this note

`../..` ...is the grandparent of this note.

Using paths implies:

- knowledge of the Tinderbox document's outline structure
- where in the outline that named notes are used
- that the outline will not change.

If a note name contains a forward slash, as in 'Large/Medium', then the path must be enclosed with quotes, e.g. `Large/Medium` should be cited as `"Large/Medium"`. A deprecated legacy alternate is to escape the slash by a preceding backslash, e.g. `Large\/Medium`. Doing otherwise, the backslash is treated as part of a path, i.e. is a container delimiter.

A path beginning with a / is always a (complete) absolute path down from a top-level note (though it need not be the cover if there is more than one top level note). Relative paths start with `../` (i.e. period+period+slash). A path not starting \ either of these will result in Tinderbox treating the proffered string as a single note name.

Quoting of Paths

Historically paths were not quoted where used in Tinderbox code, and thus may be omitted in many older code examples you find. However, users should switch to quoting paths as they are string literals. Past convention in code examples is to quote paths in action code but not in export code, but this largely reflects past usage and should not prompt current users to omit quotes.

An exception to quoting paths occurs, where a path argument is actually an expression or an attribute name. A simpler way to view this is that quote enclosing a string of text is to say to Tinderbox, "This is just text. Do not try interpreting any control/code characters found within it", i.e. string literal.

Using Path arguments in Tinderbox

Many operators and codes in Tinderbox that allow a note name as an argument also allow a 'path' to be used. All attributes can also take a path argument, allowing code to refer to an attribute from some other note.

For new users, figuring out paths can be a bit confusing. But firstly, why might a path argument needed? If a note name is not unique across the whole TBX document, then when Tinderbox finds a code mentioning that name, it will use the first matching note as found by $OutlineOrder. So, using a path rather than a note name can help identify exactly to which note you wish to refer. Consider using paths explicitly if:

- Your document deliberately does not use unique note names
- It may be that names may not remain unique over the life of the document.

That's the 'why', but how do you refer to a note using a path? To investigate this consider the following TBX outline:

```
First Root
  Child A
    Sibling A1
    Sibling A2
  Child Z
Second Root
  Child A
    Sibling A1
  Child B
    Sibling B1
    Sibling B2
  Child C/D
    Child of D
```

In this demo, you will note that there are some duplicate note names. For the examples below, to help give starters code such as they might use for real, the examples are written as if referring to a value of the system attribute 'Created'.

Note name (i.e. no path)

Just use the note name on its own:

`$Created(Child B)`

`$Created(Child A)`

`$Created("Child C/D")`

`$Created("/Second Root")` N.B. need for root slash

Notes:

- note name will not suffice if you want to reach the 'Child A' in 'Second Root'. The note of the same name within 'First Root' will always get called.
- any paths (or just note names) with forward slashes must be 'escaped' by quoting the whole string.
- citing root-level note names must include a slash prefix, i.e. when the $Name is the $Path—or rather the $Path is "/"+$Name . See also notes on partial paths further below.
- there is still legacy support for the *now deprecated* old usage of preceding (/) slashes in paths with a backslash (\) character.

Ways to view a note's path

The clearest way is to view $Path data but various views offer ways to see all or part of the path:

- hoisted views show a 'breadcrumb' trail indicating ancestor container, and thus the path to notes in the current view.
- Outline, Chart and (albeit less obviously) Treemap views allow the (ancestor) path to be viewed. In the case of Outline and Chart it may involve expanding the outline branch holding the current note. In the case of the treemap it is a cas of spotting the nested containers.
- Text pane. Holding the cursor over the note name results in a tooltip showing the full path of the note. This is useful when a $Name is not unique within the document.

Paths from Attribute Values

A path may be supplied as an attribute to address an attribute of another note than the current one.

`$Text($MyPath)`

…where $MyPath is a user attribute holding a path.

Paths from Expressions

Path may also be supplied in the form of an expression such a string concatenation or simple arithmetic:

`$Text(' "../"+(1+2) ')`

…would collect the $Text of a note named "3".

Paths with Designators

Most Tinderbox designators can use a path to refer to a different context. For example:

`min(child("/some/other map note",$Xpos)`

`^min(child("/some/other map note",Xpos)^`

Paths with nested designators: it is possible to cite an offset combining designators.

```
ancestor
  Examples
    A
      1
      2
    Aliases
      A
another
  More
    B
      3
```

For the alias A

$Container(this) is "Aliases"

$Container(original) is "Examples"

The designator parent(original) will give the parent of the original -- the parent of Examples, So $Container(parent(original)) is "ancestor".

The designator original(parent) will give the original of the parent -- "Aliases". So $Container(original(parent)) is "another".

Thus nested designators like this tend to be written as spoken: the 'parent of the original' being parent(original).

Other ways to identify a note other than via its path

The $Path attribute holds the note's path. For unique note titles the title alone ($Name) can be used as a proxy for the path in many cases. A robust but less human-readable path alternate (but accurate) is to use $ID or the newer v9.5.0 $IDString. The latter can be useful if the $Path data contains problematic characters like a forward slash in any note title in the path.

Further notes on Paths

See here:

- Absolute Paths
- Relative Paths
- Partial Paths

Absolute Paths

With an absolute (or 'full') path you are stating the full and exact path within the Tinderbox document. These are absolute paths:

`$Created(/First Root/Child A/Sibling A1)`

`$Created("/Second Root/Child C/D/Child of D")`

Points to note:

- there can be more than one root (top level note) but regardless the path always begins with a slash (/) and the name of a root note.
- every note from the root to the destination note is included in the path, divided with a forward slash.
- the path does not end in a slash character.
- the path may contain spaces but does not needed to be enclosed in single quotes, although for consistency it may be a good idea to quote. Since v4.6, syntax has been moving towards explicit quoting of all strings.
- any path containing forward slashes must be enclosed in quotes.

Absolute paths are not a big deal in a small document like the example, but if it is a big TBX with deep nesting, you will want to consider using relative paths.

Relative Paths

The basic rules are roughly the same as for absolute paths:

- the relative path always begins with a dot-dot-slash (../)
- navigating upwards, a dot-dot-slash is used for every level of outline traversed.
- navigating downward use the name of the notes working down from that point divided with a forward slash between each nested container's name.
- the path does not end in a slash character.
- the path may contain spaces but does not needed to be enclosed in single quotes, although for consistency it may be a good idea to quote. Since v4.6, syntax has been moving towards explicit quoting of all strings.
- any path containing forward slashes must be enclosed in quotes.

For the following example assume the currently selected note is 'Child B' within 'Second Root'.

```
$Created(../Child A)
$Created(../../First Root/Child A)
$Created("../Child C/D/Child of D")
```

A relative call from Sibling B2 to its grandparent (Second Root) would be:

```
$Created(../../Second Root)
```

The above examples show two subtle points to note about relative paths:

- to access a sibling (or its children) you must go 'up and back' to the current note's parent. You cannot call a sibling by using its name with no slash prefix.
- when accessing notes under a different *root*, you must navigate to a level above the root note (effectively the document itself) before you navigate downwards.

If you are used to UNIX file navigation, note the absence of the dot '.' method. Also, if you are used to HTML relative links, note the 'up and back' method of navigation: you cannot cite a sub-folder or sibling simply by using its name (without a preceding slash). So some slight variance form what some more expert users may intuit but the usage is consistent within Tinderbox and not difficult to learn.

Partial Paths

An easy mistake to make, especially when trying to disambiguate between two same-named notes, is to use a 'partial' path, such as in citing `container name/note name`. Although not illegal usage, this does not necessarily get parsed a the user assumes. The logic runs like this:

- If the path argument is an attribute value, that value is evaluated. If the path is quoted, the quoted expression is evaluated and used. Else…
- If the path argument is a designator (e.g. 'parent'), use the designator. Else…
- If the path argument starts with a '/', it is evaluated relative to the root of the document. If not, it is evaluated relative to the current note (though such relative paths are not generally recommended). If the absolute or relative path locates a note that exists, use that note. Else…
- If the path argument designates a note that is an immediate child of this note, use it. Else…
- If the path argument designates any note in the document whose title ($Name) exactly matches the note, use it.

Bottom line: do not use use partial paths as shorthand. Use a full path or a relative paths as described above.

Uniqueness, duplicates and matching notes

Does a note (agent, etc.) need to have a unique title ($Name)?

No, though usually unique, an object's title ($Name) is not required to be unique. Same-titled objects will have different a $Path path, unless they are in the same container. Even then, all objects (including aliases) have a discrete $ID enabling duplicate objects to be addressed discretely.

Be aware that generally, that when using code with attributes using an offset address argument, most users only employ an object's $Name and this is apparent in most code examples found in the forums and in this resource.

Is is possible to test for duplicated titles?

Yes, by using isDuplicateName(). For instance a stamp with code:

```
$MyBoolean = isDuplicateName($Name);
```

… when applied to one or more objects will set $MyBoolean to `true` for any object whose $Name is not unique. The scope of the test is always the whole document.

Are there any $Name values to avoid duplicating?

Yes. A number of build-in features—when used—generate root-level containers: prototypes, (export) templates, composites, and hints. It is a good idea to avoid duplicating these features' default containers, indeed all sub-containers within '/Hints'. If it is necessary to duplicate any of these containers (and at root level) ensure the built-in feature generated one is *first* by outline order ($OutlineOrder).

How do duplicate titles affect queries and finds?

Duplicate-titled notes have no effect on queries which will generate an alias (agent) or path (find()) for each discrete note, even if the duplicate-titled notes are in the same container. It is not documented but it may be because the duplicates ar resolved using $ID or $IDString (from v9.5.0) which will be unique for each note. Note that whilst a find() uses a query, like an agent, the difference is that a find() does not weed out matches aliases whereas an agent does so..

How do duplicate titles affect queries and finds?

When referring to another note via its $Name as an attribute argument, e.g. `$MyString("Some note")`, if there is more than one match the resulting action(s) will be carried out only on the first match by outline order ($OutlineOrder). Thus the $Name matches 3 discrete notes, the action will occur three times on the first matched not and not once each on each matched note. This issue can be countered in part by using $Path rather than $Name for the argument. If the matches are duplicates within a single container, only $ID can be used to ensure every match note is acted on by the action.

Summary

It can be seen that actions—including agent actions—rather than queries that are affected badly by notes with a common name. If document's subject requires the existence of duplicate note titles, extra care should be taken in how actions are addressed to the corrected note(s).

Attributes

Many aspects of Tinderbox are set and stored by using attributes, such as a note's name, the colour of its icon in a Map view, etc. Attributes apply to every note (including special notes like separators and adornments) and agents.

An Attribute has a case-sensitive name, a value type, a value and an optional default value. Some attributes are read-only, meaning they cannot be modified by the user. If no default value is set the attribute is empty/zero/ `false`/etc., according to data type. There are two primary types of attributes:

- **System attributes**: information built into all Tinderbox documents, such as the colour of the note, or its width and height. Tinderbox keeps this information about every note, and you may view, use, and change it.
- **User attributes**: you may add your own attributes that every note will have. For instance, you could add the attribute "Priority" and give every note a priority level from one to five. Or, in a bibliography create the attribute "Type" and tag the note for each source as either "Primary" or "Secondary". More on naming attributes.

Additionally, system attributes may be:

- Read-only attributes: information such as the date and time the note was created, or the date and time it was last modified. Tinderbox keeps this information about every note, and you may view and use it, but you cannot change it.

The two primary attribute listings, General and User, and their data types, etc., are covered in more detail in the Attributes section and all General attributes are described system attribute list. Values used by attributes are often inherited. There are also naming rules for attributes.

System attributes will generally have a default value which may be set via Preferences or, at document level, by editing the default value.

In Tinderbox all attributes are global in the sense that they exist and can be given a value for any note or agent, even if it might not be useful in a particular case. Just ignore attributes when they do not make sense. Put in database terms, Tinderbox implements one big table (flat file style), rather than a set of related tables.

The attributes of TBX document are set at document creation by Tinderbox and may be altered via the Document Inspector's System tab, though some can also be set via Document Settings. Thus you can set the Map view's background colour by opening the document's Info view and changing the $MapBackgroundColor attribute; be aware changing this attribute only affects the document if the current 'item' is the document and not a note, i.e. no note is currently selected.

In Action code, attribute values are referred to in Tinderbox attribute coding by using a '$' prefix, e.g. `$Color` returns the _value_ for the attribute 'Color' (i.e. the note's colour in the view pane). By general informal usage, the Tinderbox community will also use (misuse!) the $-prefix in general writing to make explicit they are talking about a the name of an attribute. Thus "See the $Name of…" or "You can use the $Height…" where in both cases the context implies that "$X" means the "the attribute named X". This latter convention is used in aTbRef.

There are a variety of methods for editing attribute values.

Give consideration to naming attributes in a manner that matches Tinderbox's assumptions. If attribute names are wrong, either due to import or because they were created before reading the notes on naming styles, attributes can be renamed.

Attributes all have one of a number of data types.

Attribute use in the context of automation ('coding') in Tinderbox is discussed in detail under Use of Attributes.

Prototypes

Prototypes are a method to let a single note specify the default value for an entire class of notes. This is achieved by specifying an existing note as the prototype, via the $IsPrototype attribute; this tells Tinderbox to use the prototype's value for the all the notes set to use that prototype. The easiest way to make a note a prototype is via the Properties Inspector's Prototype tab.

Attribute inheritance

See also: Inheritance of attribute values.

Whenever Tinderbox uses a prototype it will re-use an attribute value from the prototype for its own attributes unless the attribute is specifically set for that note (by action or manual edit). Such manual edits break inheritance (for that attribute

only) but inheritance can be reset. There are some special mechanisms for disabling action inheritance.

When you change an attribute in a prototype, you change it for the notes that use that prototype. A note inherits all attributes from its prototype except for intrinsic attributes. Prototypes are not a necessary feature for everyday use but they can powerful time-savers for complex projects. Indeed, any large or long term Tinderbox project is likely to benefit for judicious use of prototypes.

As well as normal notes, an agent or adornment may be a prototype, but setting a normal note to use either of those will note change the nature of the note; it will simply use the customisations it can so, for example an agent query would be ignored. Nonetheless if making large use of agents or adornments, prototypes can speed their configuration.With agents, ideally the prototype should not have a query set though Tinderbox will cope with such a configuration. The Rule and Display Expression disabling attributes can help by ensuring things like self-resetting rules only run in non-prototype notes.

If a note or agent is copied and pasted to a *different* document, the new item does not inherit its source's $Prototype value. This is a defensive assumption by Tinderbox as it cannot be sure pasted notes are from the current TBX and thus th all source prototypes exist. When creating 'template' type notes/containers bear this in mind. Think about using an agent to help set prototype values directly or indirectly via rules and actions. The |= operator can be used to ensure actions occur once only.

Inheritance of Text

When a note's prototype is set or changed, the note copies the prototype's text ($Text) if the note has none, or if the text it inherited from its former prototype has not been changed.

Inheritance of Text links

From v9.5.2, if the prototype's $Text contains any text links, these are inherited when $Text is inherited (as described above)

Prototypes bequeathing child notes

Prototypes can have children that are created/added to a note if that prototype is applied, described in Tinderbox as 'bequeathing' notes— see more.

Built-in Prototypes

Tinderbox makes it easy to experiment with prototypes as a number of simple generic ones are predefined and built-in, noting that a few of these—such as for notes in DEVONthink watch folders—are added as-needed by the app and are nc available for manual insertion via the File menu.

See more:

- Setting prototypes
- Disabling action inheritance
- Prototypes 'Bequeathing' Child Notes
- Duplication and prototype assignment
- Shared prototypes
- Chaining prototypes

Setting prototypes

There are many ways to set prototypes, many using the prototype pop-up menu:

- Outline view. Right click icon of selected item(s), select from pop-up menu
- Map view. Right click prototype tab of selected item(s), select from pop-up menu.
- Text window. Edit the $Prototype attribute displayed as a displayed attribute.
- Properties Inspector, Prototype sub-tab. A tick-box is available to set prototype status.
- Properties Inspector, Quickstamp sub-tab. Select one or more items. Either:
  - Type all or part of the attribute name. If part, pick a completion from the pop-up auto-complete list. Press Return (↩) top populate the relevant Group and Attribute selections. Click the 'Apply' button.
  - First select the 'General' attribute group (second pop-up list). Now from the third pop-up select 'Prototype'. Click the 'Apply' button
- Get Info dialog, attributes tab. Edit the $Prototype attribute.
- Action code, via Stamps, or OnAdd/AgentAction, Rule, Edict, etc. Set the prototype directly in code, e.g. `$Prototype="Person"`.

Disabling action inheritance

Three special boolean system attributes control inheritance of rules, edicts and display expressions: $RuleDisabled, $EdictDisabled and $DisplayExpressionDisabled. Importantly, these attributes are all *intrinsic*, i.e. their values are *not* inherited.

This mechanism allows a prototype to store a rules, edict or display expression, but have the option to not execute the code in the prototype. Consider a rule that alters one of the note's attributes. If run in the prototype, this modified would be unintentionally inherited by other notes. By disabling the prototype's rule via $RuleDisabled, the rule does *not* run in the prototype but *does* run in notes using the prototype. Of course, any note using the prototype is still free to disable the rule via its own $RuleDisabled as it doesn't inherit the prototype's value.

The document default value of $RuleDisabled is `false`. If it is set to `true` in the prototype, then the value inherited by notes using the prototype is `false`. Why not a `true` *value*? This occurs because the attribute is intrinsic and its state is never inherited so the (inherited) default value is always the document level default.

This leads to a reverse problem, what if it is desirable for the prototype to tell inheriting notes to observe the disablement state in their prototype? See chaining prototypes.

Prototypes 'Bequeathing' Child Notes

If you want a note that has children to be a prototype, but without its children being inherited, there is an attribute ($PrototypeBequeathsChildren) to control this aspect of inheritance. Inheriting notes gain (a new copy of) all the descendants o the prototype, not just immediate children. The outline arrangement of descendants is preserved, i.e. the note using the prototype becomes a container with the same descendant layout as the prototype. The maximum limit on bequeathed items, limited to avoid accidental infinite recursion, is 500 children (originally 100).

A note with existing children will not be bequeathed children, even if an applied prototype is set to do so. This is because the bequeathal notion was for implementing data structures rather than as some form of stamp to add additional children.

Note: prototype children are not copied recursively. When setting the prototype for a note that already exists, Tinderbox will add new children if the prototype has children, but Tinderbox will not delete any child notes that already exists. If the cloned note would have the same name as an existing child, the existing child will be unchanged and the prototype child will not be cloned.

Therefore, do ensure that none of the bequeathed descendants are actual prototypes, i.e. all prototypes used in the bequeathal should be direct children of the /Prototypes container. If any of them need to be prototype-based, create separate prototypes and store these outside the bequeathing prototype container. Thus if prototype A has child AB and grandchild ABC, then if either both AB and ABC are prototyped, their bequeathed copies are non-prototype copies of AB or ABC. Instead, if AB and ABC are based on prototypes, then the bequeathed notes inherit their source note's prototype (i.e. achieving the original desired aim).

Agent prototypes can bequeath any adornments saved within them, including image adornments.

If nesting prototypes with no intent to bequeath children, $PrototypeBequeathesChildren should be set to `false`. Note this is the attribute's *non-default* value.

It is possible to designate some prototypes as 'private' ($Private), meaning it does not appear in the selector lists available in Map & Outline view. Private prototypes can only be created via setting $Private and not via the UI.

Duplication and prototype assignment

Duplicated notes retain the source note's $Prototype assignment. Consider prototype 'Note A' used by 'Note B', then

- duplicating 'Note A' makes a new note 'Note A copy'. The duplicate 'Note A copy' is itself a new prototype note but has no prototype link to B; B retains its 'A' $Prototype assignment. That is congruent with links & duplication, noting that prototype assignment is effected as a (hidden) link. The duplicate does have copies of other outbound links for A, but prototype links are no longer duplicated.
- duplicating 'Note B' makes a new note 'Note B copy'. The duplicate 'Note B copy' is a new normal note as retains 'note A' as its prototype.
- duplicating both notes togetherhe same outcome occurs. Although 'Note A copy' is a prototype, both 'Note B' and 'Note B copy' retain 'Note A' as their prototype.

Shared prototypes

Tinderbox allows users to store prototypes that can be added to any Tinderbox document open on the same Mac. These stored prototypes essentially allow adding 'custom' built-in templateshis can assist with configuring new documents more quickly.

Locally shared prototypes are stored as top-level notes in a Tinderbox document named Prototypes.tbx in the Tinderbox support folder's sub-folder " prototypes", at:

```
~/Library/Application Support/Tinderbox/prototypes/Prototypes.tbx
```

These shared prototypes will appear at the bottom of the Built-In Prototypes sub-menu. Choosing one of these prototypes will add it to the current document's 'Prototypes' container.

If the current document's 'Prototypes' container already holds a prototype with the chosen name, the command has no effect and no new prototype is created.

The newly added prototype shares most attributes with the source prototype in Prototypes.tbx, with the following exceptions:

- User attributes that do not exist in your document
- Intrinsic attributes such as $Xpos and $Container, that are never inherited and can be different for a note and each of its alias(es).
- Read-only attributes.

Chaining prototypes

Prototypes may themselves use a prototype, i.e. assign a $Prototype value. This can be useful where several different prototypes share a common base. In such circumstances, it can be useful to use one precursor template and then have several other prototypes using it as their prototype. Changing the upstream prototype will affect all prototypes using it and all notes/agents using the downstream prototype.

An example is swapping in/out a complex Displayed Attributes table, rather than constantly showing/hiding the table. Here, the main prototype has no $DisplayedAttributes value set as does one of the two upstream prototypes. The other upstream prototypes has the desired full Displayed Attributes table set. Toggling the main prototype between using the other two prototypes magically sets/removes the Displayed Attributes table.

Beware the above scales awkwardly if you want to toggle more than one feature separately from others. We might want to alternate between two differing Displayed Attributes tables or none. For that, *three* upstream prototypes (none, table table B). But, it we want to separately control Displayed Attributes and Display Expression, for example, we need four upstream prototype: A & B, A only, B only, neither. For 3 things, it is 3 (ABC on, AB only, AC only, BC only, A only, B only, C only, none). Four discrete aspects? Sixteen upstream prototypes.

So, this is best used to control one or two discrete features. However, if all affected features change together, then only two upstream prototypes could toggle rule/edit/Displayed Attributes/Display Expression/shape/colour/export templates/etc. via one prototype selection switch.

Another instance is where it is desirable for the prototype to tell inheriting notes to observe the disablement state of their prototype's rule, edict or code: recall that being controlled by intrinsic attributes that state is not inherited). For instanc if hundreds of notes re-use a complex rule inherited from their prototype and constant code execution slows the document down. One solution is to use an edict as that runs less frequently. But how to make an on/off switch?

An elegant solution is for the prototype itself to use a pair of upstream prototypes. Thus, one upstream prototype has no rule set, the other one does. By resetting the main prototype's rule to the default, it now inherits its rule for (no rule code or for a rule) from the upstream prototype selected.

Inheritance of attribute values

The series of notes in this section of aTbRef describe how inheritance of attribute values works in Tinderbox as this can be be confusing for those beginning to use Tinderbox.

Gaining a working understanding of the process is a solid step towards deeper use of the program.

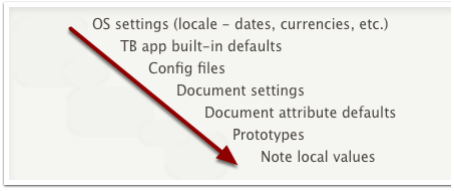
- Attribute value inheritance flows downwards
- Inheritance for Intrinsic or Read-only and attributes
- Inheritance is not based on the actual outline of your data
- The Inheritance Cascade
- Inheritance and prototypes

Attribute value inheritance flows downwards

The relationship illustrated here shows the possible sources of the default for an attribute value as found in a Tinderbox note. As attributes also control many aspects of how data is displayed within Tinderbox, knowing how these values are inherited and (re-)set will help you understand the look and feel of a Tinderbox document as well as any data the user adds.

The inheritance principle applies to attributes of all data types. There is a slight difference of scope in the case of user attributes. As these are created within a document, their inheritance tree starts part way down the above sequence, at "Document attribute defaults".

A few special attributes do not inherit values in this manner…



Inheritance for Intrinsic or Read-only and attributes

An exception to the inheritance principle is that intrinsic attributes do not inherit. This is deliberate design intent. This is to allow aliases and original notes to hold a different value of the same attribute in pertinent circumstances.

For instance, have a different `$Xpos` and `$Ypos` from that of their original note. This enables an aliases to hold a different map position on the map to that of its original, be it the same map or a different one.

A number of System attributes are read-only because their values are constantly (re-)calculated. They actually inherit defaults, but these are generally not seen as the calculated value supersedes it.

Next, inheritance vs. hierarchy…

Inheritance is not based on the actual outline of your data

Regardless of how other outliners behave, Tinderbox's attribute value inheritance is *not* connected with a note's outline position. Simply placing one item inside another *does not create inheritance*.

In Tinderbox, notes inherit from their grandchild as easily as from their parent, or indeed from a completely different branch of the outline.

If desired, Tinderbox can be manually configured by the user to make notes inherit values from their parent, but that required using actions rather than normal inheritance. The document's outline hierarchy of content is described separately.

With these factors in mind, it is now time to consider the inheritance cascade…

The Inheritance Cascade

In broad terms, an attribute inherits its value from the element above it in the cascade. Change an attribute value at high level, thus cascades down to all elements below.

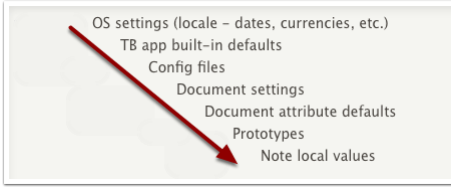
For those used to the functioning of Cascading Style Sheet (CSS) styles in Web pages, the cascade is a similar process. The cascade flows downwards as long as a lower object does not set a value for the same thing, in which case the cascade is broken from above and restarted with a new value for objects further below.

Remember, the cascade is not the outline hierarchy.

The cascade is as follows:

- OS Settings
- Tinderbox app built-in defaults
- Config files—for expert users
- Document Settings
- Document attribute defaults
- Using Prototypes
- Local Values at Note Level
- Note local values can inherit directly or via prototypes

Next, inheritance and prototypes…



OS Settings

Note the OS dialog illustrated tends to get tweaked over successive OS releases so your own Mac's version of the dialog may differ slightly in layout & labelling.

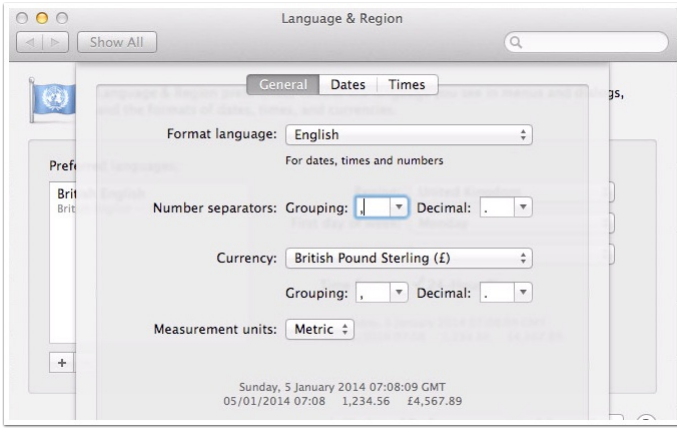
This section is a slight aside, as OS Settings inform how some data types are shown rather than actual attribute values. However, it is pertinent to consider in inheritance terms, as these settings affect how you will see some data displayed in Tinderbox (and therefore dictate the format used to enter things like dates). Here the author is British, so the OS is set to use UK settings. These will affect things like:

- Date formats, including day/month vs. month/day order, date format delimiters (slashes, hyphens, etc.), time formats.
- Currency delimiters (full stop, coma, hyphen, etc.).
- Alpha-numeric sort order: for accented languages this may differ from English.

So, the 'locale' of settings used on your Mac will affect how attribute data is shown and/or edited. The most obvious example is that of dates: on a UK system, 24 March 2014 would be shown or entered as "24/03/2014" whereas for a US locale it would be "03/24/2014". If you change your OS settings, Tinderbox will adapt to reflect the change (if Tinderbox is open during the OS settings change to may need to restart the app).

Tinderbox also has action code support for altering the locale used in certain aspects of the app. If you need this, see the locale() operator.

The true cascade starts with the built-in app defaults…

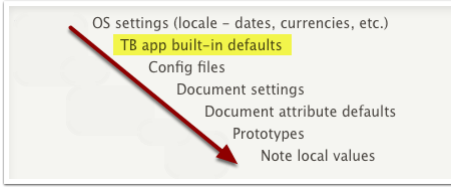


Tinderbox app built-in defaults

When Tinderbox creates a new document, it pre-configures all the System attributes and assigns each a default value (which can be a value of nothing/no value). If no other changes intervene, when a new note is added to the document, it directly inherits those values as the defaults of its own attributes.

The default values for each attribute data-type (e.g. String, Number, Date, etc.) are listed here.

Next in the cascade are config files…

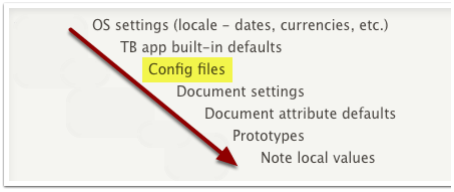


Config files—for expert users

N.B.: these are a feature for the more expert user, so learners and general users can skip straight to the next section, Document Settings, and return to this later if needs be.

Config files are a way of pre-setting some document preferences/attribute defaults so that all new documents receive that customisation. Effectively it allows customisation of a limited number the app's built-in defaults.

Next in the cascade are Document-level Settings…



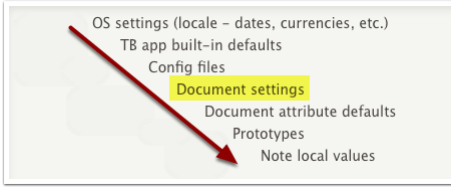
Document Settings

These are effectively per-document preferences that can, if desired, alter the defaults for some System attributes. The Document Settings dialog tabs do not list attributes directly, but rather some of the controls have the effect of altering the built-in default values of some of the System attributes (c.20 in all).

For instance, the font used for all note text is 'MercurySSm-Book', a serif font. But, you may prefer a non-serif such as 'Helvetica Neue'. By changing the Documents Settings ▸ Text tab's ▸ Text Font control to 'Helvetica Neue' you are in fact seeding a new default for the system attribute $TextFont—but only for the *current* document—which tells Tinderbox which font to use when adding text to a new note.

System attributes whose defaults by Document Settings cannot be edited in the Document Inspector's System tab—use the appropriate control in Document Settings.

Next in the cascade are Document system attribute defaults…



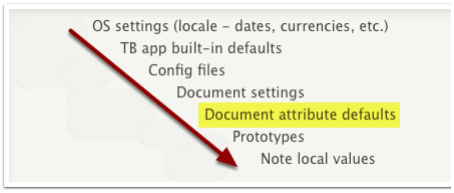
Document attribute defaults

Document attribute defaults are the result of all of the processes already described and are effectively the baseline for attributes' default values once a document is created. System attribute defaults are documented. New user attributes take their data-type defaults.

The document is the highest possible level for configuration of User attributes, as they only exist in a document if added by the user.

Document defaults can always be reviewed—or indeed altered, where allowed—using the Document Inspector's System and User tabs.

Next, the role of prototypes…



Using Prototypes

A way to think of a prototype in Tinderbox is as a pre-customised note. Any note may be made into a prototype at any time (or indeed cease to act as a prototype); more details here. As well as ordinary notes, container notes, agents, adornments and separators may all use a prototype (N.B. prototype adornments can only be seen in map view, just as with normal adornments).

Should a prototype's attribute values differ from higher-level inherited defaults, notes using the prototype inherit the *prototype's* value rather than the higher level default.

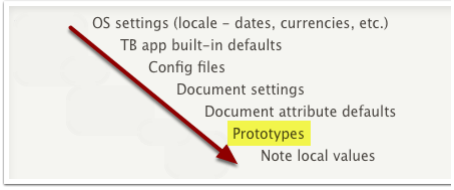
A prototype's customisation is achieved by setting (non-inherited) values for some attributes. These become the inherited defaults for that attribute for any notes using the prototype. Customisation does not need to be visual, though it often helps if some inherited visual feature (e.g. note colour or badge) indicates a note is using a prototype.

A note inherits the contents of a prototype's Displayed Attributes table via the $DisplayedAttributes system attribute.

Remember that intrinsic attributes are excluded from inheritance via prototypes.

The status of a note acting as a prototype is stored in the attribute $IsPrototype. The prototype, if any, that is being used by a note is set in that note's $Prototype attribute. Both settings can easily be viewed/set via the Properties Inspector's Prototype tab.

Prototype attribute values cascade to note 'local' level…

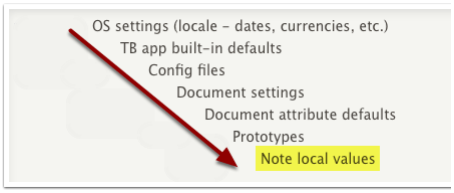


Local Values at Note Level

Lastly, an attribute value can be set in a note. Although other notes may use the same value for the same attribute this is coincidental. The note with a local value is now no longer inheriting a value for that attribute. Note though that inheritance is per attribute. Even if one or more attributes have a local value, all other attributes still retain their inheritance.

For most users, this is the context wherein you undertake the majority most of your editing of attribute values. Attributes can also be altered by action code as well as by manual edit or inheritance.

Note that the cascade does not need to use prototypes …



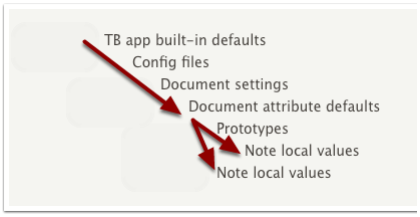
Note local values can inherit directly or via prototypes

Summing up, the original inheritance hierarchy can be re-stated as shown above. Prototypes are an optional part of the inheritance chain - they are part of it only if a note is set to use a prototype†. Otherwise a note inherits from document level defaults.

It has also been shown that whilst local values trump inherited ones, it is still possible to re-enable inheritance, offering a lot of flexibility.

†. Prototypes can inherit from other prototypes, so there may be several in the chain between the document level default and the inheriting note. However, this behaviour is uncommon and only needed in more specialist documents with a highly defined rule-based structure. Intrinsic attribute values are never inherited.

Back to the cascade overview…



Inheritance and prototypes

This section deals with the mechanics of how inheritance works in the context of a prototype. Note: the same general process occurs if attributes are edited at a higher level in the cascade, e.g. at document level, and with or without the involvement of prototypes.

Are prototypes required in a document?

No. There is not a requirement for you to use prototypes at all, or to use them with all notes if present. They are simply an easy and powerful affordance offered by Tinderbox. It is important to understand how they work so you know their effect if you want to use them. Once comfortable with how prototypes work, their usefulness will become apparent.

A note can be turned into a prototype at any point. This is part of Tinderbox's ability to allow for emergent structure (delayed formalisation). In a traditional application, such usage would likely have to be planned and implemented before other content was added, or re-added.

Prototypes can even inherit from other prototypes allowing for quite nuanced behaviours.

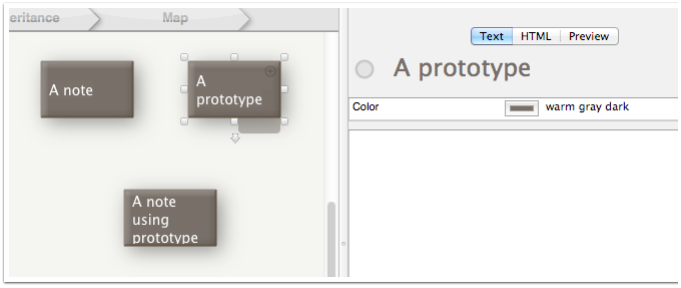
Note: this section uses Displayed Attributes for illustrative purposes. Remember, Displayed Attributes are a means of displaying user-selected attributes. Displayed Attributes have *no special importance* beyond their display in the text pane.

Now to explore the inheritance process:

- Prototypes can change the cascade—I
- Prototypes can change the cascade—II
- Prototype values override document defaults
- Note local values override inheritance
- Can inheritance be restored after setting a local value?
- Re-enabling inheritance—I
- Re-enabling inheritance—II
- Inherited local values without a prototype
- Re-setting all defaults

Prototypes can change the cascade—I

Any note set to use a prototype then begins to inherit the prototype's customisations, as will be seen…



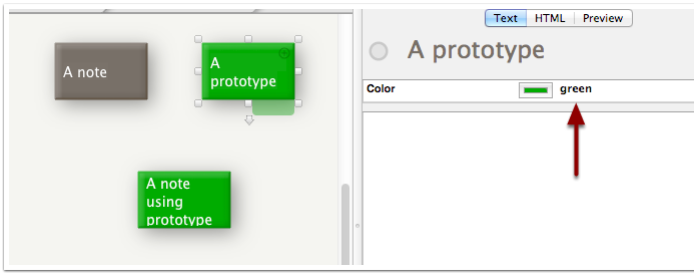
Prototypes can change the cascade—II

Thus if the normal colour of a note (attribute $Color) is 'warm gray dark', and a prototype then has its `$Color` altered to 'green' then the notes using that prototype will inherit that change whereas other notes will not. This holds for any attribute, System or User, that can be altered by the user—i.e. all except read-only attributes.

Also notice that compared to the previous step's image, when `$Color` is displayed as a Displayed Attribute, the name and value were shown in normal weight font but are now in bold. This is a deliberate tell-tale. Bold text indicates that this particular attribute in this particular note is set 'locally' (or immediately'). Local values always override defaults.

The same normal/bold hinting is used in both Displayed Attributes tables in notes (as seen above) as well as in the attribute tables used in the Get Info dialog. One other styling used in these tables is a light grey text for attributes that are read-only, and which cannot be edited.

Now to see how prototype alter the cascade from higher levels…

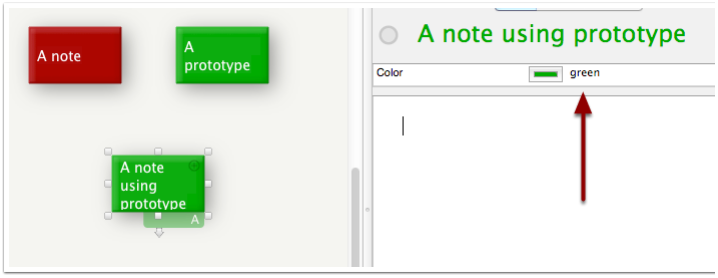


Prototype values override document defaults

Proving the point that the set-in-prototype values override defaults inherited from higher levels in the inheritance chain, if the document default for attribute `$Color` is now set to 'red', notice that only the note **not** using the prototype takes on the revised document default of red. The prototype is coloured green because its `$Color` was expressly changed to 'green', whilst the current note is inheriting a `$Color` value of green via the prototype's `$Color`.

Notice also that for the note using the prototype, the listing for `$Color` is not in bold. This is because, whilst the `$Color` value is *set locally in the prototype*, the same value is simply *inherited* by the note using the prototype.

What about 'local' values, set at a note level…

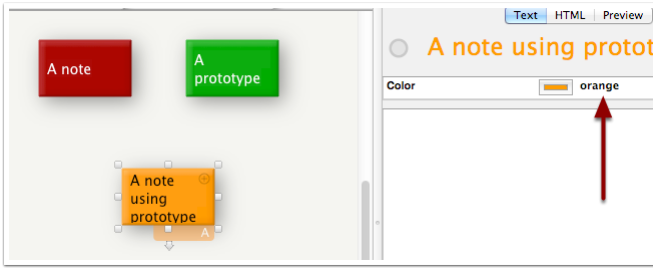


Note local values override inheritance

Consider the diagram above and the note at the bottom of the map. It is using a prototype ('A prototype'), inheriting the `$Color` 'green'. If the the note has its `$Color` set to 'orange', the new colour—by being set local to the note—overrides the prototype-inherited colour. Note the `$Color` data in the Displayed Attributes is listed in bold as it is now set specifically for this note. In Tinderbox terminology this is a *local* value.

You may now see this is also how a prototype alters the cascade. If a prototype sets a local value, it breaks inheritance from above, yet creates a new one below it in the cascade. Thus a local value in a note using a prototype also breaks the cascade, i.e. the note now does not use the prototype's cascaded value.

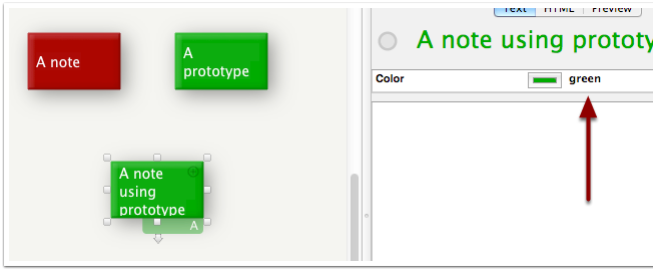
But, can cascaded inheritance be restored? …



Can inheritance be restored after setting a local value?

There is one other important aspect to consider: how to re-enable normal inheritance once a local value is set? What happens if the bottom note's `$Color` is simply set to be the old inherited value? Whilst the bottom note is now green as before, notice the `$Color` data is still in bold. So, the note has the same value as the prototype but is not inherited.

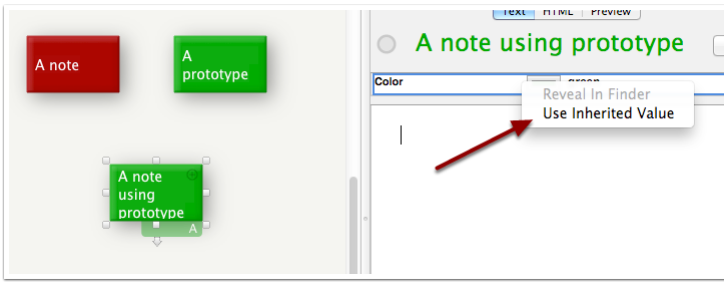
Next…



Re-enabling inheritance—I

The answer to re-enabling inheritance is to right click the relevant attribute in the Displayed Attributes table (in either the caption or value columns) and in the pop-up menu then click 'Use Inherited Value'. This resets inheritance. Will the note be red or green?

Next…



Re-enabling inheritance—II

It is 'green', inheritance is restored. As the `$Color` entry is now not bolded, it is signalling that the 'green' value is now inherited from prototype 'A prototype'.

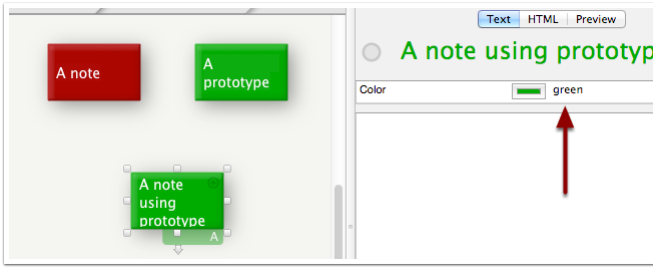
Because the prototype's own `$Color` is set locally, that blocks it inheriting the document's 'red' default. Another way to prove the point is to remove the prototype assignment from the current note.

To reset inheritance in action code, set a (null) value of ';' , thus:

```
$Color =;
```

This method is described further under the Actions & Rules section.

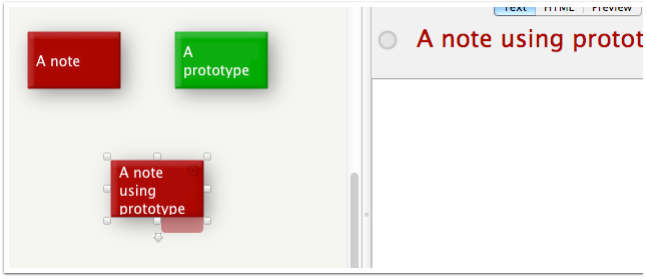
Remember, inheritance can occur without prototype involvement…



Inherited local values without a prototype

Here, without a prototype, the bottom note inherits the altered document default colour of 'red'. Note too, that `$Color` is not shown as a Displayed Attribute any more, even though the bottom note has focus. Why? Because that too was a customisation inherited from the prototype, showing that inheritance involves more than just aspects as obvious as a note's colour.

It is also possible to re-assert the full cascade…



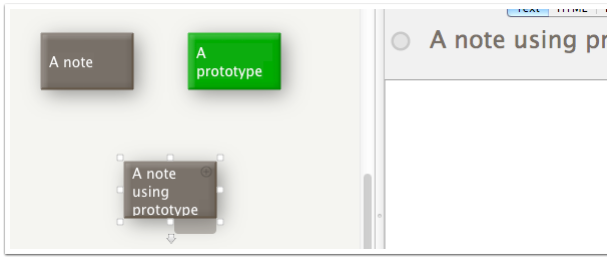
Re-setting all defaults

For completeness, restoring the document-level `$Color` default back to the standard new document default re-applies a value of 'warm gray dark'.

This completes the walk-through of the attribute value cascade. Hopefully it has shown the flexibility of the cascade. That is one of the aspects of Tinderbox that makes it so flexible in terms of implementing structure as it becomes apparent, rather than having to know in advance. In the same way, if a false start is made, it is quite easy to roll back changes.

As should now be apparent, using prototypes to set values and insert them into the cascade (or remove them from it) can rapidly alter the nature of your document. I the preceding example the colour of a note was used because it is easy to illustrate in a screen-grab, but remember that prototypes can be used to set any non-read-only attribute.

Back to inheritance and prototypes overview…



Data export

Tinderbox offers extremely flexible export, both in format and scope.

The is all covered in the section on Export.

Data import

Tinderbox can both import data via a variety of means though there is no primary 'Import' command or mechanism. Broadly, Tinderbox accepts drag drop of plain text data including tab-delimited text. There is also support for data interchang with some specific applications.

Data import is explored in detail in the section on Import.

Unicode

Since Tinderbox version 5.x, Unicode is supported in note text ($Text) and all system/user attributes. From version 6.x, all UI elements support Unicode display and input.

Data exported from Tinderbox will use Unix-style line ends, i.e. line feed character. In Action code this is inserted as the string `"\n"`.

Inline images

Tinderbox can embed and show bitmap images in two places:

- In note body text. Such images are exported via HTML export.
- Map image adornments. These cannot be exported but the overall View pane can be exported using the 'Copy View as Image' Edit menu option.

These images are embedded into the Tinderbox document and form part of the data (and stored size) to the TBX file. Depending on the nature of a TBX document, it may make sense to store large images externally and link to them.

If the TBX is primarily intended for export, such as this document, it may make more sense to store the images outside the TBX and link the exported pages to the external assets. Given that embedded images do not export (at least not currently) then it is possible for a note to have both an embedded image for viewing within Tinderbox and separately alongside it export code that links to an externally stored copy of the image for use with HTML output.

Tinderbox compresses images; this substantially decreases the size of Tinderbox files with embedded images. Tinderbox currently uses JPEG compression, and compresses fairly aggressively; this significantly diminishes the size and loading time of Tinderbox files at the cost of some image degradation.

Another way images may be used as fills for Map view notes and adornments and the view background (see $Fill and $MapBackgroundFill).

Fonts: italics and bold

Using current Apple frameworks, italics and bold are generated by using the italic or bold font variants of the normal version of the specified typeface. Thus if $TextFont is Helvetica, italic text is rendered in Helvetica Italic and not a false italic Helvetica; similar for bold text.

Some fonts well used in the past as Tinderbox defaults may lack these font variants. For instance, macOS currently has no italic variant of Lucida Grande, as was used for $TextFont in earlier versions of Tinderbox. This can be problematic when opening TBX documents created using the latter as italic text and titles are rendered as normal text. In particular this makes it difficult to tell aliases apart from original notes.

As $Text is saved as Rich Text (i.e. with embedded style information) it is not possible to recover 'lost' style caused by the above issue and it will need to be re-applied manually. For title contexts (e.g. aliases styled in italics), there are two possible solutions.

- If a permanent change of font is acceptable, change to one that has both italic and bold variants. If in doubt, use the v7 defaults for the relevant attribute.
- If the existing font is still desired, there is a document setting that allows for aliases to be rendered in underlined normal fonts in addition to the italic styling; i.e. if italics are available the title is underlined and in italics, otherwise it is only underlined).

Coding

To quickly find listings of system attributes, action code operators, designators and export codes, please use the quick links at top and bottom of the page.

This section contains:

- Use of Attributes
- Use of Agents
- Action Code
- Links
- Macros
- Export Codes
- Tinderbox URL schema

Use of Attributes

Aside from what an attribute is as a concept within Tinderbox, it is useful to understand how code interacts with attributes and how choices about attribute type and name can affect their use. See:

Attribute Naming

Attribute names, whatever their type, are:

- Case-sensitive. 'pronoun', 'ProNoun', 'Pronoun', 'PRONOUN' are all valid names that if created would be 4 discrete attributes.
- The first character must be a letter (upper or lower case) from the Roman alphabet, i.e. A to Z. It must **not** be a number or an underscore. The initial letter **can** be accented (bearing in mind that at a computer code level, accented letter are a *different* character) and may be a non-roman script character.
- The name must include at least one character (see further below for why single-character named attributes are not generally a good idea).
- Until v6 only Roman letters and numerals, plus the underscore character, were permitted in attribute names. So, the allowed characters are:
  - capital A to Z (including accented characters), including non-Roman scripts.
  - lowercase a to z (including accented characters), including non-Roman scripts.
  - numerals 0 to 9 (not recommended - see below)
  - underscore
- non-Roman alphabet Unicode characters may be used for all except the first character. This can include characters from double-byte languages, but right-to-left oriented languages (e.h. Hebrew, Arabic) should be avoided; in both thes cases check carefully before committing to extensive use.

The Document Inspector's User tab for making new user attributes will suppress keyboard input for any other characters to help enforce this rule. So you may have 'ToDo' but not 'To-Do', or To Do' as an attribute name. A name of '8' or '009' is not allowed. When typing a new attribute name in the Inspector, or changing an existing one, unsupported characters (in the position as typed) appear in red. The attribute's name cannot be saved if any red characters are showing.

When importing spreadsheet type data tables and new attribute names are coerced from column names, an underscore will be used to replace any unsupported characters (e.g. accented characters).

Take great care when importing data with numerical column heads, e.g. '2' or '37'. If the attribute names and default value (likely zero) differ this can confuse Tinderbox's legacy support in code parsing. Thus id attribute '2' has a default value of zero, `$SomeNumber + 2` may be misinterpreted as `$SomeNumber + $2`, i.e. `$SomeNumber + 0`, and thus no change. Or, if the $2 in context has a value of '10' then the code might parse as `$SomeNumber + 10` o not as expected. S number-only attribute names are allowed, but liable to cause problems so are an ill-advised choice unless circumstances dictates so.

User Attributes: suggested naming style

Although you may use any letter case combination it is worth noting that the general style in Tinderbox, and examples in the manual etc., tend to one of two styles opening capital ('Prototype') or inter-capitalised ('AgentPriority'). For newer users, adopting the latter form for user attributes will help to distinguish attributes when seen in action and export code - see more on suggested naming.

Perhaps self-evident, but adopting self explanatory names is useful both in writing codes but also when displaying Displayed Attributes. Also make use of the attribute's Description box, to describe the attribute's purpose (do not rely on you memory some years later).

Further naming considerations

User Attributes can be renamed (but not system attributes).

Single character names (i.e. single letter - see list above) are allowed though are impractical and not recommended. Such brevity is likely to cause error at some point.

NOTE re Attribute names & the $-prefix. In action code, when referring to an attribute's value a $-prefix is used. Thus to call the value if the attribute WordCount, $WordCount is used. To refer to the name of said attribute, use "WordCount". Within aTbRef, inline mention in general text which refer to Tinderbox attributes names generally use a $-prefix as a marker that it is an attribute name being cited. The latter is a slight variance with in-app usage but does aid clarity in the articles.

Attribute Data Types

All Tinderbox attributes use a specific data types. If not the author of the document, there are ways to determine an attribute's data type. In general descriptive usage, and in the UI, Tinderbox's labelling of data types is case insensitive, i.e. 'number' or 'Number' are the same in meaning.

There are 14 types of attribute data types. Note that Action and Font types are not available for use by User attributes.

An Unsigned type is listed but not used or currently described, it should be regarded as experimental. Experimental attribute types appear from time to time and are usually adopted formally or removed; the release notes normally give detail on these types.

Note that Action, Color, File, Font, List, Set and URL data types are effectively all special forms of String data type attributes and share a common default: an empty string. Boolean attributes default to false (which is not a string), Date attributes to the string "never", Interval attributes to the string `"00:00"` and Numbers to `0` (zero). Empty/Default values are discussed further for each action type in the articles linked in the list below.

A combined list of default values for each individual data type can be found here. See per-type listings for detail of the value sorting order of that type.

These are the current Tinderbox attribute data types

- Action-Type Attributes
- Boolean-Type Attributes
- Color-Type Attributes
- Date-Type Attributes
- Dictionary-Type Attributes
- Email-Type Attributes
- File-Type Attributes
- Font-Type Attributes
- Interval-Type Attributes
- List-Type Attributes
- Number-Type Attributes
- Set-Type Attributes
- String-Type Attributes
- URL-Type Attributes

Action-Type Attributes

Action

The attribute data type of 'action' represents a string that can be used as an action; under the hood, Tinderbox caches the compiled action so it does not have to constantly re-parse the same expression. Thus these attributes should contain string [sic] of valid Tinderbox action code.

Some pre-existing system attributes, such as OnAdd and Rule have been re-classed from 'string' to 'action' type.

This data type is not available for user attributes. Use 'string' instead.

Default/Empty value

An empty string.

Sorting order

As for String Data Type.

Action-type System Attributes

Built-in attributes of the action data type are listed below:

- $AgentAction
- $AutoFetchCommand
- $BeforeVisit
- $DisplayExpression
- $Edict
- $HoverExpression
- $OnAdd
- $OnJoin
- $OnRemove
- $OnVisit
- $Requirements
- $ResetAction
- $Rule
- $TableExpression

Boolean-Type Attributes

Boolean

A binary value: either `true` *or* `false`. These values are case sensitive. If no default value is specified, a Boolean type defaults to a value of `false`.

In the Tinderbox UI, Boolean attributes may often be represented as tick-boxes, e.g. in a note's displayed attributes table. A ticked box equates to `true`, and and empty box to `false`.

When coercing a Boolean value—such as an expression result—to Number-type of data, `true` coerces to 1 and `false` coerces to 0 (zero).

Default/Empty value

A value of `false`.

Sorting order

`false` (un-ticked), then `true` (ticked).

Boolean-type System Attributes

Built-in attributes of the boolean data type are listed below:

- $AgentCaseSensitive
- $AutoFetch
- $AutomaticIndent
- $BadgeMonochrome
- $Checked
- $Direction
- $DisplayExpressionDisabled
- $EdictDisabled
- $HideDisplayedAttributes
- $HideKeyAttributes
- $HideTitle
- $HTMLDontExport
- $HTMLEntities
- $HTMLExportChildren
- $HTMLFileNameLowerCase
- $HTMLMarkDown
- $HTMLMarkdown
- $HTMLMarkupText
- $HTMLOverwriteImages
- $HTMLQuoteHTML
- $IsAction
- $IsAdornment
- $IsAgent
- $IsAlias
- $IsComposite
- $IsMultiple
- $IsPrototype
- $IsSeparator
- $IsTemplate
- $LeafDirection
- $Lock
- $MapBackgroundShadow
- $MyBoolean
- $NameBold
- $NameStrike
- $NeverComposite
- $NoSpellling
- $OutlineColorSwatch
- $Private
- $PrototypeBequeathsChildren
- $ReadOnly
- $RuleDisabled
- $Searchable
- $Separator
- $Shadow
- $ShowTitle
- $SmartLinks
- $SmartQuotes
- $SortBackward
- $SortBackwardAlso
- $Sticky
- $TextSidebar
- $TimelineAliases
- $TimelineDescendants
- $TimelineMarker
- $UpdateTextLinksAfterRename
- $ViewInBrowser
- $Ziplinks

Color-Type Attributes

Color

Any one of the following:

- Hexadecimal. A colour represented as #RRGGBB; a hash sign followed by 6 hexadecimal digits, a pair for the intensity level 0 to 250 in hexadecimal of Red, Green and Blue. This is the same format used for describing colours in HTML. Example: `#A482BF`. The short form is also accepted, e.g. #f00 for #ff0000, where the two hex characters of each per-colour value are identical and the second character is omitted. Bare hex strings in action code, i.e. unquoted number preceded by a hash sign, are correctly identified as hex input values
- Named Colours. The name of a colour that either comes pre-defined in Tinderbox, or which you have defined for this Tinderbox document using the Colors pane of the attribute palette. Examples: blue, red, minty-fresh-green. When setting a shade of a named colour in a string the shade value (lighter, darkest, etc.) comes before the colour, unlike listings in Info view and the Inspectors; thus "dark warm gray" not "warm gray dark", "light cyan" not "cyan light".
- HSV (Hue-Saturation-Value) value. Hue: a value from 0 to 360 degrees. Saturation and Value: a value from 0 to 100 percent. Enclosed in parentheses and preceded by 'hsv'. Examples: `"hsv(0,100,50)"`, `"hsv(240,80,80)"`.
- RGB (Red-Green-Blue). Intensity levels represented as integers from 0 to 250. Enclosed in parentheses and preceded by 'rgb'. Examples: `"rgb(0,0,0)"` and `"rgb(68,153,68)"`. This should not be confused with the rgb() action operator (which sets a hex format value string (see first bullet above).

When displayed in a note's displayed attributes table, Color-type attributes show a colour-chip icon before the attribute value. Pressing the icon opens an OS colour picker dialog.

If no default value is specified, a Color type defaults to an empty string (which is treated by Tinderbox as the value 'normal'); in Displayed Attributes table displays a white colour chip is displayed. In some cases, Tinderbox will show a dark red colour as a 'not initialised' colour value; it is hex colour code #6f0000, though its derivation as a value is not clear.

Note that as 8-bit colour settings are translated within the app into 16-bit colours, rounding errors can occur. So, in some cases—most likely using HSV and HSV based transforms—the values used might not correspond precisely to those set by the user.

Similarly, using format() with a colour attribute will always result in a hexadecimal value string regardless of the stored value. Thus it makes sense to export colour attributes as ^value(format($ColorAttr))^ or ^value($ColorAttr.format())^. However, note that coercing a default (blank) colour to a string value using format() or .format() will result in the colour value "#ffffff", which is not strictly the blank value. So, be wary of using that formatting as a means to check for a blank colour value; better is to use a short form Boolean test.

Use in Export code

When accessed via ^value()^ export code, it will return either a hex colour value or a named Tinderbox colour, depending on what is set for the destination attribute.

Legacy only: using the deprecated ^get()^ or ^getFor()^ export codes, the colour is always returned as a hex figure including a # prefix, e.g. "#FF9900".

Default/Empty value

An empty string.

Sorting order

Named colours in decreasing lexical sort order, then #-based codes in decreasing lexical sort order with same valued three-character codes sorting after six-character ones (i.e. #ffffff before #fff).

Color-type System Attributes

Built-in attributes of the Color data type are listed below:

- $AccentColor
- $BorderColor
- $CaptionColor
- $Color
- $Color2
- $GridColor
- $HoverBackgroundColor
- $MapBackgroundAccentColor
- $MapBackgroundColor
- $MapBackgroundColor2
- $MapBodyTextColor
- $MapPrototypeColor
- $MyColor
- $NameColor
- $OutlineBackgroundColor
- $PlotBackgroundColor
- $PlotColor
- $PrototypeHighlightColor
- $ShadowColor
- $SubtitleColor
- $TextBackgroundColor
- $TextColor
- $TextColorBlue
- $TextColorGray
- $TextColorGreen
- $TextColorRed
- $TextHighlightBlue
- $TextHighlightGreen
- $TextHighlightMagenta
- $TextHighlightRed
- $TextHighlightYellow
- $TimelineBandLabelColor
- $TimelineColor
- $TimelineGridColor
- $TimelineScaleColor
- $TimelineScaleColor2
- $TitleBackgroundColor
- $TitleForegroundColor

Date-Type Attributes

Date

A date *and time* string. It should always be enclosed in double quotes (legacy code without these may cease to work as expected). Dates can be added/subtracted/compared in mathematical fashion, as per number data types. The resulting 'string' can be formatted by means of date format codes. If no default value is specified, a Date type defaults to a value of never. If a date is specified but not a time, time always defaults to current system time except the seconds are not shown in Displayed Attributes (visual format may vary by locale, e.g. on the use of the colon delimiter).

Time is used in date comparisons except when the == or != operators are used. Also see Basic Comparison Operators and date comparisons; the same rules hold true for data comparisons in action code expressions. Seconds are acknowledged in comparisons, except those excluding all time elements of a date/time (Previously, seconds were not used and/or are always considered to be 00. To use exact equality tests on full date-time data use interval()).

From v9.5.0, adding or subtracting Interval data from Date data whose value is "never", correctly returns "never".

Tinderbox recognises date keywords such as **today**, **now**, **yesterday**, **tomorrow**, **day**, **week**, **month**, **year**, **hour** and **minute** as date-defining placeholders when working in action code (see date keywords link for a full list). Constructions like `"yesterday + 1 week"`, `"tomorrow + 1 year"` or `"tomorrow - 2 hours"` are also possible; note how the whole expression is placed in one set of double quotes. If number is supplied without a date placeholder, e.g. "today + 1", then 'day' is assumed as a default unit value, i.e. as if the input was "today +1 week". See more on the syntax for setting/adjusting dates using date keyword strings.

The 'now' placeholder can be used as an alternate for 'today'. As both the latter use the current system clock time for their time element, note that it is possible to set an explicit time alongside designators (only use 24-hour clock time: `"today 16:00"` or `"tomorrow - 2 days 23:59"`. See more on setting times.

Initialising a Data attribute without time information

Beware the use of only date placeholders, without time indicators, to initially set a Date attribute will result in the time portion (hours:minutes) being set to the users local OS system time. Entering a time-less date into a Displayed Attribute value box, e.g. "21 Aug 2001" will also result in the time portion being set from the OS local time. Using a date designator to alter an existing date never alters the time part of the date attribute value. The time part of a date can be altered either by manually (re-)setting it or by use of time-related action codes. If current time is 17:03:24 (5 PM, 3 minutes and 24 seconds):

`$StartTime = date("27/03/2022")` gives a time of 17:03:24 on 27 March 2002 (US users only, reverse day/month order)

`$StartTime = date("27/03/2022 00:00:00")` gives a time of 00:00:00 on 27 March 2002 (US users only, reverse day/month order)

Similarly if initialising a Date attribute by typing in a Displayed Attributes or Get Info value, you must explicitly include the h/m/s time unless you wish to use the default (system) time.

Date Comparisons

Considering date vs. date-time in queries is discussed here.

Effects of OS locale. Tinderbox will assume a manually supplied version uses the same day/month order as the user's current locale settings. In other words, if working on a US system using month/day order enter 6 July 2001 as "7/6/2011" whereas on a British system using day/month order use "6/7/2011", and so on for other locales. Dates that use slashes "1/7/05" and the Continental format with periods, e.g. "1.7.05" are both recognised.

For years with only one or two digits, e.g. 10CE, you should always enter the year part of the dates as 4 digits (e.g. 0010) when setting dates via code or Displayed Attribute input.

Users should check the Formats pane of the International pane (Formats tab) of their macOS System Preferences and ensure the 'short' format uses four-digit year numbers. To do this (for a region), on the Formats tab, click the the 'Customize…' button to the right of the example dates. The 'Show' pop-up is most likely set to 'Short'; leave it as that. In the white text box below click on to year; you will see a drop-down menu that will allow you to choose either 2-digit or 4 digit dates, select the four-digit date. US users should note the default US setting for "short" date formats is to display 2-digit years, i.e. 10/3/65 (for 1765, 1865, etc.). All numeric reverse-order dates like '20090130' can be used though do tes the exact desired format before committing to extensive use.

Use of negative dates

Tinderbox will allow negative dates to allow analysis of data either side of the CE/BCE (AD/BC) boundary. Negative integers from -1 to -2500, when coerced to dates, are interpreted as years BCE (BC). Some current limitations re 'negative' dates:

- Currently, there is no way in a Displayed Attribute box or in display expressions to display the fact that a date is negative.
- Negative dates cannot be tested for using greater than/less than comparisons.
- Negative dates cannot easily be set via a Displayed Attribute input. The easiest method is to make a stamp with code like this: `$StartDate.year = ($StartDate.year-(2*$StartDate.year)-1)`. That code will flip a positive date to a negative one. Given the first limitation above re displaying BCE dates, you will need to check out that the result is correct.
- Date.year always returns a positive number, even from a negative date.
- There are no date-format strings that will show the epoch.

Miscellanea about Date-type data

Tinderbox XML documents store dates using ISO 8601 format. Tinderbox also recognises date strings entered as Displayed Attributes or converted from strings in actions when in the ISO 8601 format (e.g. 2008-11-20 16:55:00).

Two dates may be subtracted with a result in days even if the result is a string. This helps with tasks like setting $DisplayExpression.

Tinderbox offers numerous date formats to allow non-default representations of date/time both in internal action code (use with format()) or for export (use with ^value(format())^).

From v9.6.0, the Date conversion of integers has been modified for better handling of prehistoric dates. When integers are converted to dates, the conversion is as follows:

- -35000 to +2500: refers to a year between 35,000 BCE and 2500 CE.
- otherwise: refers to the number of seconds before or after midnight UTC on 1 Jan 1970.

Default/Empty value

The string "`never`" (no date set).

Sorting order

Ascending (later) date/time order; "never" always sorts last.

Storing date/time durations

This is done using the Interval data type. The maximum unit size in an Interval is a day and it can work for up to 30 days. But generally it is for periods of a day or or less thought of in hours, minutes and seconds.

Doing date/time calculations with Date and Interval types

This is described in discussion of the Interval data type and here.

Date-type System Attributes

Built-in attributes of the date data type are listed below:

- $Created
- $DueDate
- $EndDate
- $LastFetched
- $Modified
- $MyDate
- $NotesModified
- $SimplenoteModified
- $SourceCreated
- $SourceModifed
- $StartDate
- $TimelineEnd
- $TimelineStart

Dictionary-Type Attributes

Dictionary

A Dictionary attribute replaces like the older Tinderbox feature of lookup tables, as a list of paired **key**s with **value**s: `key:value;`

The **key** part comes first with a terminating colon. The **value** part follows, terminated by a semi-colon. White space handling in key:value pairs is described below.

Dictionaries are faster to construct, and large dictionaries are far faster to check, than lookup tables. Dictionaries are not generally intended for handing large/complex values or for values including punctuation and symbols. A Dictionary may be considered better for tasks previously configured using look-up lists.

Like other attribute types, a Dictionary follows normal rules of scope and inheritance. A note using a prototype with a Dictionary holding data, will inherit that data dictionary. Similar to a Set type, a Dictionary does not allow duplicate keys, but multiple keys may use the same value. A note using a prototype with a dictionary holding data, will inherit that dictionary.

A key may have *multiple* values; this can be either []-enclosed List-type or {}-enclosed Dictionary-type data. For example, the dictionary:

```
{Tinderbox: 1;Storyspace: {Editor: 2; Reader:3}}
```

contains two elements. The first has a key of "`Tinderbox`" and a value of `1`. The second has a key of "`Storyspace`" and the value that is itself a dictionary, `{Editor: 2; Reader:3}`.

Dictionary key:value syntax

A dictionary collects pairs of strings separated by a colon. The first string is the *key,* and the second string is the *value.* The **dictionary()** operator creates a dictionary (in the current note), here with 3 such key:value pairs. The first form is best practice:.

```
$MyDictionary={cat:animal; dog:animal; rock: mineral};
$MyDictionary=dictionary("cat:animal; dog:animal; rock: mineral");
```

The **key** "cat" has the **value** "animal", while the **key** "rock" has the **value** "mineral".

Although it is still possible, via legacy support, to populate a directory by passing a dictionary-formatted list as a quoted string to a Dictionary-type attribute, this is *deprecated* in favour of the `{}` method, or using the **dictionary()** operator is explicit and indicates intent unambiguously when parsed by Tinderbox. See dictionary() for usage.

Case-sensitivity

A **key** is *case-sensitive* and must be unique to the dictionary; the **value** may be the same as values for different keys within the current dictionary. Rather like the Set data type, duplicates are weeded. If a new key:pair is added to a dictionary and the key (case-sensitively) already exists, the existing value (only) for that key is updated to use the value from the new pair.

```
$MyDictionary = dictionary("cat:animal; dog:animal; rock: mineral");
$MyDictionary = $MyDictionary + "cat:mammal";
```

Now, the value of key "cat" becomes "mammal" and the older value is lost.

Use of whitespace, symbols, quotes, etc., with keys & values

Any whitespace either side of a colon (:), or semi-colon (;), or at beginning or end of dictionary data is ignored. As the colon (:) and semi-colon (;) characters are used as delimiters in the stored Dictionary data, they should also not be used within key names or values.

The expectation is that keys and values will be simple strings or numbers, so avoid use of characters like punctuation as you may experience unexpected results; additionally avoid using parentheses (), square [] or curly {} brackets, backward and forward / slashes, single/double quotes or commas.

A good rule of thumb is to think hard if using other than `A-za-z0-9` or underscore/hyphen/period (the period might be needed for a decimal number value).

Adding key:value pairs

To add to a new key:value pair, use addition, as with Lists and Sets…

```
$MyDictionary = $MyDictionary + {apple:plant};
$MyDictionary += {apple:plant}";
```

adds they key "apple" and associates it with the value "plant". If the key was already found in the dictionary, its value is replaced by the new value. If the key was not found in the dictionary, both the new key and the new value are added.

As well as using a literal value, the key (and value) to be added can be give as an attribute value or variable:

```
$MyString = "apple:plant"; $MyDictionary += $MyString;
var:string vKey = "apple:plant"; $MyDictionary += vKey;
```

Deleting keys (implicitly key:value pairs

Entries can be deleted from a dictionary by subtracting the key:

```
$MyDictionary = dictionary("dog:animal; cat:animal; rock:mineral");
$MyDictionary = $MyDictionary - "dog"; // gives "cat:animal; rock:mineral"
```

or in newer form:

```
$MyDictionary -= "dog"; // gives "cat:animal; rock:mineral"
```

Be aware that setting an empty value for a key doesn't delete the key:

```
$MyDictionary[somekey] = "";
```

leaves the key 'somekey' with no value, with (qv below) is not encouraged.

As well as using a literal value, the key to be removed can be give as an attribute value or variable:

```
$MyString = "dog"; $MyDictionary -= $MyString;
var:string vKey = "dog"; $MyDictionary -= vKey;
```

It is not possible to subtract a list of keys. Instead, such a list must be iterated as a succession of individual key deletions. Alternatively simple set the key's value to a new literal string/list/dictionary, thus completely replacing the existing value(s).

Changing the value of a key

New values may be assigned to specific dictionary keys.

```
$MyDictionary[apple] = "pie"
```

Adds the key "apple" to $MyDictionary with the value "pie". If the dictionary previously contained a value for "apple", it is replaced by "pie"; if not, a new key and a new value are added to the dictionary. From v9.1.0, .add() can be used:

```
$MyDictionary.add("apple","pie");
```

Note that the += and -= operators are not currently available for changes to dictionary (numerical) values of the form

```
$MyDictionary[key] += 1;
```
WRONG!

Instead, use the conventional form:

```
$MyDictionary[key] = $MyDictionary["key"] + 1;
```

Deleting the (single) value of a key

Ideally in such a case remove the key. However is possible to delete a key's value, by setting it to an empty string "" (or reading from a stored value that is such). To set the value of key 'apple' to "":

```
$MyDictionary[apple] = ""
```

Generally, the expectation is that a key has an actual value .

Keys with multi-item values are supported using [] or {} notation

From v9.6.0, Dictionary key values can be multi-valued using either the new `[]` (list/set) or `{}` (dictionary) notation. This holds for creating dictionaries or setting key values. When retrieving a multi-value result pass the data to an appropriately type attribute or variable. The data can then be accessed using appropriate key or list address syntax. Examples of nested multi-value values:

```
$MyDictionary={cat:animal; dog:animal; rocks:{granite:mineral;basalt:mineral}};
$MyDictionary={cat:animal; dogs{terrier:dog;labrador:dog}; rock: mineral};
```

Legacy behaviour: Prior to v9.6.0, a value could not contain more than one value The value for a key 'pie' might reasonably be apple, lemon and quince. Whilst it was possible to define a dictionary like this:

```
$MyDictionary["pie"] = "apple;lemon;quince"
```

… the fact a semi-colon is used as both a list delimiter (within the **value**) and as a **key**:**value** pair delimiter, means Tinderbox *currently* does not know how to parse this; work is in hand to support multi-item value, i.e. lists.

Temporary note. At present (v9.6.1) generating a dictionary with nested multi-values (list/dictionary) is possible only if using literal string values but not if using variables/attribute values. This is likely a by-product of the new [ ] and { } notations and subject to change/fix.

Merging Dictionaries

Dictionaries may be merged by adding them.

```
$MyDictionary = $MyDictionary+"apple:plant"
```

adds they key "apple" and associates it with the value "plant". If the key was already found in the dictionary, its value is replaced by the new value. If the key was not found in the dictionary, both the new key and the new value are added.

It is possible to add two dictionaries.

```
$MyNewDictionary = $MyDictionary+$MyOtherDictionary
```

Note that if the dictionaries share (case-sensitive) keys that have differing values, the values from the last-added, therefore those in $MyDictionary, will replace those in $MyDictionary. Thus:

```
$MyFirstDictionary = dictionary("apple:fruit; granite:mineral");
$MyOtherDictionary = dictionary("pear:fruit; granite:rock");
$MyDictionary = $MyFirstDictionary + $MyOtherDictionary;
```

MyDictionary now contains "granite: rock; pear: fruit; apple: fruit"

Note the order of dictionary items is re-ordered, reinforcing the point that additions/subtraction of dictionary data may alter item order, and which the user does not control. It is not possible to:

- subtract a dictionary from a dictionary
- subtract lists of keys
- subtract individual values

Dictionaries lack the 'default' key used by look-up lists

User of look-up lists with list-type data might assume the same notion holds for Dictionary-type: it does not! It is still possible to define a 'default' key but using it requires a little extra code. Whilst it is possible to check if a key exists with Dictionary.contains(), it may be more useful to check for a default empty ("") value being returned. For instance, instead of, in a `.each(aKey){}` loop, where you might code:

```
$MyString = $MyDictionary[aKey];
```

consider (and assuming a 'default' key/value exists):

```
if($MyDictionary[aKey]){$MyString = $MyDictionary[aKey];}else{}$MyString = $MyDictionary[default];};
```

or:

```
if($MyDictionary.contains(aKey)){$MyString = $MyDictionary[aKey];}else{}$MyString = $MyDictionary[default];};
```

But, the key point is that is a default response is needed, the user must (a) define the 'default' key:vale pair and (b) add the code to invoke it. More code-savvy users may choose to encapsulate this in a user function.

Dictionary and values()

You can get the values of a Dictionary, but you cannot look up a key that corresponds to a given value. Remember, multiple keys may same value. Therefore, using values() with dictionaries is not recommended as this returns all the unique key:pair values as opposed to only keys or only values.

Iterating Dictionaries

Use Dictionary.keys to get a list of the keys and then iterate that list using List.each().

Default/Empty value

If a requested key is not defined, or the key has no value, an empty string is returned.

Sorting order

Not applicable. As a Dictionary is addressed via a key value rather than a list offset, the stored order for keys is moot. By observation, a Dictionary is generally stored in key lexical sort order but this should not be assumed to always be the case..

Additional Dictionary definition mark-up

From v9.5.0, Dictionaries gain additional new mark-up. Dictionaries may be written by enclosing them in braces, for example a dictionary with two key:value pair elements:

```
$MyDictionary = dictionary({Dog: animal; Crocus: plant});
```

The use of the braces removes the need to use quotes around string values.

This new form of Dictionary-type data allows for nesting; for example, the dictionary

```
{Tinderbox: 1;Storyspace: {Editor: 2; Reader:3}}
```

contains two elements. The first has a key of "`Tinderbox`" and a value of `1`. The second has a key of "`Storyspace`" and the value that is itself a dictionary, `{Editor: 2; Reader:3}`.

The dictionary .add() and .extend() operators now take a single argument — a dictionary of elements which will replace or extend the current elements.

Dictionary-type System Attributes

Built-in attributes of the date data type are listed below:

- $MyDictionary
- $ReferenceDictionary

Email-Type Attributes

Email

From v9.6.0, an Email attribute data type is offered.

An email address string, e.g. `jdoe@example.com`. If no default value is specified, an Email type defaults to an empty string. Emails are not delimited (i.e. quotes are not required).

Email attributes behave like string attributes, but with a special behaviour when displayed in a note's displayed attributes table. There, Email-type attributes show an envelope icon before the attribute value. Pressing the icon opens a draft email to that address using the default email client using the notes text as the draft content.

More precisely, pressing the button composes a message, whose body is the $Text of the note, and opens that message in the default email client (e.g., Apple Mail), from which it might be sent to the address that appears in $Email. There is no validation that the value in $Email is correctly formed.

Default/Empty value

An empty string.

Sorting order

As for String Data Type.

Email-type System Attributes

Built-in attributes of the Email data type are listed below:

File-Type Attributes

File

A path to a file. When setting this type of attribute a file selection dialog is shown instead of a text box. The selected file is entered as the attribute value for that note. If no default value is specified, a File type defaults to an empty string.

The path stored is in POSIX form (forward slash delimited) and is the full local path to the file or folder.

When files from the user's directory are dropped into Tinderbox or into a file attribute, Tinderbox uses the tilde (~) abbreviation to represent the path to the user's directory. This should make it easier to share one Tinderbox document across several machines, each of which share files in common locations (e.g. Dropbox) but which have different user names.

When displayed in a note's displayed attributes table, Find-type attributes show a folder icon before the attribute value. Pressing the icon opens that attribute's stored OS file path (if any) for a file, using the in the default app as determined by Finder, or for a folder it opens Finder.

Default/Empty value

An empty string.

Sorting order

As for String Data Type.

File-type System Attributes

Built-in attributes of the file data type are listed below:

- $File
- $Fill
- $HoverImage
- $RSSChannelTemplate
- $RSSItemTemplate
- $WatchFolder

Font-Type Attributes

Font

The attribute data type of 'font' represents a string defining a font name.

Some pre-existing system attributes, such as $NameFont have been re-classed from 'string' to 'action' type.

This data type is not available for user attributes.

Default/Empty value

An empty string.

Sorting order

As for String Data Type.

Font-type System Attributes

Built-in attributes of the file data type are listed below:

- $DisplayedAttributesFont
- $CodeFont
- $TextFont
- $HTMLFont
- $AdornmentFont
- $KeyAttributeFont
- $NameFont
- $CaptionFont
- $TitleFont
- $HoverFont
- $GridLabelFont

Interval-Type Attributes

Interval

A new attribute type, Interval, represents time intervals and durations. The default value is a string " `00:00`" representing zero minutes and zero seconds. Longer durations, of over one hour are supported—see below—but the general designed/expected usage is for durations shorter than an hour. For example, the interval value

```
01:30
```

represents one minute and thirty seconds. Note that minute and second values do not need zero-padding, entering "4:2" results in a displayed value of "04:02"

When reading/copying Interval type data, it is always in String form, i.e. "22:09", even though the Interval data is stored internally (in the running app) in a different form. As a result, an Interval can be passed directly to a string attribute without needing modification. So, adding an interval to a string:

`"The answer is:" + $MyInterval` returns a *string*.

Using intervals of an hour or more

Although originally intended for short (sub 1 hour) durations, Intervals can accept hour or day inputs. For long intervals note that hours and days are accepted units: year or month scope inputs are not understood by the interval parser.

Thus the duration

```
01:01:00
```

represents a duration of one hour, one minute, and thirty seconds. The seconds cannot be omitted, even if '00' as the above written as "01:01" would be parsed as one minute and one second. Only "01:01:00" ensures the hour element is detected correctly.

Days are defined using the word 'day(s)':

`2 days 01:00:00` represents 49 hours

Duration components larger than a day (of 24 hours), e.g. months or years, are not supported, though is is possible to use large day values:

`62 days 01:30:10`

representing an interval of over two months' duration. The maximum length of an interval but it is not advised to exceed 1 year (but specified in days, i.e. 365 days) but ideally intervals are used for a few days or less. If the duration of the interval is desired in whole days only, use normal Date-type arithmetic and store the duration as a Number-type.

Formatting Interval data

Using, Interval.format() only accepts a limited range—two choices—of formatting string options, i.e. **not** the full range of Date formats. The displayed format of Intervals, in Displayed Attributes and Get Info cannot be modified. If passing the data for use elsewhere, e.g. for export, the string representation can be modified using normal String manipulation operators.

Negative values

An interval may have negative duration, "-05:30":

```
-05:30
```

represents a *negative* duration of 5 minutes and 30 seconds. This can be useful when doing short duration date arithmetic (as below).

Use in Date arithmetic

Intervals may be added or subtracted from Date attributes. They may multiplied or divided by constants or numeric attributes, and may be compared for equality using == and != or for magnitude using < and >.

Note: subtracting two dates does **not** currently return an interval; rather, it returns the number of days between the two dates in accordance with pre-existing Date-type attribute behaviour. To get the interval between two Dates, use the interval() operator. The latter does return interval type data.

If $MyInterval is "`-05:30`" and $MyDate is `24/10/2022 12:00:00` (midday on 24 October 2022):

```
$MyDate = $MyDate + $MyInterval;
```

results in $MyDate being `24/10/2022 11:54:30`. By accepting negative intervals the same action code can deal with positive and negative durations. So to find the number of (whole) minutes in the interval:

```
$MyNumberOfMinutes = minutes($MyDate,$MyDate+$MyInterval);
```

A more generalised approach to the last:

```
$MyNumberOfMinutes = minutes(date("now"),date("now")+$MyInterval);
```

Here minutes are tested but the method could test days, hours or seconds (not bigger units as Interval data us usually a few days at biggest and generally less than a whole day).

Alternative syntax

Some alternative syntax is supported, including an 'h' suffix/separator for hours and 'd' for days. Thus the duration

```
1h30
```

represents one hour and thirty minutes, whilst:

`1 day 01:00:00` represents 25 hours

`2d2h30` represents two days, two hours and 30 minutes

whilst:

```
2d5
```

is treated as 2 days, 5 hours, as hours is the next smaller duration measure than days.

The 'd' marker is always resaved/displayed as 'day(s)':

`2d5` is resaved as "2 days 05:00:00

`1d5` is resaved as "1 day 05:00:00

Note that whilst 'm' and 's' suffixes are understood for minutes and seconds; the 's' is superfluous anyway as it is always the last segment if used. If a duration is entered as "3m20" Tinderbox will re-save the value in the default format of "03:20" so use of 'm' is deprecated. Thus a duration entered as follows will be understood:

```
1h30m10s
```

but it is treated as one hour, thirty minutes and 10 seconds but will be re-saved as:

```
1:30:10
```

Note slight variations in accepted abbreviations compared to pre-existing Date-type attribute date arithmetic usage.

Use of floating point (decimal) numbers

From v9.5.0, assigning a floating point (i.e. 'decimal') number to an interval is supported, and interpreted as a number in *seconds*; the numbers after the decimal point are discarded—no rounding up/down occurs. This can be useful as time codes, e.g. as in offsets within audio or video files, may be given with a decimal element: '45.3' vs. '45'. Thus:

```
$MyInterval = 5406;
```

gives in interval of "01:30:06". Note however that

`$MyInterval = 5406.8;`

gives the *same* result of "01:30:06". The '0.8' of a second in the input value does not cause the seconds element of the interval to round up to ':07'. This latter is the equivalent of a floor() operation where a decimal number is always rounds down the the existing whole integer:

`$MyInterval = floor(5406.8);` (returns integer '5406')

Default/Empty value

The string "`00:00`" (zero hours:minutes).

Sorting order

Increasing duration, so unset ("00:00") values list first.

Coercing interval data

Interval-type System attributes

Built-in attributes of the file data type are listed below:

- $MyInterval

List-Type Attributes

List

A semi-colon delimited list of string values. In terms of stored data Sets and Lists are the same: a string containing one or more semi-colon delimited items. The difference is in the way Tinderbox handles the two data types, as lists may contain duplicate items. Although the Set-type pre-dates List-type in Tinderbox, Lists should be thought of as the underlying form and Sets as a form of List with special features (de-duplicated, always sorted, etc.) form of List.

List items can be read/set via their list order number (see below) but note that this is *zero-based*, i.e. list item #1 has address value zero (`0`).

A List, even a list of numbers or boolean values, is stored as a semi-colon delimited string, i.e. list items are not stored with an explicit data type, though multi-value value can be list or dictionaries.

Declaring a new List

If setting a List's literal values via action code use the square bracket `[ ]` List definition:

`$MyList=[Frogs;Dogs;Logs];`

Above, the brackets *replace* the older *now-deprecated* method of enclosing quotes:

`$MyList="Frogs;Dogs;Logs";` NOTE: do not use this for new code!

Whilst both methods work and the latter, older, method will be found in many demos and tutorials, the bracketed notation is now the preferred method.

Importantly, when typing/pasting a List's values into a UI input box such as the Inspector, Get Info, or Displayed Attributes, the enclosing brackets or quotes (as used above) should be omitted, i.e. using `planes;trains;automobiles` no `[planes;trains;automobiles]`. If brackets are used by mistake, the Tinderbox parser should ignore an outmost pair (as it it would quotes) abut still honour brackets within the overall list value as implying a nested list. If Tinderbox reads from a List via code and reports (logs) a value with enclosing brackets, Tinderbox knows—in code—to ignore those as simply being list delimiters.

Adding new list items

With a List you can add/remove individual or multiple values and test its contents. In actions, `+` adds an item to a set if it is not already present. Values should be enclosed in square brackets (previous, and still supported is to use enclosing double quotes). If $PetTypes' value is "cats;dogs", then:

`$PetTypes=$PetTypes+[rabbits];` adds the new value

`$PetTypes=$PetTypes+[rabbits];` unlike a Set, this adds a second instance of "dogs" to the *end* of the List.

The `+=` increment operator can also be used. If the List is 'cats;dogs;frogs':

`$PetTypes=$PetTypes+[cats;frogs];` this gives a value of 'cats;dogs;frogs;cats;frogs'; the new values appended to the list in the order supplied.

To add new—not replacement—item(s) to the beginning of a list, use the `[]` list declaration method:

`$PetTypes= [rabbits;hamsters]+$PetTypes;`

N.B. *the last fails if using to old quote-enclosed format* instead of square brackets.

To add a new item that is itself a list, note the extra set of brackets needed. The following adds a new list item whose value is a two-item list.

`$PetTypes=$PetTypes+[[rabbits;bunnies]];`

To insert a new list at a specified location, i.e. adding a list item as opposed to replacing an existing value, iterate the list using List.each() with a custom counter. Store existing items in a new list and add the term when the loop reaches the desired place in the list. At the end of the loop write the new list back over the original List.

Deleting items

In actions using a `-` (minus) removes the supplied value(s) if present. Importantly, this removes *all* occurrences if the deleted item;

`$PetTypes=$PetTypes-[dogs];` leaves only "cats" as a value.

To delete only specific instances would require using List.each() with a custom counter.

The `+=` increment operator can also be used. If the List is 'cats;dogs;frogs':

`$PetTypes=$PetTypes+[cats;frogs];` this gives a value of 'cats;dogs;frogs;cats;frogs'; the new values appended to the list in the order supplied.

The `-=` decrement operator can also be used. If the Set is 'cats;dogs;frogs':

`$PetTypes=$PetTypes-[cats;frogs];` this leaves only 'dogs' as a value.

Replacing a list item's value

The List[N] notation addresses any single List item using a *zero-based* address. Thus `List[1]` addresses item #2, whilst `List[0]` addresses the first. To replace item #3 in a list (assuming a big enough list):

`$MyList=[Frogs;Dogs;Logs];`

`$MyList[2]=[Pogs];` results in a list of "Frogs;Dogs;Pogs"

Additional list definition mark-up

From v9.5.0, Lists and Sets may now be written by enclosing them in square brackets. Lists may be nested; for example, the list

`[ 1; [2;3]; 4]`

contains a 3 elements — 1, the nested list 2;3, and 4.

Long Tinderbox precedent holds that list addition adds each element of the two lists. For example

`$MyList = [1] + [2;3]`

results in the list `1;2;3` and **not** `1;[2;3]`. To add a sublist to a list, use the operator List/Set.extend(). Thus:

`$MyList = [1].extend( [2;3])`

results in the list `[1;[2;3]]`.

From v9.6.0, offset assignment to a list of notes recognises bracket-enclosed lists correctly, for example:

`$Status([this;parent])="urgent";`

From v9.6.0, implicit evaluation is no longer performed in bracketed lists. For example,

`$DisplayedAttributes=[MyList;MyString];`

is now equivalent to

`$DisplayedAttributes="MyList;MyString";`

If new List terms need evaluation, use list() instead.

Accessing nested lists

Consider the following:

`$MyTestList = [1;[a;b];3];`

To retrieve the nested list:

`$MyString = $MyTestList[1];` results in '[a;b]'

Note the square brackets are retrieved too, so for further use of the list this might be better:

`$MyString = $MyTestList[1].substr(1,-1);` results in '[a;b]'

To retrieve a value from that list:

`$MyString = $MyTestList[1][1];` results in 'b'

De-duplicating a List: List vs. Sets

Lists, unlike Sets, allow duplicate values. To de-dupe a List, use the .unique dot-operator:

`$MyList=$MyList.unique;`

An older alternate method, which may be found in old demos is simply put its contents into a Set-type attribute:

`$MySet=$MyList;` (deprecated in favour of the method above)

If $MySet and $MyList both have the value [cats;dogs]: the following have different outcomes:

`$MySet=$MySet + [dogs];` gives 'cats;dogs'

`$MyList=$MyList + [dogs];` gives 'cats;dogs;dogs'

The Set attribute does not add the duplicate value but the List attribute does. List data values are stored in the order added.

Testing (querying) Sets & Lists

To test a set or list, use the .contains() operator, syntax `AttributeName.contains("tested_value")`, returns `true` if any Set/List discrete value exactly matches the designated tested_value; if case sensitivity is irrelevant for the query use .icontains(). If a user attribute `$PetTypes` has a value of 'dogs;cats' then

`$PetTypes.contains("dogs")` is `true`,

but

`$PetTypes.contains("dog")` is `false`

This is because Let/List matching does not allow partial matches, as via regex, unlike with String-type data.

Other variants:

```
$PetTypes.contains("Dogs").lowercase is true
$PetTypes.icontains("DOGS") is true
```

It can be useful to use a stored value as the search term, for instance using the name of an agent as the search term:

```
$PetTypes.contains($MyString) is true
```

Escaping literal semi-colons

If a list item must contain a semi-colon, it must be escaped, using a backslash, '\;'. Once the backslash is entered, it disappears and the list item containing the semi-colon is enclosed in double-quotes. Do not try to escape a value by adding the quotes directly, use the backslash method. Action code methods to make lists will treat a '\;' in an input string as an escape and act accordingly. Consider using String.replace() as a method for escaping backslashes (though only where intended!).

Listing and Exporting sets

The format() action operator and more recent .format() dot operator offer ways to turn sets into HTML lists for export. See Exporting Set-type data for more.

System Attributes: Sets vs. Lists

Most group-scope operators can work with lists or sets, as well as the find() operator (whose own output is a list) and literal list-based group designators; exceptions include $DisplayedAttributes where duplicates would not be helpful. It is the declared data type of the attribute being collected that informs the operator to return a list or set.

Default/Empty value

An empty string.

Sorting order

Lists are not sorted, so retain the order in which items were passed into the list, most recent being at the end. Lists can be sorted using action code sort operators or by setting the sort attributes of containers.

List-type System Attributes

Built-in attributes of the List data type are listed below:

- $Aliases
- $Authors
- $DisplayedAttributes
- $Flags
- $GridLabels
- $MyList
- $PlotColorList
- $PosterLabels
- $PosterX
- $PosterY
- $Sentiments
- $TimelineBandLabels

Number-Type Attributes

Number

A numerical value. If no default value is specified, a Number type defaults to the value 0 (zero). User attributes of Number type may be configured, at first creation, as 'sequential'.

A sequential number attribute is like a database counter field, where the next unused number is issued and values no longer used are not re-issued; i.e. the next number is always higher than the last. Sequential numbers start at 1. When a new sequential attribute is added to an existing document, the already-created notes are initialised with sequential values of the new attribute, starting with the the first notes in $OutlineOrder sequence. Thereafter notes are numbered in the order added, regardless of $OutlineOrder.

Very large or small numbers may be displayed (and set) in exponential notation:

```
0.000001 = 1e-06
$MyNumber = 1.2e+3 sets '1200'
```

When very large or small number strings are typed into a Displayed Attributes box the result may be '0', in which case use exponential notation.

The Displayed Attributes display will also truncate the number of decimal places shown. The display limits seem to be:

- Whole (non decimal) numbers will display up to 9 digits, i.e. between between -999999999 and 999999999.
- Numbers over 1, positive or negative, will display 6 digits e.g. 150.123 or 1.12346.
- Numbers over 1, positive or negative, will display 6 decimal places e.g. 0.012345.

When typing/pasting in number values to a KA box the input value is stored and not the rounded/truncated visible value. To see the decimal places in full coerce the value to a string, e.g. set a string attribute to the value of the number attribute. Beware that values copied from a Displayed Attributes display box use the visible value and not the real underlying value, so passing the number to a string and reading the latter is useful if data fidelity is required.

Default/Empty value

The value `0` (zero).

Sorting order

Ascending value in numeric sort order; default (0) thus lists first.

Number-type System Attributes

Built-in attributes of the number data type are listed below:

- $AdornmentCount
- $AgentPriority
- $BadgeSize
- $Base
- $Bend
- $Border
- $BorderDash
- $CaptionOpacity
- $CaptionSize
- $ChildCount
- $DescendantCount
- $DisplayedAttributesFontSize
- $EstimatedNoteSize
- $FillOffsetY
- $FillOpacity
- $GridColumns
- $GridLabelSize
- $GridOpacity
- $GridRows
- $Height
- $HoverOpacity
- $HTMLFileNameMaxLength
- $HTMLFontSize
- $ID
- $ImageCount
- $ImageSizeLimit
- $InboundLinkCount
- $InteriorScale
- $IrisAngle
- $IrisRadius
- $KeyAttributeFontSize
- $Latitude
- $LeafBase
- $LeafBend
- $LeafTip
- $LineSpacing
- $Longitude
- $MapBackgroundFillOpacity
- $MapBodyTextSize
- $MapNameSize
- $MapScrollX
- $MapScrollY
- $MapTextSize
- $mt_allow_comments
- $mt_allow_pings
- $MyNumber
- $NameLeading
- $Opacity
- $OutboundLinkCount
- $OutlineDepth
- $OutlineNameSize
- $OutlineOrder
- $OutlineTextSize
- $PlainLinkCount
- $PlotBackgroundOpacity
- $PosterZoom
- $Range
- $ReadCount
- $RSSItemLimit
- $ScreenHeight
- $ScreenWidth
- $ScrivenerLabelID
- $ScrivenerStatusID
- $SelectionCount
- $Sentiment
- $ShadowBlur
- $ShadowDistance
- $SiblingOrder
- $SimplenoteSync
- $SimplenoteVersion
- $SubtitleOpacity
- $SubtitleSize
- $TextFontSize
- $TextLength
- $TextLinkCount
- $TextPaneRatio
- $TextPaneWidth
- $TextWindowHeight
- $TextWindowWidth
- $TimelineBand
- $TimelineBandLabelOpacity
- $Tip
- $TitleHeight
- $TitleOpacity
- $Visits
- $WebLinkCount
- $Width
- $WordCount
- $Xpos
- $Ypos

Set-Type Attributes

Set

A Set date type is a special type of string, within which discrete values are delimited by semicolons. When defining a Set in code, use the conventions are described for the List data type.

In terms of stored data the Set and List list are the same: a string containing one or more semi-colon delimited items. The difference is in the way Tinderbox handles the two data types, as lists may contain duplicate items. Although the Set-type pre-dates List-type in Tinderbox, Lists should be thought of as the underlying form and Sets as a refined (de-duplicated) form of List.

A Set is useful for lists of topics, categories, and tags where duplication of listed items is *not* wanted:

```
[astronomy;marine biology;chemistry]
[dogs;cats]
[3;9;15]
```

Note that the Set is always a string even if the values happen to be numbers: item values in a Set do not have explicit data types. A Set can have a single value, e.g. the third example above. With a single value there is no need to add a final semicolon and the same holds for the last of multiple values. Tinderbox will not mind if you supply a semicolon there (or after the last of multiple values), it will just strip it off as it processes the data. The default Set value is an empty string. A Set-type differs from a List-type in that duplicate values are not allowed, and the list of values is always A–Z lexically sorted.

To apply values directly to a Set-type attribute via the Displayed Attributes or via Info view or via the Inspector's Quickstamp, simply type the values as seen into the data box. In all these methods, you do not need to add enclosing quotes, bu each discrete value should be separated with a semi-colon.

Sets and auto-sorting

From v9, Set attributes were reimplemented to improve performance with large sets, which makes a Set more aggressive in asserting its control of the sequence of their elements. Compared to previous use, the stored order of items in the Set's value list are much more likely to change from the order as originally entered into case-sensitive lexical sort order. This may catch out long term users being used to Sets generally retaining their as-created item order. It the latter is needed, it may make more sense to use a List type instead.

Declaring a new Set

If setting a Set's literal values via action code use the square bracket `[ ]` List/Set definition:

```
$MySet=[Frogs;Dogs;Logs];
```

Above, the brackets *replace* the older *now-deprecated* method of enclosing quotes:

```
$MySet="Frogs;Dogs;Logs"; NOTE: do not use this for new code!
```

Whilst both methods work and the latter, older, method will be found in many demos and tutorials, the bracketed notation is now the preferred method.

Importantly, when typing/pasting a Set's values into a UI input box such as the Inspector, Get Info, or Displayed Attributes, the enclosing brackets or quotes (as used above) should be omitted, i.e. using `planes;trains;automobiles` no `[planes;trains;automobiles]`. If brackets are used by mistake, the Tinderbox parser should ignore an outmost pair (as it it would quotes) abut still honour brackets within the overall list value as implying a nested list. If Tinderbox reads from a Set via code and reports (logs) a value with enclosing brackets, Tinderbox knows—in code—to ignore those as simply being list delimiters.

Adding values

With a Set you can add/remove individual or multiple values and test its contents. In actions, `+` adds an item to a set if it is not already present, and `--` removes it if it is present. Values must be enclosed in double quotes. If $PetTypes' value "cats;dogs"

```
$PetTypes=$PetTypes+[dogs]
```

leaves $PetTypes unchanged, since 'dogs' is already in $PetTypes

The `+=` increment operator can also be used. If the Set is 'cats;dogs;frogs':

```
$PetTypes=$PetTypes+[owls;dogs];
```

the Set is 'cats;dogs;frogs;owls'.

Deleting values

In actions using a `-` (minus) removes the supplied value(s) if present. Importantly, this removes *all* occurrences if the deleted item;

```
$PetTypes=$PetTypes-[dogs];
```

leaves only 'cats' as a value.

The `-=` decrement operator can also be used. If the Set is 'cats;dogs;frogs':

```
$PetTypes=$PetTypes-[cats;frogs];
```

this leaves only 'dogs' as a value.

Testing (querying) Sets & Lists

To test a Set or List, use the .contains() operator, syntax `AttributeName.contains("tested_value")`, returns `true` if any set/list discrete value exactly matches the designated tested_value; if case sensitivity is irrelevant for the query use .icontains(). If a user attribute `$PetTypes` has a value of 'dogs;cats' then

```
$PetTypes.contains("dogs");
```

is `true`,

but

```
$PetTypes.contains("dog");
```

is `false`

This is because Set/List matching does not allow partial matches, as via regex, unlike with String-type data.

Other variants:

```
$PetTypes.contains("Dogs").lowercase()
```

is `true`

```
$PetTypes.icontains("DOGS")
```

is `true`

It can be useful to use a stored value as the search term, for instance using the name of an agent as the search term:

```
$PetTypes.contains($MyString)
```

is `true`

Listing and exporting Sets

The format() action operator, and newer .format() dot operator offer ways to turn sets into HTML lists for export. See Exporting Set-type data for more.

Set data vs. List data

List-type attributes came to Tinderbox after Sets. Lists, unlike Sets, allow duplicate values and Sets can better be thought of as de-duped versions of Lists, i.e. lists with no duplicate entries.

To de-dupe a List, simply put its contents into a Set-type attribute:

```
$MySet=$MyList;
```

Escaping literal semi-colons

If a list item must contain a semi-colon, it must be escaped, using a backslash, '\;'. Once the backslash is entered, it disappears and the list item containing the semi-colon is enclosed in double-quotes. Do not try to escape a value by adding the quotes directly, use the backslash method. Action code methods to make lists will treat a '\;' in an input string as an escape and act accordingly. Consider using String.replace() as a method for escaping backslashes (though only where intended!).

Default/Empty value

An empty string.

Sorting order

As for a String Data Type, using the literal string values of the Set in case-sensitive lexical sort order. For example, the 5-item of values 'Ant', 'ant', 'bee', 'Cow', 'cow' when passed to a set would store as "`Ant;Cow;ant;bee;cow`".

Set-type System Attributes

Built-in attributes of the set data type are listed below:

- $Associates
- $ClusterTerms
- $Deck
- $KeyAttributes
- $LocalAttributes
- $MySet
- $NLNames
- $NLOrganizations
- $NLPlaces
- $NLTags
- $Participants
- $RefKeywords
- $ScrivenerKeywords
- $SimplenoteTags
- $Tags
- $Tot

String-Type Attributes

String

A sequence of characters, most often as words and sentences. If no default value is specified, a String type defaults to an empty string.

Strings are not delimited (i.e. quotes are not required).

The set and action data types effectively special forms of the string data type.

Default/Empty value

An empty string.

Sorting order

Ascending order using literal string value in lexical sort order.

String-type System Attributes

Built-in attributes of the string data type are listed below:

- $Abstract
- $AccessDate
- $Address
- $AgentQuery

- $ArticleTitle
- $Author2
- $Author3
- $Author4
- $Badge
- $BookTitle
- $BorderBevel
- $CallNumber
- $Caption
- $CaptionAlignment
- $ChosenWord
- $City
- $CleanupAction
- $Container
- $Country
- $CreatedFrom
- $Creator
- $DEVONthinkGroup
- $DEVONthinkLabel
- $DisplayedAttributesDateFormat
- $DisplayName
- $District
- $DOI
- $DominantLanguage
- $Edition
- $Email
- $EmailSubject
- $EmailTemplate
- $EvernoteNotebook
- $FormattedAddress
- $FullName
- $GeocodedAddress
- $HTMLBoldEnd
- $HTMLBoldStart
- $HTMLCloud1End
- $HTMLCloud1Start
- $HTMLCloud2End
- $HTMLCloud2Start
- $HTMLCloud3End
- $HTMLCloud3Start
- $HTMLCloud4End
- $HTMLCloud4Start
- $HTMLCloud5End
- $HTMLCloud5Start
- $HTMLCodeEnd
- $HTMLCodeStart
- $HTMLExportAfter
- $HTMLExportBefore
- $HTMLExportCommand
- $HTMLExportExtension
- $HTMLExportFileName
- $HTMLExportFileNameSpacer
- $HTMLExportPath
- $HTMLExportTemplate
- $HTMLFirstParagraphEnd
- $HTMLFirstParagraphStart
- $HTMLImageEnd
- $HTMLImageStart
- $HTMLIndentedParagraphEnd
- $HTMLIndentedParagraphStart
- $HTMLItalicEnd
- $HTMLItalicStart
- $HTMLLinkExtension
- $HTMLListEnd
- $HTMLListItemEnd
- $HTMLListItemStart
- $HTMLListStart
- $HTMLOrderedListEnd
- $HTMLOrderedListItemEnd
- $HTMLOrderedListItemStart
- $HTMLOrderedListStart
- $HTMLParagraphEnd
- $HTMLParagraphStart
- $HTMLPreviewCommand
- $HTMLStrikeEnd
- $HTMLStrikeStart
- $HTMLSubscriptEnd
- $HTMLSubscriptStart
- $HTMLSuperscriptEnd
- $HTMLSuperscriptStart
- $HTMLUnderlineEnd
- $HTMLUnderlineStart
- $IDString
- $ISBN
- $Issue
- $Journal
- $KeyAttributeDateFormat
- $LeftMargin
- $MapBackgroundFill
- $MapBackgroundPattern
- $mt_convert_breaks
- $mt_keywords
- $MyString
- $Name
- $NameAlignment

- $NotesFolder
- $NotesID
- $NoteURL
- $Organization
- $Pages
- $ParagraphSpacing
- $Path
- $Pattern
- $PostalCode
- $PosterCSS
- $PosterSettings
- $PosterTemplate
- $Prototype
- $PublicationCity
- $PublicationYear
- $Publisher
- $RawData
- $ReferenceRIS
- $ReferenceTitle
- $RefFormat
- $RefType
- $RightMargin
- $Role
- $ScrivenerID
- $ScrivenerLabel
- $ScrivenerNote
- $ScrivenerStatus
- $ScrivenerType
- $Shape
- $SimplenoteKey
- $Sort
- $SortAlso
- $SortAlsoTransform
- $SortTransform
- $State
- $Subtitle
- $SyntaxHighlighting
- $TableHeading
- $Tabs
- $Telephone
- $Text
- $TextAlign
- $TextExportTemplate
- $TimelineEndAttribute
- $TimelineStartAttribute
- $Twitter
- $UUID
- $Volume
- $WeblogPostID

URL-Type Attributes

URL

A URL string. If no default value is specified, a URL type defaults to an empty string. URLs are not delimited (i.e. quotes are not required).

URL attributes behave like string attributes. When displayed in a note's displayed attributes table, URL-type attributes show a globe icon before the attribute value. Pressing the icon opens that attribute's stored URL (if any) in the default web browser.

Links dropped on a URL-type Displayed Attribute populate that attribute; links dropped on any other data type Displayed Attributes populate the system $URL.

If using a local path to a file, e.g. to enable AutoFetch use, you must use the file:// protocol rather than a bare file path.

Default/Empty value

An empty string.

Sorting order

As for String Data Type.

URL-type System Attributes

Built-in attributes of the URL data type are listed below:

- $PosterURL
- $ReferenceURL
- $SourceURL
- $URL

Determining the data type of an attribute

All attributes in Tinderbox, System or User, are one of a number of different Data Types (String, Boolean, etc.). The type of an attribute is important in Tinderbox action code as it has some affect on how you may use the attribute and the under-the-hood assumptions Tinderbox makes when using it. How then to tell the data type for any given attribute?

User or System? If uncertain, look at the User Attributes list first, as there are usually only a few of these. If the attribute is a User one, click in the left column and look at (but do not change!) the 'type' pop-up at the top of the right side of the dialog.

If the attribute is not on the User list, it must be a System attribute. The quickest way to then find its data type is to use aTbRef's System Attribute listing, which lists all system attributes in alphabetical order.

If you do not have access to aTbRef, use the Document Inspector's System tab listing to find the attribute. Once the attribute is selected, the data type is shown on the tab.

Default values for attribute Data Types

If no value has been defined via a preference or otherwise inherited or manually set, the following are the value of 'no value' for each attribute data type.

- Boolean: `false`
- Date: "`never`" (a string)
- Number: `0`
- Interval: "`00:00`" (a string representing zero hours:minutes)
- String & String-type: "" (an empty string, no data).
- List†: `[]`.
- Set†: `[]`.
- Dictionary†: `{}`.

All the remaining Tinderbox data types are essentially special forms of a String-based data type and so use the same default value, of an empty string:

- Action.
- Color.
- File.
- Font.
- URL.

†. **Multi-value types**. Only 3 types, *out of **all** the data types above*, allow the storage of multiple values (i.e. 'lists' of discrete values), two of which are themselves specialised versions of the first:

- List. A list of values.
- Set. A list where any case-sensitive duplicate values are automatically removed (so strings 'Cat' and 'cat' are still treated discrete values). Sets also use a lexical auto-sort so cannot be relied upon to retain user-set listing order (e.g. such as by setting $Sort).
- Dictionary. Holds a list of 'key:value' pair values. Keys must be unique, as this is how the list data is queried/altered (i.e. asked for or edited). Currently, key values that are themselves lists are allowed but cannot be accessed via action code. Again sorting the list is of little value as data is accessed via the key for any discrete 'key:value' list value.

Set vs. List

A historical note for long-time users. Originally, the Set data type was the only multi-value data type, with List and Dictionary being added later on. The original Set only auto-sorted occasionally, if at all. Since changes in v8 Sets now are always auto-sorted, for performance reasons. That change may catch out long-term users of Tinderbox as previously Sets re-sorted so rarely that the made sense as de-duplicated Lists. Now that affordance has gone, use List type to preserv order and List.unique if de-duplication is needed.

However, the List data type is best understood as the basic (multi-value) listing type—it records what is entered in the order entered and does not alter it.

Document versus Application: system attributes and defaults

When Tinderbox generates a new TBX document, all current system attributes are added to the document. Each such attributes also stores the (then current) default for that attribute.

But, over time, new versions of Tinderbox are released. Occasionally, an existing system attribute may change its default. An example is the default font used in text ($TextFont). Separately, new system attribute may appear. Less often, old system attributes are no longer added to new files.

So, what happens to existing TBX files in these situations?

A new default value for an existing system attribute

This has no effect on existing documents as they retain their originally defined default value. By comparison, new documents will define the same attribute use the newer, now current default. If the user wishes to use the newer value, then they edit the attribute's default—see the 'review actions' section below. This avoids nasty surprises were the new default to be auto adopted and where the existing document behaviour assumed the old default value, such as might affect automation within the document.

A new app version gains a new system attribute

On first use in the new version of the app, the new system attribute(s) will be added to the documents existing system attribute, using the current default values.

An existing system attribute is dropped from the schema in a new app version

No change occurs—existing apps retain their pre-existing system attributes, even if some are now moribund. There is no way for the user to delete these now obsolete or un-needed attributes but they should simply be ignored. Any *new* documents will lack these attributes completely. A point to bear in mind is that occasionally a new attribute replaces an old one, but both are supported. Thus $AccentColor replaced $Color2. Whilst, newer TBXs will lack $Color2, older documents will likely have both and both will store the same values. This ensures old document's action code does not just break. Ideally, if such a 'replacement' occurs, users should review code and replace any use of to the old attribute name with the new name.

Review actions for the user to consider

If any of the above happen:

- Consider if a new default value for a system attribute is appropriate for existing document(s). For those documents where change is appropriate, manually change the default value in the System Inspector or, for attributes whose value i set via Document Settings, in the appropriate tab of Document Settings.
- Where new attributes are added, deprecating older ones, review documents for use of the older attribute. The two key places to check:
  - Displayed Attributes. Those using prototypes will find this easier than where Displayed Attributes are set per note. Do not forget, a note's list of Displayed Attributes is stored in $DisplayedAttributes. Thus, using $Color2 as an example a query like `hasLocalValue("DisplayedAttributes")&$DisplayedAttributes.contains("Color2")` will find all notes that have Displayed Attributes set and which not inherited (e.g. not using a prototype) AND where the Displayed Attributes table includes the 'Color2' attribute.
  - Action code. For attribute based code such as rules, edicts, OnAdd etc. ,the same sort of query as above can be used, e.g. for edicts: `hasLocalValue("Edict")&$DisplayedAttributes.contains("Edict")`. For stamps and functions, add the built-in Hints (if not already in the document) and test the $Text of Stamp and Library notes.

Renaming an attribute

Attribute names are case-sensitive. Unlike previous versions, User [sic] attributes can be renamed and re-data-typed after creation. System attributes can not be renamed or have their data type changed. A rename might be to use a completely different term, e.g. $Cost renamed to $Price, or it might be a change of case, $cost to $Cost.

Simply select the attribute in the Document Inspector's User tab, select the name, enter a new value and press Return. Attribute listings and Displayed Attributes entries will update. However, any explicit use of the old name in action or expor code, templates, boilerplate code, etc., will be unaffected. Such mentions must be manually edited to reflect the change but do not overlook agents as a help to find such code passages for correction. Places where you may need to manually change attribute names:

- Action-type system attributes.
- Action code in code notes.
- Stamps.

* Export templates, e.g. in or value()calls. **Changing attribute name or data type & existing values** Do not assume renaming an attribute will retain the values used under the old name. Although values may persist in *some* cases, do not rename attribute based on that premise. If the attribute you wish to rename already has values in some notes, do not rename but rather make a new attribute, transfer the values and then delete the old attribute. For example, an existing Number-type user attribute $Price needs to be renamed to $TotalPrice but $Price already has values. The process advised is as follows (it only looks long as it is given in a lot of detail) it is actually very quick to do: * Work on a copy of the data: make a copy of the TBX before starting this process * Make a new Number-type user attribute 'TotalPrice' * Disable any rules/agents that might be alter values of $Price while making this attribute change. * Create an agent with the quer `hasLocalValue("Price")`. This will match all notes that have a (local, not inherited) value set for the existing Price attribute. If you have a prototype that sets a value inherited b other notes, only the prototype will match as well as notes tha have a value set deliberately in that note. * Check the agent matches to sort of notes expected: this is good to do *before* diving straight in to copying any values. If all looks good… * Set an agent action `$TotalPrice = $Price`. For each matched note—and only matched notes—the value of the $TotalPrice attribute is set to the notes existing $Price value. * At this point *both* attributes in matching notes have a value, and the same one. This is another opportunity to check things look good if trying this for the first time (Outline column view can help). Assuming all looks good… * Delete the agent. Do not forget this step! * Open the Document Inspector's User tab, and select the 'Price' attribute. Using the gear-wheel button (at right) from the pop-up choose the 'Delete user attribute' option. The 'Price' attribute is deleted along with any data. * Other tidy-up issues: ** Displayed Attributes. If $Price was used as a Displayed Attribute you will need to us an agent to find and remove the old name and add the new one. ** Action code. Review code in Stamps and any Action-type attributes (e.g. $Rule, $AgentAction, etc.) to find and remove the old name and add the new one. **Changing data type** If changing the data type of an attribute, consider coercion effects: * To Boolean. No value: `false`. All other values: `true`. * String-based to Number. Strings of number characters: numbers. All other strings: no value. * Number to string based. Numbers become literal strings. * To/from Date. Date to String will likely give the date in string form as would be shown in Displayed Attributes in the current TBX. Other type conversions may have variable effects, e.g. Date to Number which is not a sensible change, anyway. * List/Set to String. A semi-colon concatenated string of list values. * String to List/Set. The current string will split to list values at any semi-colons in the original value. * To/From Interval. Not documented. Bear in mind that Colour, URL and File types are essentially String type with a special form of handling

Keywords for Date-Type Attributes

Note: both 'keyword' and 'designator' have long been used interchangeably for the terms described below. Other types of designator are described here.

When defining Date-type data Tinderbox's code parser accepts the following expressions as placeholders for calculated dates:

- **yesterday**
- **today**
- **now**
- **tomorrow**

Duration keywords

All provide a date/time, to seconds-level granularity. **today** and **now** return exactly the same value, with **yesterday** being 24 hours previous to **now** and **tomorrow** 24 hrs later. These alternate keywords can make for more intuitive contextua use in code.

The following designators can be used to *modify* the above. However, if simply used on their own, rather than as a modifier they do not return a valid date/time:

- **day(s)**
- **week(s)**
- **month(s)**
- **year(s)**
- **hour(s)**
- **minute(s)**
- **second(s)**

No value keyword

There is a further special case:

- **never**. No date, and regarded as earlier or later than any real date. This is also the default/empty value for Date-type attributes.

Case-sensitivity of keywords

Treat usage for the above placeholders as case sensitive.

Day-of-the-week keywords

Day placeholders, such as **Sunday**, **Monday**, etc., may also be used. Tinderbox recognises the day of the week and interprets it as the day *after* today with that week day. Thus on Sunday June 1, the date "Sunday" refers to Sunday, June 8 These keywords are locale-observant.

- The date parser recognises the day of the week in the current (macOS) locale. Recognised forms include the full day (Sunday), the short day (Sun), and the very short day (S). Note though, that the very short day is ambiguous in many languages, including English.

In other respects, days of the week act like the core keywords, e.g. **today**, in that they can be modified similarly, e.g. "Monday - 1" week is the previous Monday.

Dates may be modified by adding and subtracting "minutes", 'hours", "days", "weeks" and "years":

```
date("today+65 minutes")
```

Explicit (24-hour clock) time can be used, in which case it comes at the end of the string:

```
date("today+7 days 00:00")
```

The latter is useful to ensure that a base like 'today' or 'now' that take system clock time are set to to an exact known time, as may be needed for accurate date comparison.

Normally time is input is the form consistent with the users short date(/time) form such as "09:30" or "9:30 AM"but more flexibility is offered for import making it easier to set times as part of date/time designators, such as:

- `"today 0930"`
- `"today 930pm"`
- `"today 11"`

The first two before half-past nine in the morning, The latter will be interpreted as 11 AM (11:00): the lack of minutes implies to set them as 00.

Note that the date(string) operator arguments are written as quoted strings which may include addition/subtraction signs. This format is deprecated:

```
date("today" +" 65 minutes") INCORRECT!
```

for this form, noting all string inputs are in one string:

```
date("today + 65 minutes")
```

If using Attribute values in a date expression simple use the $-prefixed attribute name outside the quoted string:

```
date($Modified+"5 minutes")
```

If modifying a date and the expression part of code can not be parsed as a date modified, the unmodified date is returned; this avoids unintentional creation of undefined ('never') dates.

Date parsing: the date "tomorrow 8" is treated as 8 o'clock tomorrow. Note in that older versions it was interpreted as 8 days from tomorrow. "Tomorrow+8" continues to denote 8 days from tomorrow.

Beware that any Date-type attributes initialised using using year, month & day *without* a time element will use current system time as opposed to a 00:00:00 which might otherwise be assumed.

What are Displayed Attributes?

The old term 'Key Attributes' was replaced by the more descriptive 'Displayed Attributes'

Any note or agent may optionally display a table of user selected attributes. The point is to enable the user to view those attributes' values in the note text pane instead of having to open the Get Info pop-over or use the Properties Inspector Quickstamp method.

Importantly, there is nothing special about these attributes compared to other attributes: new user often assume these are the only available attribute—wrong!.

A note's Displayed Attributes, i.e. the contents of its Displayed Attributes table, is simply a *user-chosen* list of attributes considered worthy of display for a given note. Indeed, this listing is stored, and inherited as the value for attribute $DisplayedAttributes.

Reasons for using Displayed Attributes so can include:

- Making it easy to see or edit certain attributes.
- Using the Displayed Attributes table as a form of dashboard.
- In demos and tutorials, it can help the learner view and edit pertinent attributes without having to use other (unfamiliar) parts of the program to do so.

As new users often first come across Displayed Attributes or reference to the feature, this can cause confusion. To be clear: adding an attribute to the Displayed Attributes table *has no effect on the displayed attribute(s)' value or upon those attribute(s)' inheritance*. However, such editing of the Displayed Attributes table *does break inheritance of $DisplayedAttributes itself*. Therefore a 'displayed' attribute is merely more easily viewed/edited in the context of the selected note.

Pre-populating Displayed Attributes pop-up lists

When string-based attributes (String, List, Set) are shown as Displayed Attributes, once discrete values are added to the attribute these are all shown via a pop-up list via the triangular icon at the right end of the attribute value edit box. For fast entry, autocompletion using matched list items is also possible. However, do bear in mind limitations on the length of and number of entries in these lists.

Or set Suggested value lists. These can be set manually via the Inspector for user attributes (and *some* system attributes) or by using action code. consider an attribute $MyStatus that will hold the status of bids. To preset 3 suggested value 'won', 'pending' and lost':

```
attribute("MyStatus")[suggested]=[won;pending;lost];
```

But, what if that list needed to change is a new default status was added, e.g. a 'cancelled'. The above list could be stored in a list attribute that could be edited as a Displayed Attributes avoiding having to set the value is code. Another way can be used is there is a note for each potential value. Thus a container 'Bid States' might hold child notes 'won', 'pending', 'lost'. This code can be used to set the $MyStatus suggested values from the children of the 'Bid States' container:

```
attribute("Bid States")[suggested]=collect(children(/Status),$Name);
```

If that code is run as a rule or edict, whenever a new note is added to, deleted from or renamed in that container, the suggested values will get updated.

Attribute inheritance of preferences and settings

Settings controlling the look or behaviour can be set at various levels. Consider something controlled at note level via an attribute. Working back up the inheritance chain, this is how the value may be derived, if not explicitly set at a given level The lowest level (and the higher in the list below) at which a value is set dictates the value used at note level:

- within the TBX

- Attribute value for this note
- (Attribute) value inherited from a prototype
- (Attribute) value inherited from a doc-level user-set default.
- Value from Document Settings (i.e. this TBX only)

- at the TBX creation

- Value from Tinderbox built-in document defaults

The latter can, in some cases, be also be modified by the user editing configuration files, or at the extreme by modifying the app package's configuration files though this is not suggested!

Intrinsic attributes

Attributes that are intrinsic to all objects and thus not inherited, or shared, by aliases.

While most attributes are inherited from prototypes, a few attributes are intrinsic, i.e. their values apply to specific notes and are not inherited via prototypes or aliases. For example, $Xpos and $Ypos are not inherited (because moving a prototype should not move notes that happen to use the prototype!). $Width and $Height are also not inherited, but note changes allowing one-time setting of height and width on first applying a prototype. If they were, then resizing a prototype might in turn resize notes that inherit from that prototype—perhaps covering other notes or disrupting a complex map. A separate case is an attribute like $RuleDisabled for prototypes, for controlling scope of action code execution as explained under disabling action inheritance.

There are few examples:

- $Xpos
- $Ypos
- $Height
- $Width
- $Container
- $ID
- $OutlineOrder
- $OutlineDepth
- $SiblingOrder
- $DisplayExpressionDisabled
- $RuleDisabled
- $TimelineBand
- $Searchable
- $EdictDisabled
- $Flags
- $InboundLinkCount
- $OutboundLinkCount

See a full listing of intrinsic attributes: be aware that many read-only system attributes are also intrinsic. Whether a note is intrinsic or not is also noted on the Document Inspector's System tab. It can also be verified in the TBX document's XML.

User attributes are never intrinsic and cannot be made so.

Aliases can show a discrete $DisplayName but it can only be set via conditional Display Expression code. Simply using the Text Inspector's Title sub-tab's $DisplayExpression input box with non-conditional code always sets both the original's and all aliases' $DisplayName to the same value.

For an alias to use a different $DisplayName from its original or other aliases (in different containers), the conditional code needs to identify the context of the note in scope. This is most easily done by looking at an attribute of the note's par as this is always discrete to the individual original or alias. Using data stored in the parent's attribute(s) is also a neat way for the conditional code to insert different values in different branches of the expression's code.

Intrinsic attributes are also significant if setting attribute values of a note via an alias, such as in an agent's action. If it is necessary to use values from the original note and not the alias, then code should use ' original' designator. Thus, in an agent action:

```
$Xpos = 4;
```

…sets the $Xpos of the alias currently being processed. To make the action set the attribute in the original of the alias, use:

```
$Xpos(original) = 4;
```

You might want to set both the alias and original values, in which case, this also works:

```
$Xpos = 4;$Xpos(original) = $Xpos;
```

Of course, for all non-intrinsic attributes, setting the attribute in the alias also sets it for the original and using the original designator is not required (though it would still work if you used it by mistake).

The intrinsic status of an attribute is not inheritable.

Attribute Listings

Attributes are the building blocks of a Tinderbox note. This section contains listings of Tinderbox's system attributes in 3 different forms:

- System Attribute List. A full list of all the predefined attributes in a Tinderbox file. There are 413 system attributes.
- Attribute Data Types. Descriptions of the different attribute Data Types (now stored outside this section).
- Attribute Groups. Descriptions of the various Groups of attributes, as seen in the Attributes dialog's System pane.
- Attributes described by purpose.

Also in this section:

- Editing attribute values.
- Attribute naming guidelines.
- Determining the data type of an attribute
- Use of the $ prefix for attribute references

Remember that the attribute value actually used at note level is subject to inheritance. As well as setting a new value to an attribute Tinderbox offers ways to:

- Set/reset an attribute's default.
- Use attributes as global variables/constants.
- Set no value for an attribute and re-enable inheritance.
- Rename an attribute.

Links to child articles:

- System Attribute List
- System Attribute Groups within Tinderbox
- Unusual attributes
- Attributes grouped by purpose

System Attribute List

System Attributes tell the System how to display, handle and export Notes and links. System Attributes are defined for every note, agent and adornment (though in the latter two cases not all attributes are used/applied).

Occasionally, Tinderbox releases may add some attributes not described here in aTbRef. These attributes are generally experimental and not intended for use by ordinary users. Such attributes will either disappear again or be formally adopte and only then described in aTbRef.

In documents used across many successive versions of the app be aware that system attribute may retain old system attributes no longer generated in new documents; these legacy items will cause no harm.

As current versions of the app do not normally define legacy-supported attributes in *new* TBX documents, the count of system attributes here may be c.10–15 items more than the count seen for a new Tinderbox document.

The attributes are broken into several groups and when listed in $DisplayedAttributes or the Note Info window, attributes are listed alphabetically within the groups. The groups are as follow, and are shown in the order they are listed in the UI:

- **System Attributes**:
  - Agent. Information pertaining to agents.
  - AI. Attributes related to Natural Language Processing (NLP) and Artificial Intelligence (AI).
  - Appearance. Attributes controlling the appearance of view icons.
  - Composites. Attribute relating to use of composites.
  - Events. Attributes relating to event date/times.
  - General. Generic information about notes. Many of these attributes are calculated read-only values.
  - Grid. Setting relating to (map view) adornment grids.
  - HTML. Settings related to HTML export.
  - Iris. Experimental (currently not used).
  - Map. Information relating to visual display of notes, agents and adornments in Map view (and other Views).
  - Net. For data derived from the Net, e.g. RSS, etc.
  - Outline. Attributes controlling Outline view related features.
  - People. For data about people and their contact info.
  - Places. For data about location.
  - Posters. For data configuring map posters.
  - References. For import of reference data.
  - Sandbox. For easy testing of action code.
  - Scrivener. For Scrivener import.
  - Sorting. Flags for sorting container content.
  - Storyspace. Data used by Tinderbox's sister app Storyspace (both use a common file format)
  - TextFormat. Information pertaining to visual display of a note's data in Note windows.
  - Textual. Text export settings and general text-related data.
  - Watch. This group holds a series of attributes to support the exchange of information with the web (and via that the iPhone/iPad) using DEVONThink, Apple Notes, and Finder folders. (The group used to be called 'Simplenote'. No longer supported apps: ~~Simplenote~~, ~~Evernote~~).
  - Weblog. Data for configuring Weblog export.
- **User (created) Attributes**. Any attributes created by the user, none are created by default. These user attributes are limited to the current document and cannot be transferred except by copying the document or using an OS Stationar document.

This is a list of all System attributes:

- $Abstract
- $AccentColor
- $AccessDate
- $Address
- $AdornmentCount
- $AdornmentFont
- $AgentAction
- $AgentCaseSensitive
- $AgentPriority
- $AgentQuery
- $Aliases
- $ArticleTitle
- $Associates
- $Author2
- $Author3
- $Author4
- $Authors
- $AutoFetch
- $AutoFetchCommand
- $AutomaticIndent
- $Badge
- $BadgeMonochrome
- $BadgeSize
- $Base
- $BeforeVisit
- $Bend
- $BookTitle

- $Border
- $BorderBevel
- $BorderColor
- $BorderDash
- $CallNumber
- $Caption
- $CaptionAlignment
- $CaptionColor
- $CaptionFont
- $CaptionOpacity
- $CaptionSize
- $Checked
- $ChildCount
- $ChosenWord
- $City
- $CleanupAction
- $ClusterTerms
- $CodeFont
- $Color
- $Color2
- $Container
- $Country
- $Created
- $CreatedFrom
- $Creator
- $Deck
- $DescendantCount
- $DEVONthinkGroup
- $DEVONthinkLabel
- $Direction
- $DisplayedAttributes
- $DisplayedAttributesDateFormat
- $DisplayedAttributesFont
- $DisplayedAttributesFontSize
- $DisplayExpression
- $DisplayExpressionDisabled
- $DisplayName
- $District
- $DOI
- $DominantLanguage
- $DueDate
- $Edict
- $EdictDisabled
- $Edition
- $Email
- $EmailSubject
- $EmailTemplate
- $EndDate
- $EstimatedNoteSize
- $EvernoteNotebook
- $File
- $Fill
- $FillOffsetY
- $FillOpacity
- $Flags
- $FormattedAddress
- $FullName
- $GeocodedAddress
- $GridColor
- $GridColumns
- $GridLabelFont
- $GridLabels
- $GridLabelSize
- $GridOpacity
- $GridRows
- $Height
- $HideDisplayedAttributes
- $HideKeyAttributes
- $HideTitle
- $HoverBackgroundColor
- $HoverExpression
- $HoverFont
- $HoverImage
- $HoverOpacity
- $HTMLBoldEnd
- $HTMLBoldStart
- $HTMLCloud1End
- $HTMLCloud1Start
- $HTMLCloud2End
- $HTMLCloud2Start
- $HTMLCloud3End
- $HTMLCloud3Start
- $HTMLCloud4End
- $HTMLCloud4Start
- $HTMLCloud5End
- $HTMLCloud5Start
- $HTMLCodeEnd
- $HTMLCodeStart
- $HTMLDontExport
- $HTMLEntities
- $HTMLExportAfter
- $HTMLExportBefore
- $HTMLExportChildren
- $HTMLExportCommand
- $HTMLExportExtension

- $HTMLExportFileName
- $HTMLExportFileNameSpacer
- $HTMLExportPath
- $HTMLExportTemplate
- $HTMLFileNameLowerCase
- $HTMLFileNameMaxLength
- $HTMLFirstParagraphEnd
- $HTMLFirstParagraphStart
- $HTMLFont
- $HTMLFontSize
- $HTMLImageEnd
- $HTMLImageStart
- $HTMLIndentedParagraphEnd
- $HTMLIndentedParagraphStart
- $HTMLItalicEnd
- $HTMLItalicStart
- $HTMLLinkExtension
- $HTMLListEnd
- $HTMLListItemEnd
- $HTMLListItemStart
- $HTMLListStart
- $HTMLMarkdown
- $HTMLMarkDown
- $HTMLMarkupText
- $HTMLOrderedListEnd
- $HTMLOrderedListItemEnd
- $HTMLOrderedListItemStart
- $HTMLOrderedListStart
- $HTMLOverwriteImages
- $HTMLParagraphEnd
- $HTMLParagraphStart
- $HTMLPreviewCommand
- $HTMLQuoteHTML
- $HTMLStrikeEnd
- $HTMLStrikeStart
- $HTMLSubscriptEnd
- $HTMLSubscriptStart
- $HTMLSuperscriptEnd
- $HTMLSuperscriptStart
- $HTMLUnderlineEnd
- $HTMLUnderlineStart
- $ID
- $IDString
- $ImageCount
- $ImageSizeLimit
- $InboundLinkCount
- $InteriorScale
- $IrisAngle
- $IrisRadius
- $IsAction
- $IsAdornment
- $IsAgent
- $IsAlias
- $ISBN
- $IsComposite
- $IsMultiple
- $IsPrototype
- $IsSeparator
- $Issue
- $IsTemplate
- $Journal
- $KeyAttributeDateFormat
- $KeyAttributeFont
- $KeyAttributeFontSize
- $KeyAttributes
- $LastFetched
- $Latitude
- $LeafBase
- $LeafBend
- $LeafDirection
- $LeafTip
- $LeftMargin
- $LineSpacing
- $LocalAttributes
- $Lock
- $Longitude
- $MapBackgroundAccentColor
- $MapBackgroundColor
- $MapBackgroundColor2
- $MapBackgroundFill
- $MapBackgroundFillOpacity
- $MapBackgroundPattern
- $MapBackgroundShadow
- $MapBodyTextColor
- $MapBodyTextSize
- $MapNameSize
- $MapPrototypeColor
- $MapScrollX
- $MapScrollY
- $MapTextSize
- $Modified
- $mt_allow_comments
- $mt_allow_pings
- $mt_convert_breaks
- $mt_keywords

- $MyBoolean
- $MyColor
- $MyDate
- $MyDictionary
- $MyInterval
- $MyList
- $MyNumber
- $MySet
- $MyString
- $Name
- $NameAlignment
- $NameBold
- $NameColor
- $NameFont
- $NameLeading
- $NameStrike
- $NeverComposite
- $NLNames
- $NLOrganizations
- $NLPlaces
- $NLTags
- $NoSpellling
- $NotesFolder
- $NotesID
- $NotesModified
- $NoteURL
- $OnAdd
- $OnJoin
- $OnRemove
- $OnVisit
- $Opacity
- $Organization
- $OutboundLinkCount
- $OutlineBackgroundColor
- $OutlineColorSwatch
- $OutlineDepth
- $OutlineNameSize
- $OutlineOrder
- $OutlineTextSize
- $Pages
- $ParagraphSpacing
- $Participants
- $Path
- $Pattern
- $PlainLinkCount
- $PlotBackgroundColor
- $PlotBackgroundOpacity
- $PlotColor
- $PlotColorList
- $PostalCode
- $PosterCSS
- $PosterLabels
- $PosterSettings
- $PosterTemplate
- $PosterURL
- $PosterX
- $PosterY
- $PosterZoom
- $Private
- $Prototype
- $PrototypeBequeathsChildren
- $PrototypeHighlightColor
- $PublicationCity
- $PublicationYear
- $Publisher
- $Range
- $RawData
- $ReadCount
- $ReadOnly
- $ReferenceDictionary
- $ReferenceRIS
- $ReferenceTitle
- $ReferenceURL
- $RefFormat
- $RefKeywords
- $RefType
- $Requirements
- $ResetAction
- $RightMargin
- $Role
- $RSSChannelTemplate
- $RSSItemLimit
- $RSSItemTemplate
- $Rule
- $RuleDisabled
- $ScreenHeight
- $ScreenWidth
- $ScrivenerID
- $ScrivenerKeywords
- $ScrivenerLabel
- $ScrivenerLabelID
- $ScrivenerNote
- $ScrivenerStatus
- $ScrivenerStatusID
- $ScrivenerType

- $Searchable
- $SelectionCount
- $Sentiment
- $Sentiments
- $Separator
- $Shadow
- $ShadowBlur
- $ShadowColor
- $ShadowDistance
- $Shape
- $ShowTitle
- $SiblingOrder
- $SimplenoteKey
- $SimplenoteModified
- $SimplenoteSync
- $SimplenoteTags
- $SimplenoteVersion
- $SmartLinks
- $SmartQuotes
- $Sort
- $SortAlso
- $SortAlsoTransform
- $SortBackward
- $SortBackwardAlso
- $SortTransform
- $SourceCreated
- $SourceModifed
- $SourceURL
- $StartDate
- $State
- $Sticky
- $Subtitle
- $SubtitleColor
- $SubtitleOpacity
- $SubtitleSize
- $SyntaxHighlighting
- $TableExpression
- $TableHeading
- $Tabs
- $Tags
- $Telephone
- $Text
- $TextAlign
- $TextBackgroundColor
- $TextColor
- $TextColorBlue
- $TextColorGray
- $TextColorGreen
- $TextColorRed
- $TextExportTemplate
- $TextFont
- $TextFontSize
- $TextHighlightBlue
- $TextHighlightGreen
- $TextHighlightMagenta
- $TextHighlightRed
- $TextHighlightYellow
- $TextLength
- $TextLinkCount
- $TextPaneRatio
- $TextPaneWidth
- $TextSidebar
- $TextWindowHeight
- $TextWindowWidth
- $TimelineAliases
- $TimelineBand
- $TimelineBandLabelColor
- $TimelineBandLabelOpacity
- $TimelineBandLabels
- $TimelineColor
- $TimelineDescendants
- $TimelineEnd
- $TimelineEndAttribute
- $TimelineGridColor
- $TimelineMarker
- $TimelineScaleColor
- $TimelineScaleColor2
- $TimelineStart
- $TimelineStartAttribute
- $Tip
- $TitleBackgroundColor
- $TitleFont
- $TitleForegroundColor
- $TitleHeight
- $TitleOpacity
- $Tot
- $Twitter
- $UpdateTextLinksAfterRename
- $URL
- $UUID
- $ViewInBrowser
- $Visits
- $Volume
- $WatchFolder
- $WebLinkCount

- $WeblogPostID
- $Width
- $WordCount
- $Xpos
- $Ypos
- $Ziplinks

Abstract

Attribute Data Type:	string [other string-type attributes]
Attribute Default Value:	(not set - empty string)
Attribute Group:	References [other References Group attributes]
Attribute Purpose:	Data synch
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	No
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

Holds the abstract from reference imports.
Previously this data was imported to $Text.

AccentColor

Attribute Data Type:	color [other color-type attributes]
Attribute Default Value:	dark warm gray dark
Attribute Group:	Appearance [other Appearance Group attributes]
Attribute Purpose:	Map item general configuration
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	Yes
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

Specifies the note's accent colour (replaces $Color2)
It is used for the alternate colour of the $Pattern drawn on the (background) $Color of a note in a map view.
It is also used for the accent colour in bar() and vbar() map progress bar patterns, as well as container plots.
The default is 'dark warm gray dark'.
The colour can be set via the Appearance Inspector ▸ Interior tab.
Retained for backwards compatibility, $Color2 is deprecated in favour of $AccentColor which fulfils the same role but with a more descriptive name. Both set the same colour.

AccessDate

Attribute Data Type:	string [other string-type attributes]
Attribute Default Value:	(not set - empty string)
Attribute Group:	References [other References Group attributes]
Attribute Purpose:	Data synch
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	No
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

Used for Bookends reference import. Maps to RIS data tag 'Y2'.

Address

Attribute Data Type:	string [other string-type attributes]
Attribute Default Value:	(not set - empty string)
Attribute Group:	Places [other Places Group attributes]
Attribute Purpose:	Locational data
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	No
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

Intended for storing a person's address.
If populated, Tinderbox will attempt automatic geolocation and to populate $Latitude and $Longitude.

AdornmentCount

Attribute Data Type:	number [other number-type attributes]
Attribute Default Value:	(calculated)
Attribute Group:	General [other General Group attributes]
Attribute Purpose:	Calculated data
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	No
Attribute Read-Only?	Yes [other read-only attributes]
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

The number of adornments that are immediate children of this agent or container.
Adornments are **not** included in $ChildCount.

AdornmentFont

Attribute Data Type:	font [other font-type attributes]
Attribute Default Value:	RingsideCondensedSSm-Light
Attribute Group:	Map [other Map Group attributes]
Attribute Purpose:	Map configuration
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	No
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

This determines the font used by adornments.

If $AdornmentFont is empty, then the value of $NameFont is used. The default value of $AdornmentFont is Ringside Condensed.

AgentAction

Attribute Data Type:	action [other action-type attributes]
Attribute Default Value:	(not set - empty string)
Attribute Group:	Agent [other Agent Group attributes]
Attribute Purpose:	Action code
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	Yes
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

Action code expression that an agent2 applies to each of its matching aliased notes, i.e. its children.

The agent's actions are applied to aliases though the exact object referenced be modified by use of the ' original' (especially for intrinsic attributes) or 'agent2' designators.

The contents of $AgentAction are shown in the Action Inspector ▸ Action sub-tab, allowing review and manual editing of its value. This attribute is the equivalent of $OnAdd for a normal container note. The agent's actual $OnAdd attribute is ignored for action purposes. $AgentAction can also be set via the ▸ agent tab.

AgentCaseSensitive

Attribute Data Type:	boolean [other boolean-type attributes]
Attribute Default Value:	false
Attribute Group:	Agent [other Agent Group attributes]
Attribute Purpose:	Agent configuration
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	No
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

Control case-sensitivity of the agent query.

Used for agents and (smart) adornments only. It can only modify the String.contains() operator (and older deprecated precursors) and not List/Set.contains().

Though not itself formally deprecated, $AgentCaseSensitive is really only of use if using now-deprecated very old early action code syntax which some long-standing documents may contain.

AgentPriority

Attribute Data Type:	number [other number-type attributes]
Attribute Default Value:	1
Attribute Group:	Agent [other Agent Group attributes]
Attribute Purpose:	Agent configuration
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	Yes
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

Allows agent to run less frequently than normal, or be turned off.

A pop-up menu in the agent's Action Inspector ▸ Query sub-tab, or Get Info ▸ agent tab, allows manual change of agent priority. Regardless of the setting, every agent is updated before export and when updating manually, *except for agent that have been turned off*.

Unless an agent is turned off before creating a query, an agent query will always run at least once as soon as the agent is first created.

The mapping of number values to the text names used in the Priority pop-up menu on the Get Info agent tab and the Action Inspector Query sub-tab is as follows:

- Highest = 0. Runs every few seconds.
- Normal = 1 (default). Runs roughly every ten seconds.
- Low = 4. Runs roughly every minute.
- Lowest = 10. Runs roughly every five minutes'
- Occasional = 20. Runs roughly every thirty minutes.
- Off = -1. Does not run (retains any existing aliases)

The $AgentPriority can only be set as a number. Do not use text values or you get an error. Any other number value than those listed above causes the dialog pop-up to show 'highest'. Whether this is the real result, i.e. that setting '9' results in the same as setting '0' is not clear.

All agents that are not 'off' run once on document load or a force refresh of agents.

The default value of $AgentPriority is 1, which will cause the agent to be checked at roughly ten second intervals. Higher values of $AgentPriority cause the agent to be checked less frequently.

Setting $AgentPriority to 0 asks Tinderbox to check the agent as frequently as possible.

Setting $AgentPriority to -1 asks Tinderbox to retain existing matches but never update, effectively turning the agent off whilst retaining the last set of data and is excluded from Export and 'manual' update agent updates. Thus the current results can be 'frozen' and persist between sessions. However, if an agent is being turned off just to drop it out of regular cycles it will need to manually be set to a value or zero or greater before it can be used/updated and manually reset to 'off' after use if it desired to 'freeze' it again. When an agent is off:

- Query does not run
- Agent actions ($AgentAction, $OnRemove) do not run.
- Agent rules and edicts still run, i.e. in the agent itself
- Child aliases' rules and edicts still run.

AgentQuery

Attribute Data Type:	string [other string-type attributes]
Attribute Default Value:	(not set - empty string)
Attribute Group:	Agent [other Agent Group attributes]
Attribute Purpose:	Action code
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	Yes
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

Action code expression an agent will use to match and gather notes.
Shown as an input box in the Action Inspector ▸ Query sub-tab of agents and smart adornments.

Aliases

Attribute Data Type:	list [other list-type attributes]
Attribute Default Value:	(not set - empty string)
Attribute Group:	General [other General Group attributes]
Attribute Purpose:	Calculated data
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	No
Attribute Read-Only?	Yes [other read-only attributes]
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

Returns a list of paths to each alias of the original of this note.
This information is seldom if ever necessary for normal use. If you need to rely on $Aliases, consider whether a different approach might be better.

ArticleTitle

Attribute Data Type:	string [other string-type attributes]
Attribute Default Value:	(not set - empty string)
Attribute Group:	References [other References Group attributes]
Attribute Purpose:	Data synch
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	No
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

Used for Bookends reference import. Maps to RIS data tag 'T1', but only for reference type CHAP (Chapter).

Associates

Attribute Data Type:	set [other set-type attributes]
Attribute Default Value:	(not set - empty string)
Attribute Group:	Composites [other Composites Group attributes]
Attribute Purpose:	Composite configuration
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	No
Attribute Read-Only?	Yes [other read-only attributes]
Attribute Intrinsic?	Yes [other intrinsic attributes]
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

A set holding the $Name strings of the (titled) items in the current composite.
For new composites, or where some notes do not yet have titles set, those notes return their $Role value instead of $Name.

Author2

Attribute Data Type:	string [other string-type attributes]
Attribute Default Value:	(not set - empty string)
Attribute Group:	References [other References Group attributes]
Attribute Purpose:	Data synch
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	No
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

Used for Bookends reference import. Maps to RIS data tag 'A2'.

Author3

Attribute Data Type:	string [other string-type attributes]
Attribute Default Value:	(not set - empty string)
Attribute Group:	References [other References Group attributes]
Attribute Purpose:	Data synch
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	No
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

Used for Bookends reference import. Maps to RIS data tag 'A3'.

Author4

Attribute Data Type:	string [other string-type attributes]
Attribute Default Value:	(not set - empty string)
Attribute Group:	References [other References Group attributes]
Attribute Purpose:	Data synch
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	No
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

Used for Bookends reference import. Maps to RIS data tag 'A4'.

Authors

Attribute Data Type:	list [other list-type attributes]
Attribute Default Value:	(not set - empty string)
Attribute Group:	References [other References Group attributes]
Attribute Purpose:	Data synch
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	No
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

Used for Bookends reference import. Maps to RIS data tag 'AU'.

$Authors is a List-type attribute (it was a Set in older releases). This is to avoid the listing being re-ordered alphabetically, now that the sorting of Set-type attributes is more aggressive in v9+. That is unhelpful if the authors of academic papers have been listed in order of appearance of the paper. If author names are entered in first name(s) then last name order, default (Set) alphabetical listing will use first name.

To purge possible duplicates now $Authors is a List, consider a stamp using the list .unique operator:

```
$Authors = $Authors.unique;
```

But, be aware this will impose an alphabetical sort.

AutoFetch

Attribute Data Type:	boolean [other boolean-type attributes]
Attribute Default Value:	false
Attribute Group:	Net [other Net Group attributes]
Attribute Purpose:	Import configuration
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	No
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

Set the note's $Text to data fetched automatically from the URL in $URL.

If `true` the content is automatically fetched when the note is opened. See more.

See also, $AutoFetchCommand.

AutoFetchCommand

Attribute Data Type:	action [other action-type attributes]
Attribute Default Value:	(not set - empty string)
Attribute Group:	Net [other Net Group attributes]
Attribute Purpose:	Import configuration
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	No
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

Action code expression used for post-import processing of $AutoFetch content.

The purpose and functioning of $AutoFetchCommand is described in AutoFetch commands.

AutomaticIndent

Attribute Data Type:	boolean [other boolean-type attributes]
Attribute Default Value:	true
Attribute Group:	TextFormat [other TextFormat Group attributes]
Attribute Purpose:	Note text
Attribute Inherited from Preferences?	Yes
Attribute UI-configurable?	No
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

Controls automatic indentation of lists in text windows.

DEPRECATED: This feature is superseded by v6+'s standard list formatting and the attribute is not present in any TBX created using v6.0+. It is retained for v5 & older compatibility only. The information below applies only to pre-v6 versions.

It defaults to `true` as inherited from the Text preference "Auto-indent lists".

If a note's $AutomaticIndent is `false` Tinderbox will not automatically indent paragraphs starting with *, •, and #. This effectively disables quick lists without needing to turn off all export formatting to do so. This is useful, for example, in note that contain CSS stylesheets that can have lines starting with a # for ID style declarations. However, it does *not* affect list creation during HTML export (see $HTMLMarkupText).

NOTE: If $AutomaticIndent is turned off in an open note that contains lists in existing $Text, the lists will retain their former indentation. You can undo this by left-indent them with the Shift Left command in the Style menu (cmd-shift=[ ). This is because the attribute tries to respect the user's intent and so does not remove existing indents, it merely prevents (or not) new ones being created automatically. If altering this attribute for an existing note, you should close the note text window before altering the value.

Badge

Attribute Data Type:	string [other string-type attributes]
Attribute Default Value:	(not set - empty string)
Attribute Group:	Appearance [other Appearance Group attributes]
Attribute Purpose:	Map item badge configuration
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	Yes
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

Name of the badge icon set for the note.

The names used in $Badge are case-sensitive. The stored value is the icon name minus the extension, e.g. "ok" instead of "ok.png". UI elements that list badges omit the icon file's extension in their roll-over tooltip.

The badge's name is listed alongside the icon in the Badges pop-up menu. Badges are optionally displayed in Map and Outline Views.

The list of default badge names is given on the Badges pop-up menu page and can also be set via the Appearance Inspector ▸ Interior tab, 'Badge' control.

Emoji and other unicode characters may be used as badges. In place of the badge name, set $Badge to the unicode character to be displayed in the badge area. Best results will generally be obtained with $BadgeSize of 32 or greater.

BadgeMonochrome

Attribute Data Type:	boolean [other boolean-type attributes]
Attribute Default Value:	false
Attribute Group:	Appearance [other Appearance Group attributes]
Attribute Purpose:	Map item badge configuration
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	No
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

Causes monochrome badges to be rendered better against a dark background.

When `true` and $Color is dark, the badge artwork will be drawn in sourceOut mode instead of sourceOver. This is useful when using black badges on a dark map background.

BadgeSize

Attribute Data Type:	number [other number-type attributes]
Attribute Default Value:	(not set - 0)
Attribute Group:	Appearance [other Appearance Group attributes]
Attribute Purpose:	Map item badge configuration
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	Yes
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

Sets the size of the badge in map view.

If zero (default), the default badge size is used. When setting a value, the number used should be the pixel count. Badge icons are square and the default set are 32x32 pixels. Thus setting a value of 16 would show a 32x32 pixel badge at half normal size (16px).

The Badge Picker pop-over automatically sets $BadgeSize to 32 when selecting badges from the Avatar family; an even-larger badge may sometimes be desirable.

The 'Large' tick-box on the Appearance Inspector ▸ Interior tab, sets this attribute to a value of 64.

Base

Attribute Data Type:	number [other number-type attributes]
Attribute Default Value:	0.5
Attribute Group:	Map [other Map Group attributes]
Attribute Purpose:	Map item general configuration
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	Yes
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

This controls the shape of arrow and leaf shapes.

It replaces the deprecated $LeafBase.

The valid range is 0-1.

For notes whose $Shape employs this setting, the $Base value for the selected note can be altered using the grey dot control to the bottom of the note, in a left-right direction.

BeforeVisit

Attribute Data Type:	action [other action-type attributes]
Attribute Default Value:	(not set - empty string)
Attribute Group:	Storyspace [other Storyspace Group attributes]
Attribute Purpose:	Storyspace synch
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	No
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

In Storyspace, an action performed before the reader visits the note.

This allows Storyspace, in read mode, this action is evaluated, will be executed before the OnVisit action. If the note has any $Requirements, it will be executed before Requirements are checked. The BeforeVisit action can be convenient for actions required to prepare for Requirement checks.

This logic is **not** tested in Tinderbox.

Bend

Attribute Data Type:	number [other number-type attributes]
Attribute Default Value:	0
Attribute Group:	Map [other Map Group attributes]
Attribute Purpose:	Map item general configuration
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	Yes
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

This control the degree of bend in the 'leaf' and 'banner' shapes.

It replaces $LeafBend which is deprecated.

The valid range is -1 to 1.

For notes whose $Shape employs this setting, the $Bend value for the selected note can be altered using the black dot control to the right of the note.

BookTitle

Attribute Data Type:	string [other string-type attributes]
Attribute Default Value:	(not set - empty string)
Attribute Group:	References [other References Group attributes]
Attribute Purpose:	Data synch
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	No
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

Used for Bookends reference import. Maps to RIS data tag 'T2'.

Border

Attribute Data Type:	number [other number-type attributes]
Attribute Default Value:	1
Attribute Group:	Appearance [other Appearance Group attributes]
Attribute Purpose:	Map item border configuration
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	Yes
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

The size (width) of the border drawn around adornments, notes, agents and containers.

If $Border is 0 (zero), no border is drawn, regardless of the $BorderBevel type.

$Border can be set—to a limited number of preset values—via the Appearance Inspector ▸ Border tabs's 'Width' control.

BorderBevel

Attribute Data Type:	string [other string-type attributes]
Attribute Default Value:	plain
Attribute Group:	Appearance [other Appearance Group attributes]
Attribute Purpose:	Map item border configuration
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	Yes
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

Styles the corder around map icons.

For adornments and items (notes, agents, containers), if $Border is greater than zero, a border is drawn around the object in a style depending on this attribute's value.

In adornments, if $BorderBevel is **raised**, the border is bevelled. If **automatic** or **plain** the border is drawn as a simple outline. If $BorderBevel is **none**, the border is not drawn; note, though, that if $Border is greater than zero, space the widt of $Border is left around the main part of the icon.

In notes, if $BorderBevel is **raised** or **automatic** the border is bevelled. If **plain**, it is a simple outline. If $BorderBevel is **none**, the border is not drawn.

$BorderBevel can be set via the Appearance Inspector ▸ Border tab's 'Style' control.

BorderColor

Attribute Data Type:	color [other color-type attributes]
Attribute Default Value:	automatic
Attribute Group:	Appearance [other Appearance Group attributes]
Attribute Purpose:	Map item border configuration
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	Yes
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

If $Border is greater than zero, a border is drawn around the adornment in this colour.

The special value, "automatic", draws an opaque border in the adornment's natural colour; adornments are otherwise translucent.

If $BorderColor is "automatic" and $BorderBevel is "plain", the colour is a mixture of the note's $Color and its $NameColor. In older versions, the border was drawn in the note's $Color, and so was invisible. $NameColor was chosen because it typically contrasts with $Color, where $AccentColor, the accent colour, might be very close in value to $Color.

$BorderColor can be set via the Appearance Inspector ▸ Border tab's 'Color' control.

BorderDash

Attribute Data Type:	number [other number-type attributes]
Attribute Default Value:	(not set - 0)
Attribute Group:	Appearance [other Appearance Group attributes]
Attribute Purpose:	Map item border configuration
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	Yes
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

Display borders of map icons as a dashed line.
Only dawn if the $BorderBevel type is 'plain'. This attribute holds the length of the dash; the width of the dashed line is the $Border width.
$BorderBevel can be set via the Appearance Inspector ▸ Border tab's 'Dash' control to a single preset value.

CallNumber

Attribute Data Type:	string [other string-type attributes]
Attribute Default Value:	(not set - empty string)
Attribute Group:	References [other References Group attributes]
Attribute Purpose:	Data synch
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	No
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

Used for Bookends reference import. Maps to RIS data tag 'CN'.

Caption

Attribute Data Type:	string [other string-type attributes]
Attribute Default Value:	(not set - empty string)
Attribute Group:	General [other General Group attributes]
Attribute Purpose:	Map item caption configuration
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	Yes
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

The optional caption text drawn beneath map note icons.
$Caption can be set via the Text Inspector ▸ Caption tab, 'Caption' text box.

CaptionAlignment

Attribute Data Type:	string [other string-type attributes]
Attribute Default Value:	left
Attribute Group:	Appearance [other Appearance Group attributes]
Attribute Purpose:	Map item caption configuration
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	Yes
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

The alignment of $Caption text.
$CaptionAlignment can be set via the Text Inspector ▸ Caption tab, 'Alignment' control.

CaptionColor

Attribute Data Type:	color [other color-type attributes]
Attribute Default Value:	blue
Attribute Group:	Appearance [other Appearance Group attributes]
Attribute Purpose:	Map item caption configuration
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	Yes
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

The colour used to drawn the $Caption.
$CaptionAlignment can be set via the Text Inspector ▸ Caption tab, 'Color' controls.

CaptionFont

Attribute Data Type:	font [other font-type attributes]
Attribute Default Value:	SketchnoteSquare
Attribute Group:	Appearance [other Appearance Group attributes]
Attribute Purpose:	Map item caption configuration
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	Yes
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

Font used to draw $Caption text.

$CaptionFont can be set via Get Info ▸ attributes tab, Appearance group. Click the 'A' symbol to open the OS Fonts palette and select the desired font.

CaptionOpacity

Attribute Data Type:	number [other number-type attributes]
Attribute Default Value:	100
Attribute Group:	Appearance [other Appearance Group attributes]
Attribute Purpose:	Map item caption configuration
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	Yes
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

The opacity used to draw the $Caption.

$CaptionAlignment can be set via the Text Inspector ▸ Caption tab, 'Opacity' control.

CaptionSize

Attribute Data Type:	number [other number-type attributes]
Attribute Default Value:	100
Attribute Group:	Appearance [other Appearance Group attributes]
Attribute Purpose:	Map item caption configuration
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	Yes
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

$Caption size is set as a relative percentage of $TextFontSize.

Default is 100.

$CaptionSize can be set via the Text Inspector ▸ Caption tab, 'Size' slider control

Checked

Attribute Data Type:	boolean [other boolean-type attributes]
Attribute Default Value:	false
Attribute Group:	General [other General Group attributes]
Attribute Purpose:	General data
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	Yes
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

Tick state of the optional Outline view checkbox.

Changing the attribute will add/remove a tick from the note's checkbox; a tick is shown when the value is `true`.

If displayed in Outline view, via View menu ▸ 'Use Checkboxes'

ChildCount

Attribute Data Type:	number [other number-type attributes]
Attribute Default Value:	(calculated)
Attribute Group:	General [other General Group attributes]
Attribute Purpose:	Calculated data
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	No
Attribute Read-Only?	Yes [other read-only attributes]
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

Number of children belonging to the current note (read-only).

This returns the count of direct children (i.e. *one* level down) and not all descendants. For a count (usually larger) of *all* notes below this see $DescendantCount.

The count excludes adornments, which are counted in $AdornmentCount.

Usually, rules and queries to test whether a note has children will test $ChildCount. But for export if you want the expression to be `false` if the note has children **but** none of the children are exported, then using ^if(^childLinks^)^ …is bette

ChosenWord

Attribute Data Type:	string [other string-type attributes]
Attribute Default Value:	(not set - empty string)
Attribute Group:	Storyspace [other Storyspace Group attributes]
Attribute Purpose:	Storyspace synch
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	No
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

In Storyspace, stores the word the reader last clicked on when viewing this note.

Note: this is **not** set/used by Tinderbox.

The value stored is the link anchor text of the text link that was clicked to leave the note, when in read mode.

City

Attribute Data Type:	string [other string-type attributes]
Attribute Default Value:	(not set - empty string)
Attribute Group:	Places [other Places Group attributes]
Attribute Purpose:	Locational data
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	No
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

The City field of an address.

The automatic geocoding tries to supply these fields automatically when looking up the supplied $Address.

CleanupAction

Attribute Data Type:	string [other string-type attributes]
Attribute Default Value:	grid
Attribute Group:	Agent [other Agent Group attributes]
Attribute Purpose:	Agent configuration
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	No
Attribute Read-Only?	No
Attribute Intrinsic?	Yes [other intrinsic attributes]
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

Controls the way in which maps of agent containers are organised.

See Re-arrangeable Agent Maps.

ClusterTerms

Attribute Data Type:	set [other set-type attributes]
Attribute Default Value:	(not set - empty string)
Attribute Group:	General [other General Group attributes]
Attribute Purpose:	Map configuration
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	No
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

A set of text terms to cluster onto an adornment during force-directed layout.

CodeFont

Attribute Data Type:	font [other font-type attributes]
Attribute Default Value:	Courier New
Attribute Group:	TextFormat [other TextFormat Group attributes]
Attribute Purpose:	Note text
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	No
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

The font type to be used for 'code' sections of $Text.

Default: Courier New

This setting is intended for sections of code where a monospace font aids clarity. This can be set via the Font sub-menu of the Format menu.

Color

Attribute Data Type:	color [other color-type attributes]
Attribute Default Value:	7
Attribute Group:	Appearance [other Appearance Group attributes]
Attribute Purpose:	Map item general configuration
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	Yes
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

Specifies the colour of a note in map view.

The same colour is used in the outline of note icons in Outline/Explorer views and element outlines in TreeMap/Chart views.

The default is '7'.

Adornments have a different default: 'blue'. Adornments also have the extra option of setting 'transparent' such that neither body or border colour are drawn.

There is no way to seed this value at app level, e.g. via user configuration files. Tinderbox saved colour schemes also do not save or set the $Color attribute.

Accent colour is provided by $AccentColor.

$Color can be set via the Appearance Inspector ▸ Interior tab, 'Color' controls.

Color2

Attribute Data Type:	color [other color-type attributes]
Attribute Default Value:	dark warm gray dark
Attribute Group:	Appearance [other Appearance Group attributes]
Attribute Purpose:	Map item general configuration
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	Yes
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

The note's accent colour (use $AccentColor instead).

Retained for backwards compatibility, $Color2 is deprecated in favour of $AccentColor which fulfils the same role but with a more descriptive name. Both set the same colour.

It is used for the alternate colour of the $Pattern drawn on the (background) $Color of a note in a map view.

It is also used for the accent colour in bar() and vbar() map progress bar patterns, as well as container plots.

The default is 'dark warm gray dark'.

$Color2 can be set via the Appearance Inspector ▸ Interior tab, 'Color' controls.

Container

Attribute Data Type:	string [other string-type attributes]
Attribute Default Value:	(preset to parent's path)
Attribute Group:	General [other General Group attributes]
Attribute Purpose:	General data
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	No
Attribute Read-Only?	No
Attribute Intrinsic?	Yes [other intrinsic attributes]
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

Lets Tinderbox actions and rules change the parent of a note, moving it to a a new place within the document.

It is the parent's $Path plus a '/'. For any item, $Path is the equivalent of $Container + $Name. More on moving notes.

The underlying default value of $Container an empty string, but this is never seen as at note creation Tinderbox always inserts the path to the note's parent, with ancestor note names delimited with a '/' (forward slash) and the $Container of any root-level note is "/".

Originally, a key use is to enable automatic onward routing of notes created via emails sent to Tinderbox (a feature currently suspended).

A more useful example would be for children of a document's 'Mail' note, used for collecting inbound email. Using monthly archives might result in a structure like '/Mail/Dec 07/', i.e. note 'Dec 07' found inside the note 'Mail'. An action in Mail might be to set the added note's container to 'Dec 07'. Thus:

```
$Container="/Mail/Dec 07";
```

…will move the inbound email note from 'Mail' to 'Dec 07' inside 'Mail'.

Note that the value passed in the action is a quoted string (now the recommended method for string literals).

An alternative to a quoted string is an attribute value:

```
$Container = $MyPathAttr
```

A neat feature of this process is that any notes/containers in the path that do not currently exist will be created. *Note: the latter is a exception to the general rule that Tinderbox will not automatically create or delete notes* . Thus for more advanced users, the 'Dec 07' note name could be a value computed from the current date allowing the document to add a new folder each month without the user having to bother with this task.

If the destination container has an $OnAdd action, that action is is performed immediately on the newly added note(s). Note that if the action tries to move a note into the container in which it currently resides the $OnAdd action is not run; this makes sense as such an action will have already been run when the note was first added.

If a note cannot be moved, either because it is being moved inside its own descendant or because the destination is not a (valid) container (e.g. an adornment or some other special note type), no action occurs.

If using an agent to set a new container, it is important to consider whether the aim is to move the *original* of the note or merely the note's *alias* within the agent. Thus an $AgentAction like:

```
$Container(original)="Some Note"
```

…moves the *original* of the note matched by the agent to a new location. Conversely:

```
$Container="Some Note"
```

…moves the currently-processed agent alias.

Important: moving the agent's alias will cause the agent's current cycle to cease, so ensure that the processed note no longer meets the agent query or only the first matched item will move its alias. Otherwise, the agent action will move one alias per cycle.

Although Tinderbox does not allow code-deletion notes, you might consider setting a 'Trash' note and using a $Container="/Trash"; action. The Trash container could then be reviewed for manual deletions, or retrieval of erroneously trashed notes.

One other important consideration when using $Container is inheritance and the effect of moving on existing inherited values. If using a lot of prototypes/inheritance it is certainly worth taking extra care with use of $Container.

Email to Tinderbox allows the same $Container-setting action to be specified within an email.

Country

Attribute Data Type:	string [other string-type attributes]
Attribute Default Value:	(not set - empty string)
Attribute Group:	Places [other Places Group attributes]
Attribute Purpose:	Locational data
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	No
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

The Country field of an address.
The automatic geocoding tries to supply these fields automatically when looking up the supplied $Address.

Created

Attribute Data Type:	date [other date-type attributes]
Attribute Default Value:	(calculated)
Attribute Group:	General [other General Group attributes]
Attribute Purpose:	Calculated data
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	No
Attribute Read-Only?	Yes [other read-only attributes]
Attribute Intrinsic?	Yes [other intrinsic attributes]
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

The date a note was originally created (read-only).
If edited subsequently, the note's $Modified attribute will changed; at outset both are the same.

CreatedFrom

Attribute Data Type:	string [other string-type attributes]
Attribute Default Value:	(not set - empty string)
Attribute Group:	General [other General Group attributes]
Attribute Purpose:	Action code
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	No
Attribute Read-Only?	No
Attribute Intrinsic?	Yes [other intrinsic attributes]
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

Used by the ziplinks text link method to store the full path of the note that created the new note.
$CreatedFrom is not inherited from prototypes.

Creator

Attribute Data Type:	string [other string-type attributes]
Attribute Default Value:	system
Attribute Group:	General [other General Group attributes]
Attribute Purpose:	Calculated data
Attribute Inherited from Preferences?	Yes
Attribute UI-configurable?	No
Attribute Read-Only?	Yes [other read-only attributes]
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

Creator of a note. The 'creator' name is taken from the User Name in Document Settings (read-only).
If the latter preference is not personalised, the default value given to new notes is "system".

Deck

Attribute Data Type:	set [other set-type attributes]
Attribute Default Value:	(not set - empty string)
Attribute Group:	Storyspace [other Storyspace Group attributes]
Attribute Purpose:	Storyspace compatibility
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	No
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

In Storyspace, the card deck(s) to which this writing space optionally belongs.
A writing space (note) will belong to one or more decks if using the sculptural hypertext formalism. Otherwise the writing space does not need to have any $Deck value.
Tinderbox can read/set this value but does not enact Storyspace's hypertext reading space.

DescendantCount

Attribute Data Type:	number [other number-type attributes]
Attribute Default Value:	(calculated)
Attribute Group:	General [other General Group attributes]
Attribute Purpose:	Calculated data
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	No
Attribute Read-Only?	Yes [other read-only attributes]
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

Number of descendants of a note (read-only).
By comparison, $ChildCount will only show the number of direct child notes to the current notes (normally a smaller figure).
Note that the count excludes adornments.

DEVONthinkGroup

Attribute Data Type:	string [other string-type attributes]
Attribute Default Value:	(not set - empty string)
Attribute Group:	Watch [other Watch Group attributes]
Attribute Purpose:	Data synch
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	No
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

The name of a DEVONthink group being fetched by this note as a watched group.
This is the long alphanumeric ID string obtained from the "Copy Item Link"

DEVONthinkLabel

Attribute Data Type:	string [other string-type attributes]
Attribute Default Value:	(not set - empty string)
Attribute Group:	Watch [other Watch Group attributes]
Attribute Purpose:	Data synch
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	No
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

Stores the DEVONthink label for a DEVONthink-imported item

Direction

Attribute Data Type:	boolean [other boolean-type attributes]
Attribute Default Value:	false
Attribute Group:	Map [other Map Group attributes]
Attribute Purpose:	Map item general configuration
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	Yes
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

The direction in which a shape should point.
This replaces the deprecated $LeafDirection.
For notes whose $Shape employs this setting, the $Direction value for the selected note can be altered using the paired arrow control to the top left of the note.

DisplayedAttributes

Attribute Data Type:	list [other list-type attributes]
Attribute Default Value:	(not set - empty string)
Attribute Group:	TextFormat [other TextFormat Group attributes]
Attribute Purpose:	Note DA
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	No
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

A List holding the attribute names used in the note's Displayed Attributes table.
Note that this is a list of attribute names and not references so do not use a $-prefix if editing the list by hand or via action code.
The Displayed Attributes table is drawn in $DisplayedAttributeFont (by default the same as $NameFont).
Displayed Attributes have no special status beyond the ability to display a user-selected table of 'key' attributes in a note's text pane.
This replaces the deprecated KeyAttributes attribute.
In pre-v.9, this attribute was a Set. However, new improvements to Set processing mean the listing order cannot be relied upon not to change, thus the move to List-type even though creates the potential for duplicated entries.

DisplayedAttributesDateFormat

Attribute Data Type:	string [other string-type attributes]
Attribute Default Value:	(not set - empty file string)
Attribute Group:	TextFormat [other TextFormat Group attributes]
Attribute Purpose:	Note DA
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	No
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

Sets a custom date format string for the note's Displayed Attributes.

By default, both the date and time are shown using the system's short formats for the current locale. Other formats may be chosen by changing the value of $DisplayedAttributesDateFormat. Suggested values include "L" and "l" to display only the date in long and short format respectively. This value will override the default set in the Text tab of Document Settings, though this attribute does not inherit directly from Document Settings.

This replaces the deprecated $KeyAttributeDateFormat attribute.

DisplayedAttributesFont

Attribute Data Type:	font [other font-type attributes]
Attribute Default Value:	IdealSansSSm-Book
Attribute Group:	TextFormat [other TextFormat Group attributes]
Attribute Purpose:	Note DA
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	No
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

Name of the font used for drawing Displayed Attributes.

The font is used for both title and value columns of the Displayed Attributes table.

By default, it is the same as $NameFont.

This replaces the deprecated $KeyAttributeFont attribute.

DisplayedAttributesFontSize

Attribute Data Type:	number [other number-type attributes]
Attribute Default Value:	11
Attribute Group:	TextFormat [other TextFormat Group attributes]
Attribute Purpose:	Note DA
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	No
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

Sets the font size used to draw the Displayed Attributes table.

The stored value is the numerical value in points. The default value is 11 (point).

This replaces the deprecated $KeyAttributeFontSize attribute.

DisplayExpression

Attribute Data Type:	action [other action-type attributes]
Attribute Default Value:	(not set - empty string)
Attribute Group:	General [other General Group attributes]
Attribute Purpose:	Action code
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	Yes
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

Action code expression to customise the way a note's name is displayed in Tinderbox views via $DisplayName.

$DisplayName holds the evaluated result of $DisplayExpression' code. If $DisplayName has a value if is used instead of $Name in many views within Tinderbox; $Name is unaffected.

Some users find it helpful to use actions and rules to append information to not names, e.g. word count, number of children or the value of some attribute. However, this does change the actual name of the note and can cause problems with other actions which do not expect note names to change.

See a further description of display expressions.

$DisplayExpression can be set via the Text Inspector ▸ Title tab, 'Display Expression' input box.

DisplayExpressionDisabled

Attribute Data Type:	boolean [other boolean-type attributes]
Attribute Default Value:	false
Attribute Group:	General [other General Group attributes]
Attribute Purpose:	Action code
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	Yes
Attribute Read-Only?	No
Attribute Intrinsic?	Yes [other intrinsic attributes]
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

Allows the $DisplayExpression for a specific note to be disabled. It is intrinsic and thus not inherited.

A primary use is suppressing code from be run in prototypes.

$DisplayExpressionDisabled is convenient when a display expression is shared by all the children of a prototype, but it is not desirable that it be applied to the prototype itself. Thus it can be useful to suppress $DisplayExpression activation prototypes. Previously, it was necessary include an 'if($Prototype)' test which was cumbersome and meant every inheriting note had to run the additional if test adding load to the action code cycle. That workaround can be discarded and the attribute set to `false` in the Prototype. Because, atypically, this attribute is not inherited from a prototype, the prototype's setting does not affect the Display Expression in notes using the prototype.

For example, the intention may be for books to be displayed with their price:

DisplayExpression: `$Name+" $"+$Price`

but the prototype for books, "Book Prototype" could inhibit this $DisplayExpression to display its name without a meaningless price tag.
A similar control is supplied for rules: $RuleDisabled.
$DisplayExpressionDisabled can be set via the Text Inspector ▸ Title tab, 'Display Expression Enabled' input box. Be aware the latter label describes the 'opposite' state compared to the underlying attribute.

DisplayName

Attribute Data Type: string [other string-type attributes]
Attribute Default Value: (calculated)
Attribute Group: General [other General Group attributes]
Attribute Purpose: Calculated data
Attribute Inherited from Preferences? No
Attribute UI-configurable? No
Attribute Read-Only? Yes [other read-only attributes]
Attribute Intrinsic? No
Attribute First Added: Baseline
Attribute Last Altered: As at baseline

Evaluated result of a Display Expression, used inside Tinderbox views instead of $Name.
Being the result of $DisplayExpression, it is read-only.
Because $DisplayExpression is evaluated contextually for the original and each alias, the resulting value can vary making $DisplayName essentially intrinsic.

District

Attribute Data Type: string [other string-type attributes]
Attribute Default Value: (not set - empty string)
Attribute Group: Places [other Places Group attributes]
Attribute Purpose: Locational data
Attribute Inherited from Preferences? No
Attribute UI-configurable? No
Attribute Read-Only? No
Attribute Intrinsic? No
Attribute First Added: Baseline
Attribute Last Altered: As at baseline

The District field of an address.
The automatic geocoding tries to supply these fields automatically when looking up the supplied $Address.
'District' equates to the 'county' for US addresses.

DOI

Attribute Data Type: string [other string-type attributes]
Attribute Default Value: (not set - empty string)
Attribute Group: References [other References Group attributes]
Attribute Purpose: Data synch
Attribute Inherited from Preferences? No
Attribute UI-configurable? No
Attribute Read-Only? No
Attribute Intrinsic? No
Attribute First Added: Baseline
Attribute Last Altered: As at baseline

Used for Bookends reference import. Maps to RIS data tag 'DO'.

DominantLanguage

Attribute Data Type: string [other string-type attributes]
Attribute Default Value: (not set - empty string)
Attribute Group: AI [other AI Group attributes]
Attribute Purpose: Natural Language Processing
Attribute Inherited from Preferences? No
Attribute UI-configurable? No
Attribute Read-Only? No
Attribute Intrinsic? No
Attribute First Added: Baseline
Attribute Last Altered: As at baseline

Records Tinderbox's guess at the primary language used in the text of each note.
Languages appear as two-letter ISO-639-1 codes such as “en” for English, “de” for German, and “zh” for Chinese.

DueDate

Attribute Data Type: date [other date-type attributes]
Attribute Default Value: (not set - never)
Attribute Group: Events [other Events Group attributes]
Attribute Purpose: Timeline event configuration
Attribute Inherited from Preferences? No
Attribute UI-configurable? No
Attribute Read-Only? No
Attribute Intrinsic? No
Attribute First Added: Baseline
Attribute Last Altered: As at baseline

A date attribute provided to facilitate organisation of tasks.
See also $StartDate and $EndDate.

Edict

Attribute Data Type:	action [other action-type attributes]
Attribute Default Value:	(not set - empty string)
Attribute Group:	General [other General Group attributes]
Attribute Purpose:	Action code
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	No
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

Sets an edict (action code) to be run on the note, but infrequently.

In effect this is a low priority rule ($Rule) for tasks that do require to run more than once but not with great frequency. Execution of the edict can be suppressed via $EditDisabled. This allows the edict code to remain stored in the attribute bu for it to have no effect.

$Edict can be set via the Action Inspector ▸ Edict tab's input box.

EdictDisabled

Attribute Data Type:	boolean [other boolean-type attributes]
Attribute Default Value:	false
Attribute Group:	General [other General Group attributes]
Attribute Purpose:	Action code
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	Yes
Attribute Read-Only?	No
Attribute Intrinsic?	Yes [other intrinsic attributes]
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

This enables a note's $Edict to be disabled. It is intrinsic and thus not inherited via prototypes.

A primary use is suppressing code from be run in prototypes.

$EdictDisabled can be set via the Action Inspector ▸ Edict tab's 'Enabled' tick-box. Be aware the latter label describes the 'opposite' state compared to the underlying attribute.

Edition

Attribute Data Type:	string [other string-type attributes]
Attribute Default Value:	(not set - empty string)
Attribute Group:	References [other References Group attributes]
Attribute Purpose:	Data synch
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	No
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

Used for Bookends reference import. Maps to RIS data tag 'ET'.

Email

Attribute Data Type:	string [other string-type attributes]
Attribute Default Value:	(not set - empty string)
Attribute Group:	People [other People Group attributes]
Attribute Purpose:	Person detail
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	No
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	9.6.0

Intended for storing a person's email address.

From v9.6.0, $Email is of data type Email. Previously it had been of data type String.

EmailSubject

Attribute Data Type:	string [other string-type attributes]
Attribute Default Value:	(not set - empty string)
Attribute Group:	People [other People Group attributes]
Attribute Purpose:	HTML export file configuration
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	No
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	9.6.0
Attribute Last Altered:	As at baseline

Sets the subject of a Tinderbox generated email

New to v9.6.0, used with the create draft email feature.

EmailTemplate

Attribute Data Type:	string [other string-type attributes]
Attribute Default Value:	(not set - empty string)
Attribute Group:	People [other People Group attributes]
Attribute Purpose:	HTML export file configuration
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	No
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	9.6.0
Attribute Last Altered:	As at baseline

A template for formatting draft emails

New to v9.6.0, this set the (export) template used for creating draft emails where more that the current notes styled $Text is needed as the message body text.

EndDate

Attribute Data Type:	date [other date-type attributes]
Attribute Default Value:	(not set - never)
Attribute Group:	Events [other Events Group attributes]
Attribute Purpose:	Timeline event configuration
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	No
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

A date attribute provided to facilitate organisation of tasks.

See also $StartDate and $DueDate.

For Timeline views, events with only $EndDate or an $EndDate before $StartDate will not plot correctly. Thus if $EndDate is used, a $StartDate should be supplied. An alternate Date-type attribute to use can be supplied via $TimelineEndAttribute.

EstimatedNoteSize

Attribute Data Type:	number [other number-type attributes]
Attribute Default Value:	0
Attribute Group:	Textual [other Textual Group attributes]
Attribute Purpose:	Calculated data
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	No
Attribute Read-Only?	Yes [other read-only attributes]
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

This provides a *rough* estimate of the space that the note will require on disk.

This attribute provides a means to help with maintenance of large—in terms of file size—TBX documents as the user can track down the largest notes to review content. Although embedded images are a driver of file size, there are other factors such as: many attributes with local (i.e. note-level) values set, much text, or text with a lot of styling features.

However, embedded images/documents are the likeliest cause of very large notes, and this is the primary purpose of this attribute.

EvernoteNotebook

Attribute Data Type:	string [other string-type attributes]
Attribute Default Value:	(not set - empty string)
Attribute Group:	Watch [other Watch Group attributes]
Attribute Purpose:	Data synch
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	No
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

The name of an Evernote notebook used by a watched folder.

See more on Evernote use. Now unused/deprecated, as Evernote sync is not longer supported.

File

Attribute Data Type:	file [other file-type attributes]
Attribute Default Value:	(not set - empty file string)
Attribute Group:	General [other General Group attributes]
Attribute Purpose:	General data
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	No
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

Specifies the OS path to the object (in POSIX form).

The object can be either a file or a folder. Only one file may be attached per note; to attach several files drop a folder containing the desired files. This linked asset may also be an OS alias to a file or folder.

Adding a new note to $File replaces the existing reference. $File can also be set by dropping a file on the folder icon of $File when displayed in a note's displayed attribute's table.

When opened via the Displayed Attributes table icon, $File's contents is passed to Finder which opens a finder window (for folders) or opens the file according to the files OS-associated application.

Fill

Attribute Data Type:	file [other file-type attributes]
Attribute Default Value:	(not set - empty string)
Attribute Group:	Map [other Map Group attributes]
Attribute Purpose:	Map item general configuration
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	No
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

Designates a source image for background textures for map notes and adornments.
The $Fill is normally set via the Appearance/ Interior Inspector's **Texture** setting.
The opacity of the fill texture can be controlled by the $FillOpacity attribute.
Additional images may be placed in the custom fills folder of the Application Support folder.
A file name may be used with or without an extension. If the user file folder contains the file "TestFill.png", then setting $Fill to "TestFill" or to "TestFill.png" is equivalent.
Originally the attribute type was a string but this changed to file type. That made it easier to change fill to an arbitrary image file in the Displayed Attributes table. The value of $Fill is thus file path, e.g. `~/Pictures/Projects/logo.jpg`.

FillOffsetY

Attribute Data Type:	number [other number-type attributes]
Attribute Default Value:	(not set - 0)
Attribute Group:	Map [other Map Group attributes]
Attribute Purpose:	Map item general configuration
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	No
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

This moves the centre of the fill image vertically.
The value is set in map units (for Y, *minus* is up). See map note icon image fills.

FillOpacity

Attribute Data Type:	number [other number-type attributes]
Attribute Default Value:	0.75
Attribute Group:	Map [other Map Group attributes]
Attribute Purpose:	Map item general configuration
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	No
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

Opacity of the map icon $Fill texture, or the icon's fill colours ($Color, $AccentColor) to be varied.
A value of 1 makes the fill opaque, while 0 makes the fill invisible. The default value is 0.75.

Flags

Attribute Data Type:	list [other list-type attributes]
Attribute Default Value:	(not set - empty string)
Attribute Group:	Appearance [other Appearance Group attributes]
Attribute Purpose:	Map item general configuration
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	No
Attribute Read-Only?	No
Attribute Intrinsic?	Yes [other intrinsic attributes]
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

Holds a list of strings defining one or more flags.

FormattedAddress

Attribute Data Type:	string [other string-type attributes]
Attribute Default Value:	(not set - empty string)
Attribute Group:	Places [other Places Group attributes]
Attribute Purpose:	Locational data
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	No
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

Data from automatic geocoding of $Address.
This attribute is retained for compatibility purposes but is deprecated: use $GeocodedAddress instead
This address may include zip code and additional information absent from the supplied $Address.

FullName

Attribute Data Type:	string [other string-type attributes]
Attribute Default Value:	(not set - empty string)
Attribute Group:	People [other People Group attributes]
Attribute Purpose:	Person detail
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	No
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

Intended for storing the name of a person.

GeocodedAddress

Attribute Data Type:	string [other string-type attributes]
Attribute Default Value:	(not set - empty string)
Attribute Group:	Places [other Places Group attributes]
Attribute Purpose:	Locational data
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	No
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

Retrieved address look-up data as passed to automatic geocoding process.
The stored address may include zip code and additional information absent from the supplied $Address.
This attribute replaces $FormattedAddress.

GridColor

Attribute Data Type:	color [other color-type attributes]
Attribute Default Value:	#ffffff
Attribute Group:	Grid [other Grid Group attributes]
Attribute Purpose:	Map item grid configuration
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	Yes
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

The colour used to draw an adornment's grid.
Set via the Grid Properties pop-over.

GridColumns

Attribute Data Type:	number [other number-type attributes]
Attribute Default Value:	(not set - 0)
Attribute Group:	Grid [other Grid Group attributes]
Attribute Purpose:	Map item grid configuration
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	Yes
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

The number of columns used to draw an adornment's grid.
Set via the Grid Properties pop-over.

GridLabelFont

Attribute Data Type:	font [other font-type attributes]
Attribute Default Value:	IdealSansSSm-Book
Attribute Group:	Grid [other Grid Group attributes]
Attribute Purpose:	Map item grid configuration
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	Yes
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

Font used for adornment grid labels.
Set via the Grid Properties pop-over.

GridLabels

Attribute Data Type: list [other list-type attributes]
Attribute Default Value: (not set - empty string)
Attribute Group: Grid [other Grid Group attributes]
Attribute Purpose: Map item grid configuration
Attribute Inherited from Preferences? No
Attribute UI-configurable? Yes
Attribute Read-Only? No
Attribute Intrinsic? No
Attribute First Added: Baseline
Attribute Last Altered: As at baseline

The set of labels used for an adornment's grid.
Set via the Grid Properties pop-over.

GridLabelSize

Attribute Data Type: number [other number-type attributes]
Attribute Default Value: 10
Attribute Group: Grid [other Grid Group attributes]
Attribute Purpose: Map item grid configuration
Attribute Inherited from Preferences? No
Attribute UI-configurable? Yes
Attribute Read-Only? No
Attribute Intrinsic? No
Attribute First Added: Baseline
Attribute Last Altered: As at baseline

The sized used for drawing labels on an adornment's grid.
Set via the Grid Properties pop-over.

GridOpacity

Attribute Data Type: number [other number-type attributes]
Attribute Default Value: 75
Attribute Group: Grid [other Grid Group attributes]
Attribute Purpose: Map item grid configuration
Attribute Inherited from Preferences? No
Attribute UI-configurable? Yes
Attribute Read-Only? No
Attribute Intrinsic? No
Attribute First Added: Baseline
Attribute Last Altered: As at baseline

The opacity applied to an adornment's grid.
Set via the Grid Properties pop-over.

GridRows

Attribute Data Type: number [other number-type attributes]
Attribute Default Value: (not set - 0)
Attribute Group: Grid [other Grid Group attributes]
Attribute Purpose: Map item grid configuration
Attribute Inherited from Preferences? No
Attribute UI-configurable? Yes
Attribute Read-Only? No
Attribute Intrinsic? No
Attribute First Added: Baseline
Attribute Last Altered: As at baseline

The number of rows used to draw an adornment's grid.
Set via the Grid Properties pop-over.

Height

Attribute Data Type: number [other number-type attributes]
Attribute Default Value: 1.86
Attribute Group: Map [other Map Group attributes]
Attribute Purpose: Map item general configuration
Attribute Inherited from Preferences? No
Attribute UI-configurable? No
Attribute Read-Only? No
Attribute Intrinsic? Yes [other intrinsic attributes]
Attribute First Added: Baseline
Attribute Last Altered: As at baseline

The height of a note icon in the Map view, in map units.
See also $Width.
Aliases do not inherit this property from their original but have their own value (i.e. it is an intrinsic attribute).

HideDisplayedAttributes

Attribute Data Type:	boolean [other boolean-type attributes]
Attribute Default Value:	false
Attribute Group:	TextFormat [other TextFormat Group attributes]
Attribute Purpose:	Note DA
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	No
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

Toggles visibility of the Displayed Attributes table (if any are set).
This replaces the deprecated $HideKeyAttributes attribute.

HideKeyAttributes

Attribute Data Type:	boolean [other boolean-type attributes]
Attribute Default Value:	false
Attribute Group:	TextFormat [other TextFormat Group attributes]
Attribute Purpose:	Note KA
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	No
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

Deprecated: Hides key attributes table, even if defined for this note.
Deprecated in favour of the $HideDisplayedAttributes attribute.
In v6+, this replaces the $TextSidebar feature.

HideTitle

Attribute Data Type:	boolean [other boolean-type attributes]
Attribute Default Value:	false
Attribute Group:	TextFormat [other TextFormat Group attributes]
Attribute Purpose:	Note title
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	Yes
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

Toggles display of a note's title and subtitle in map view.
Hiding the title allows more space for text display on the note's map icon.

HoverBackgroundColor

Attribute Data Type:	color [other color-type attributes]
Attribute Default Value:	black
Attribute Group:	General [other General Group attributes]
Attribute Purpose:	Map item general configuration
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	No
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	9.6.0
Attribute Last Altered:	As at baseline

Controls the background colour of Hover Expressions.
New to v9.6.0 this controls the background of the pop-up used by Hover Expressions.

HoverExpression

Attribute Data Type:	action [other action-type attributes]
Attribute Default Value:	(not set - empty string)
Attribute Group:	General [other General Group attributes]
Attribute Purpose:	Action code
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	Yes
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

Holds an expression forming the Hover Expression code for a note.
This combines with Hover Image, if any. Text is rendered in $HoverFont.
$HoverExpression can be set via the Text Inspector ▸ Hover tab, 'Hover Expression' text box.

HoverFont

Attribute Data Type:	font [other font-type attributes]
Attribute Default Value:	RingsideCondensedSSm-Med
Attribute Group:	Map [other Map Group attributes]
Attribute Purpose:	Map item general configuration
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	No
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

The font used to draw $HoverExpression.

HoverImage

Attribute Data Type:	file [other file-type attributes]
Attribute Default Value:	(not set - empty string)
Attribute Group:	General [other General Group attributes]
Attribute Purpose:	Map item general configuration
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	Yes
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

Allows pop-up display of an image when hovering over a note.
This combines with the Hover Expression, if any.
$HoverImage can be set via the Text Inspector ▸ Hover tab, 'image file' drop-target box.

HoverOpacity

Attribute Data Type:	number [other number-type attributes]
Attribute Default Value:	0.5
Attribute Group:	General [other General Group attributes]
Attribute Purpose:	Map item general configuration
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	Yes
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

Specifies the opacity of the note's $HoverExpression panel when visible.
Allows specification of the opacity of hover expressions on a scale of zero (invisible) to 1 (opaque). The Hover pane of the Text Inspector has a slider to control opacity.
$HoverOpacity can be set via the Text Inspector ▸ Hover tab, 'Opacity' control.

HTMLBoldEnd

Attribute Data Type:	string [other string-type attributes]
Attribute Default Value:	`</b>`
Attribute Group:	HTML [other HTML Group attributes]
Attribute Purpose:	HTML export mark-up
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	Yes
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

Ending tag to use for closing bold passages of the current note's text being exported to HTML.
The opening tag is set via $HTMLBoldStart.
$HTMLBoldEnd can be set via the Export Inspector's ▸ Style tab, 'Bold' 'end' input box.

HTMLBoldStart

Attribute Data Type:	string [other string-type attributes]
Attribute Default Value:	`<b>`
Attribute Group:	HTML [other HTML Group attributes]
Attribute Purpose:	HTML export mark-up
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	Yes
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

Starting tag to use for opening bold passages of the current note's text being exported to HTML.
The closing tag is set via $HTMLBoldEnd.
$HTMLBoldStart can be set via the Export Inspector's ▸ Style tab, 'Bold' 'start' input box.

HTMLCloud1End

Attribute Data Type:	string [other string-type attributes]
Attribute Default Value:	`</span>`
Attribute Group:	HTML [other HTML Group attributes]
Attribute Purpose:	HTML export mark-up
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	No
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

Ending mark-up used to set the font size on words exported via the ^cloud, ^sectionCloud^ and ^documentCloud^ codes.
There are five different ranges of words that can have sizes set; 1 = worst match and 5 = best match.

HTMLCloud1Start

Attribute Data Type:	string [other string-type attributes]
Attribute Default Value:	`<span style="font-size: 0.75em">`
Attribute Group:	HTML [other HTML Group attributes]
Attribute Purpose:	HTML export mark-up
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	No
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

Starting mark-up used to set the font size of words exported via the ^cloud^, ^sectionCloud^ and ^documentCloud^ codes.
There are five different ranges of words that can have sizes set; 1 = worst match and 5 = best match.

HTMLCloud2End

Attribute Data Type:	string [other string-type attributes]
Attribute Default Value:	`</span>`
Attribute Group:	HTML [other HTML Group attributes]
Attribute Purpose:	HTML export mark-up
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	No
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

Ending mark-up used to set the font size on words exported via the ^cloud^, ^sectionCloud^ and ^documentCloud^ codes.
There are five different ranges of words that can have sizes set; 1 = worst match and 5 = best match.

HTMLCloud2Start

Attribute Data Type:	string [other string-type attributes]
Attribute Default Value:	`<span style="font-size: 1em">`
Attribute Group:	HTML [other HTML Group attributes]
Attribute Purpose:	HTML export mark-up
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	No
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

Starting mark-up used to set the font size of words exported via the ^cloud^, ^sectionCloud^ and ^documentCloud^ codes.
There are five different ranges of words that can have sizes set; 1 = worst match and 5 = best match.

HTMLCloud3End

Attribute Data Type:	string [other string-type attributes]
Attribute Default Value:	`</span>`
Attribute Group:	HTML [other HTML Group attributes]
Attribute Purpose:	HTML export mark-up
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	No
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

Ending mark-up used to set the font size on words exported via the ^cloud^, ^sectionCloud^ and ^documentCloud^ codes.
There are five different ranges of words that can have sizes set; 1 = worst match and 5 = best match.

HTMLCloud3Start

Attribute Data Type: string [other string-type attributes]
Attribute Default Value: `<span style="font-size: 1.4em">`
Attribute Group: HTML [other HTML Group attributes]
Attribute Purpose: HTML export mark-up
Attribute Inherited from Preferences? No
Attribute UI-configurable? No
Attribute Read-Only? No
Attribute Intrinsic? No
Attribute First Added: Baseline
Attribute Last Altered: As at baseline

Starting mark-up used to set the font size of words exported via the ^cloud^, ^sectionCloud^ and ^documentCloud^ codes.
There are five different ranges of words that can have sizes set; 1 = worst match and 5 = best match.

HTMLCloud4End

Attribute Data Type: string [other string-type attributes]
Attribute Default Value: `</span>`
Attribute Group: HTML [other HTML Group attributes]
Attribute Purpose: HTML export mark-up
Attribute Inherited from Preferences? No
Attribute UI-configurable? No
Attribute Read-Only? No
Attribute Intrinsic? No
Attribute First Added: Baseline
Attribute Last Altered: As at baseline

Ending mark-up used to set the font size on words exported via the ^cloud^, ^sectionCloud^ and ^documentCloud^ codes.
There are five different ranges of words that can have sizes set; 1 = worst match and 5 = best match.

HTMLCloud4Start

Attribute Data Type: string [other string-type attributes]
Attribute Default Value: `<span style="font-size: 2em">`
Attribute Group: HTML [other HTML Group attributes]
Attribute Purpose: HTML export mark-up
Attribute Inherited from Preferences? No
Attribute UI-configurable? No
Attribute Read-Only? No
Attribute Intrinsic? No
Attribute First Added: Baseline
Attribute Last Altered: As at baseline

Starting mark-up used to set the font size of words exported via the ^cloud^, ^sectionCloud^ and ^documentCloud^ codes.
There are five different ranges of words that can have sizes set; 1 = worst match and 5 = best match.

HTMLCloud5End

Attribute Data Type: string [other string-type attributes]
Attribute Default Value: `</span>`
Attribute Group: HTML [other HTML Group attributes]
Attribute Purpose: HTML export mark-up
Attribute Inherited from Preferences? No
Attribute UI-configurable? No
Attribute Read-Only? No
Attribute Intrinsic? No
Attribute First Added: Baseline
Attribute Last Altered: As at baseline

Ending mark-up used to set the font size on words exported via the ^cloud^, ^sectionCloud^ and ^documentCloud^ codes.
There are five different ranges of words that can have sizes set; 1 = worst match and 5 = best match.

HTMLCloud5Start

Attribute Data Type: string [other string-type attributes]
Attribute Default Value: `<span style="font-size: 2.8em">`
Attribute Group: HTML [other HTML Group attributes]
Attribute Purpose: HTML export mark-up
Attribute Inherited from Preferences? No
Attribute UI-configurable? No
Attribute Read-Only? No
Attribute Intrinsic? No
Attribute First Added: Baseline
Attribute Last Altered: As at baseline

Starting mark-up used to set the font size of words exported via the ^cloud^, ^sectionCloud^ and ^documentCloud^ codes.
There are five different ranges of words that can have sizes set; 1 = worst match and 5 = best match.

HTMLCodeEnd

Attribute Data Type:	string [other string-type attributes]
Attribute Default Value:	`</code>`
Attribute Group:	HTML [other HTML Group attributes]
Attribute Purpose:	HTML export encoding
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	Yes
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

HTML code to insert at the beginning of code-format $Text.
Can be edited via the Style pane of the Export Inspector.

HTMLCodeStart

Attribute Data Type:	string [other string-type attributes]
Attribute Default Value:	`<code>`
Attribute Group:	HTML [other HTML Group attributes]
Attribute Purpose:	HTML export encoding
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	Yes
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

HTML code to insert at the beginning of code-format $Text.
Can be edited via the Style pane of the Export Inspector.

HTMLDontExport

Attribute Data Type:	boolean [other boolean-type attributes]
Attribute Default Value:	false
Attribute Group:	HTML [other HTML Group attributes]
Attribute Purpose:	HTML export scope
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	Yes
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

Specifies whether the content of the note will export or not during export to HTML, either as a page or as an inclusion in a page.
This attribute is the setting for the 'Export' tick-box in the HTML page view. The default for pages is to Export (ticked) so this attribute's default, in keeping with its titling is `false`, i.e. avoiding a double negative "do not $HTMLDontExport".
Tinderbox does not ignore paths that include containers that are not exported. For example

```
^include(/documentation/details/note)^
```

…previously would have included nothing if **documentation** or **details** were had HTMLDontExport set to true, even if **note** had $HTMLDontExport set to `false`.
Tinderbox does not create links to notes whose $HTMLDontExport value is true. The same holds for links where any ancestor has $HTMLExportChildren set to `false`.
$HTMLDontExport can be set via the Export Inspector's ▸ Export tab, 'Export' tick box.

HTMLEntities

Attribute Data Type:	boolean [other boolean-type attributes]
Attribute Default Value:	false
Attribute Group:	HTML [other HTML Group attributes]
Attribute Purpose:	HTML export encoding
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	No
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

Indicates that non-basic characters like ampersands are converted into HTML entities
The move to UTF-8 encoded export since version 5 has lessened the need to this setting. The default is `false`.
Note: until v8 the default value of $HTMLEntities was `true`; the current should reduce confusion for those accustomed to using utf-8 in HTML.
If `true` an ampersand '&' is exported as `&`.
Added the degree symbol to the characters turned into entities. Swiss, French and German quote styles are automatically detected and 'entified' without needing to resort to this setting.
Export markup elements, e.g. ^value()^, check this attribute before encoding values.
English typographic quotation marks “ ” and ‘ ’ are exported as HTML entities rather than straight quotes, e.g. `&#147;` for “ left curly double quotes.
The symbol € is exported as `&euro;`.
Δ is exported as `&Delta;`, δ as `&delta;`, μ as `&mu;` and ∑ as `&sum;`.

HTMLExportAfter

Attribute Data Type:	string [other string-type attributes]
Attribute Default Value:	(not set - empty string)
Attribute Group:	HTML [other HTML Group attributes]
Attribute Purpose:	HTML export mark-up
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	Yes
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

Ending tag immediately after the end of the current note's text and/or includes.
Effectively, this is the 'footer' for each exported note's HTML page; q.v. $HTMLExportBefore.
$HTMLExportAfter can be set via the Export Inspector's ▸ Markup tab, 'Before/After' 'end' box.

HTMLExportBefore

Attribute Data Type:	string [other string-type attributes]
Attribute Default Value:	(not set - empty string)
Attribute Group:	HTML [other HTML Group attributes]
Attribute Purpose:	HTML export mark-up
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	Yes
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

Starting tag immediately before the current note's text and/or includes.

Effectively, this is the 'header' for each exported note's HTML page; q.v. $HTMLExportAfter.

$HTMLExportBefore can be set via the Export Inspector's ▸ Markup tab, 'Before/After' 'start' box.

HTMLExportChildren

Attribute Data Type:	boolean [other boolean-type attributes]
Attribute Default Value:	true
Attribute Group:	HTML [other HTML Group attributes]
Attribute Purpose:	HTML export scope
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	Yes
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

An attribute controlling export of the current note's children.

The interpretation of $HTMLDontExport and $HTMLExportChildren has been simplified.

$HTMLExportChildren can be set via the Export Inspector's ▸ Export tab, 'Export Children' tick-box.

If Export As Page is checked, Tinderbox will create an HTML file for the note. If it is not checked (`true`), no file will be created. The note's contents might appear on another HTML page, of course, by being ^include^-d by name or as one of another node's ^descendants^.

If Export Children is checked, Tinderbox will attempt to export the node's children (if any). Each child, of course, may have an Export As Page attribute.

Both Export Children and Export as Page are consulted when exporting, and the two settings are completely independent. We can export a page but not its children; we can export the children but not export any HTML for a note, or we can create both an HTML page and a directory of child pages.

Remember that if this attribute is false, Tinderbox never even examines the child notes when exporting to HTML. So individual child notes may have $HTMLDontExport set either `true` or `false`, and will not be exported in either case.

The easiest way to set this attribute is with the "Export children as pages" checkbox on the HTML view window for the note.

Some existing Tinderbox documents might generate some extra files or directories, because these settings formerly interacted in a complex way. The new rules are much simpler to understand, and are much more consistent.

Tinderbox does not create links to notes where any ancestor has $HTMLExportChildren set to `false`. The same holds for creating links to notes whose $HTMLDontExport value is `false`.

HTMLExportCommand

Attribute Data Type:	string [other string-type attributes]
Attribute Default Value:	(not set - empty string)
Attribute Group:	HTML [other HTML Group attributes]
Attribute Purpose:	HTML export post-processing
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	No
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

Run a program or script on the contents of the exported HTML file. Where is the working directory location assumed for executing scripts ?

If $HTMLExportCommand is empty, no special action is performed.

Otherwise, the exported file is piped to the standard input of the command line in $HTMLExportCommand, and the file is replaced by command line's standard output. In other words, any action carried out by the linked command is carried out on the exported page *after* the normal page export.

This attribute needs to be actual command line code: it cannot use code stored in another note as runCommand() can.

HTMLExportExtension

Attribute Data Type:	string [other string-type attributes]
Attribute Default Value:	.html
Attribute Group:	HTML [other HTML Group attributes]
Attribute Purpose:	HTML export file configuration
Attribute Inherited from Preferences?	Yes
Attribute UI-configurable?	Yes
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

File extension to use for exported HTML pages (includes preceding full stop).

The default value is '.html'. A new note inherits the default but the value can be altered at note level by editing the note's Get Info or via a stamp. Editing allows for other extensions, such as ASP/PHP/etc., to be specified as well as other forms like '.txt', '.js' or '.md'.

$HTMLExportExtension can be set via the Export Inspector's ▸ Export tab, 'Extension' box.

If the note's default is altered it is saved as the note's $HTMLExportExtension value.

HTMLExportFileName

Attribute Data Type:	string [other string-type attributes]
Attribute Default Value:	(not set - empty string)
Attribute Group:	HTML [other HTML Group attributes]
Attribute Purpose:	HTML export file configuration
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	Yes
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

Specific filename to be used for the note's exported HTML page instead of default calculated name.

If left empty, Tinderbox automatically generates a filename based on the note's title.

Note that a string set for this attribute should be the desired filename *without* its extension, i.e. 'index' not index.html'.

Automatic filenames strip unsafe (see below) characters from the note title and truncate the resulting string to the number of characters set in $HTMLFileNameMaxLength (where the default is 24). The case of characters in the exported name will be as per the note title except if $HTMLFileNameLowerCase is `true`.

Default: (not set: empty string)

$HTMLExportFileName can be set via the Export Inspector's ▸ Export tab, 'File Name' box.

To avoid filename naming collisions, Tinderbox has to check if the intended export name already exists in the currently exported-to folder. If a duplicate name might arise, a suffix will be added. Simplistically, if file "abc.html" is already being exported to the current folder, another note that would export with that name will be exported as "abc_1.html" (and -@, _3, etc.). From v9.5.0, punctuation characters other than a forward slash, period, and tilde (`/.~`) are also allowed in export filenames.

Because the filename *might* change on export, Tinderbox does not return a value for `eval($HTMLExportFileName)`; an exception is if this attribute already has an explicitly set value. The workaround is to use ^file()^, noting that the latter returns both file name *and* extension rather than just the name portion.

As Windows-OS web servers support fewer characters in file paths/names that other OSs, the following characters are suppressed when generating filenames from $Name:

```
[space] / \ ? % * : | ' " < > . & + ( ) ! #
```

Further manipulation is available via $HTMLExportFileNameSpacer.

$HTMLExportPath evaluates the likely exported filename and path.

HTMLExportFileNameSpacer

Attribute Data Type:	string [other string-type attributes]
Attribute Default Value:	space
Attribute Group:	HTML [other HTML Group attributes]
Attribute Purpose:	HTML export file configuration
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	No
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

Character used to replace any character normally suppressed on export.

It allows more flexibility with the construction of filenames created via HTML Export.

Reflecting the fact that long-term Tinderbox-created websites might be affected, the default value is an empty string, thus maintaining the status quo.

If used, it is most likely needed for all exported pages and thus should be set by changing the attribute default (i.e. at document level) rather than by setting the attribute at note level.

From v9.5.0, Tinderbox the default is now a space " " character rather than previous default of an underscore (_) character or not character at all.

Punctuation characters other than a forward slash, period, and tilde (`/.~`) are also allowed in export filenames.

HTMLExportPath

Attribute Data Type:	string [other string-type attributes]
Attribute Default Value:	(calculated)
Attribute Group:	HTML [other HTML Group attributes]
Attribute Purpose:	HTML export file configuration
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	No
Attribute Read-Only?	Yes [other read-only attributes]
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

The calculated export folder/filename path for the note (read-only)

The folders and file from export root that would be created if the file were exported at that time. It thus show the names of the folders & file with any characters omitted or substituted as part of the export process.

This can be extremely useful in more complex export scenarios where there is a need to reference in code an exported note's OS filename or path without being forced to hard-set $HTMLExportFileName. If not set, the latter cannot be quer within Tinderbox.

To get just the exported filename from the path:

```
$MyString = $HTMLExportPath.split("/").at(-1)
```

For this note:

$Path = "*/A Tinderbox Reference File/Automating Tinderbox/Coding/Use of Attributes/Attribute Listings/System Attribute List/HTMLExportPath* "

$HTMLExportPath = "*/index/Automating_Tinderbox/Coding/Use_of_Attributes/Attribute_Listings/System_Attribute_List/HTMLExportPath.html*"

Note the spaces being lost in the latter. The export version will also show $HTMLExportFileNameSpacer changes, if the latter is set.

HTMLExportTemplate

Attribute Data Type:	string [other string-type attributes]
Attribute Default Value:	(not set - empty string)
Attribute Group:	HTML [other HTML Group attributes]
Attribute Purpose:	HTML export encoding
Attribute Inherited from Preferences?	Yes
Attribute UI-configurable?	Yes
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

Name of the external template file or internal template to be used in exporting this note to HTML.

$HTMLExportTemplate can be set via the Export Inspector's ▸ Export tab, 'Template' control or by using the Template pop-up menu on the note's text pane's HTML tab will also set this attribute for the note. The attribute value is the templa note's $Path.

To set a value via the UI, at least one valid export template must be defined in the current document. If at least one valid template is defined, the first (by date of addition) is taken as the default and shown in the Export tab of the HTML Inspector.

HTMLFileNameLowerCase

Attribute Data Type:	boolean [other boolean-type attributes]
Attribute Default Value:	false
Attribute Group:	HTML [other HTML Group attributes]
Attribute Purpose:	HTML export file configuration
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	No
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

Letter case used for the HTML export filename for the note.

If set to 'true', this forces the exported HTML filenames to be all lowercase. This is useful if exporting to a server whose OS is case sensitive.

The default is `false`.

HTMLFileNameMaxLength

Attribute Data Type:	number [other number-type attributes]
Attribute Default Value:	100
Attribute Group:	HTML [other HTML Group attributes]
Attribute Purpose:	HTML export file configuration
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	No
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

Specifies the maximum number of characters allowed in exported filenames.

From v9.5.0, the default value is now `100`.

This change is an increase up from 8 characters in Tinderbox v1 and 24 characters in v3.0.5: previous values reflecting erstwhile OS filename and WebURL constraints.

HTMLFirstParagraphEnd

Attribute Data Type:	string [other string-type attributes]
Attribute Default Value:	`</p>`
Attribute Group:	HTML [other HTML Group attributes]
Attribute Purpose:	HTML export mark-up
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	Yes
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

Ending tag for the current note's first paragraph (and only the first paragraph).

The opening value is set via $HTMLFirstParagraphStart.

Export mark-up for all paragraphs succeeding the first are controlled by $HTMLParagraphStart and $HTMLParagraphEnd. Export mark-up for ***all*** paragraphs starting with a **tab** are controlled by $HTMLIndentedParagraphStart and $HTMLIndentedParagraphEnd.

$HTMLFirstParagraphEnd can be set via the Export Inspector's ▸ Markup tab, 'First paragraph' 'end' box.

HTMLFirstParagraphStart

Attribute Data Type:	string [other string-type attributes]
Attribute Default Value:	`<p>`
Attribute Group:	HTML [other HTML Group attributes]
Attribute Purpose:	HTML export mark-up
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	Yes
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

Starting tag for the note's first paragraph (and only the first paragraph).

The closing value is set via $HTMLFirstParagraphEnd.

Values for all paragraphs succeeding the first are controlled by $HTMLParagraphStart and $HTMLParagraphEnd. Export mark-up for *all* paragraphs starting with a **tab** are controlled by $HTMLIndentedParagraphStart and $HTMLIndentedParagraphEnd.

$HTMLFirstParagraphStart can be set via the Export Inspector's ▸ Markup tab, 'First paragraph' 'start' box.

HTMLFont

Attribute Data Type:	font [other font-type attributes]
Attribute Default Value:	Andale Mono
Attribute Group:	HTML [other HTML Group attributes]
Attribute Purpose:	General data
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	No
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

Sets the font used in the HTML (view) sub-tab of the Text pane.

Defaults to Andale Mono.

HTMLFontSize

Attribute Data Type:	number [other number-type attributes]
Attribute Default Value:	13
Attribute Group:	HTML [other HTML Group attributes]
Attribute Purpose:	General data
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	No
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

Sets the size of font used in the HTML (view) sub-tab of the Text pane.
Defaults to 13 point.

HTMLImageEnd

Attribute Data Type:	string [other string-type attributes]
Attribute Default Value:	`>`
Attribute Group:	HTML [other HTML Group attributes]
Attribute Purpose:	HTML export mark-up
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	No
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

Ending tag for images embedded in notes.
The default value is `>` but for XHTML use alter the value to `/>`. The paired attribute is $HTMLImageStart.
This, and its pair attribute $HTMLIMageStart, allow for insertion of extra HTML tag attributes, e.g. CSS information.

HTMLImageStart

Attribute Data Type:	string [other string-type attributes]
Attribute Default Value:	`<img`
Attribute Group:	HTML [other HTML Group attributes]
Attribute Purpose:	HTML export mark-up
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	No
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

Start tag for images embedded in notes.
The paired attribute is $HTMLImageEnd.
This, and its pair attribute $HTMLIMageEnd, allow for insertion of extra HTML tag attributes, e.g. CSS information.

HTMLIndentedParagraphEnd

Attribute Data Type:	string [other string-type attributes]
Attribute Default Value:	`</blockquote>`
Attribute Group:	HTML [other HTML Group attributes]
Attribute Purpose:	HTML export mark-up
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	Yes
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

Ending tag for any paragraph that begins with a **tab** character, will use this attribute rather than $HTMLFirstParagraphEnd or $HTMLParagraphEnd.
The opening export mark-up for such paragraphs is supplied by $HTMLIndentedParagraphStart.
The default is `</blockquote>` though for files created pre-v5, the old default of `</p>` will be retained so as not to upset existing output.
This default may be changed, or overridden, to generate a different indentation, to change the margins, etc.
See also $HTMLParagraphEnd and $HTMLFirstParagraphEnd.
$HTMLIndentedParagraphEnd can be set via the Export Inspector's ▸ Markup tab, 'Indented' 'end' box.

HTMLIndentedParagraphStart

Attribute Data Type:	string [other string-type attributes]
Attribute Default Value:	`<blockquote>`
Attribute Group:	HTML [other HTML Group attributes]
Attribute Purpose:	HTML export mark-up
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	Yes
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

Starting tag for any paragraph that begins with a **tab** character (used instead of $HTMLFirstParagraphStart or $HTMLParagraphStart).
The closing mark-up for such paragraphs is supplied by $HTMLIndentedParagraphEnd.
The default is `<blockquote>` though for files created pre-v5, the old default of `<p style="text-indent: 3em;">` will be retained so as not to upset existing output.
This default may be changed, or overridden, to generate a different indentation, to change the margins, etc.
See also $HTMLParagraphStart and $HTMLFirstParagraphStart.
$HTMLIndentedParagraphStart can be set via the Export Inspector's ▸ Markup tab, 'Indented' 'start' box.

HTMLItalicEnd

Attribute Data Type:	string [other string-type attributes]
Attribute Default Value:	`</i>`
Attribute Group:	HTML [other HTML Group attributes]
Attribute Purpose:	HTML export mark-up
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	Yes
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

Ending tag to use for closing italic passages of the current note's text being exported to HTML.
The code can also be set via the HTML Inspector's Style sub-tab. The opening tag is set via $HTMLItalicStart.
$HTMLItalicEnd can be set via the Export Inspector's ▸ Style tab, 'Italic' 'end' input box.

HTMLItalicStart

Attribute Data Type:	string [other string-type attributes]
Attribute Default Value:	`<i>`
Attribute Group:	HTML [other HTML Group attributes]
Attribute Purpose:	HTML export mark-up
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	Yes
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

Starting tag to use for opening italic passages of the current note's text being exported to HTML.
The code can also be set via the HTML views Style pane. The closing tag is set via $HTMLItalicEnd.
$HTMLItalicStart can be set via the Export Inspector's ▸ Style tab, 'Italic' 'start' input box.

HTMLLinkExtension

Attribute Data Type:	string [other string-type attributes]
Attribute Default Value:	(not set - empty string)
Attribute Group:	HTML [other HTML Group attributes]
Attribute Purpose:	HTML export post-processing
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	No
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

If post-processing used, this indicates the extension used when generating inter-file links.
Necessary as the extension may differ from $HTMLExportExtension if the latter is set for post-processing needs, e.g. to '.md' as opposed to '.html'. In such a case $HTMLLinkExtension would be set to '.html' if that were the file extension use
as the result of post processing.

HTMLListEnd

Attribute Data Type:	string [other string-type attributes]
Attribute Default Value:	`</ul>`
Attribute Group:	HTML [other HTML Group attributes]
Attribute Purpose:	HTML export mark-up
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	Yes
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

Ending tag to use for closing bulleted (unordered) lists in the current note's text being exported to HTML.
The opening tag is set via $HTMLListStart.
If the $HTMLListItemStart and $HTMLListItemEnd attributes are empty, all quick lists, as defined by lines starting with an * or a #, are disabled.
$HTMLListEnd can be set via the Export Inspector's ▸ List tab, 'Unordered' 'end' box.

HTMLListItemEnd

Attribute Data Type:	string [other string-type attributes]
Attribute Default Value:	`</li>`
Attribute Group:	HTML [other HTML Group attributes]
Attribute Purpose:	HTML export mark-up
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	Yes
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

Ending tag to use for closing list items in the current note's text being exported to HTML.
The opening tag is set via $HTMLListItemStart.
If $HTMLListItemStart and $HTMLListItemEnd are empty, quick lists, as defined by lines starting with an * or a #, are disabled.
$HTMLListItemEnd can be set via the Export Inspector's ▸ List tab, 'List item' 'end' box.

HTMLListItemStart

Attribute Data Type:	string [other string-type attributes]
Attribute Default Value:	`<li>`
Attribute Group:	HTML [other HTML Group attributes]
Attribute Purpose:	HTML export mark-up
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	Yes
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

Starting tag to use for opening list items in the current note's text being exported to HTML.

The closing tag is set via $HTMLListItemEnd.

If $HTMLListItemStart and $HTMLListItemEnd are empty, quick lists, as defined by lines starting with an * or a #, are disabled.

$HTMLListItemStart can be set via the Export Inspector's ▸ List tab, 'List item' 'start' box.

HTMLListStart

Attribute Data Type:	string [other string-type attributes]
Attribute Default Value:	`<ul>`
Attribute Group:	HTML [other HTML Group attributes]
Attribute Purpose:	HTML export mark-up
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	Yes
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

Starting tag to use for opening bulleted (unordered) lists in the current note's text being exported to HTML.

The closing tag is set via $HTMLListEnd.

Changing this attribute's tag to `<ul type="circle">` with allowable *type* values of `circle`, `square` or `disc` lets you alter the bullet type.

If the $HTMLListItemStart and $HTMLListItemEnd attributes are empty, all quick lists, as defined by lines starting with an * or a #, are disabled.

$HTMLListStart can be set via the Export Inspector's ▸ List tab, 'Unordered' 'start' box.

HTMLMarkdown

Attribute Data Type:	boolean [other boolean-type attributes]
Attribute Default Value:	false
Attribute Group:	HTML [other HTML Group attributes]
Attribute Purpose:	HTML export post-processing
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	No
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

Allow processing of ^value^ tags while not treating Markdown headings as ordered lists.

Replaces the older and now deprecated $HTMLMarkDown.

HTMLMarkDown

Attribute Data Type:	boolean [other boolean-type attributes]
Attribute Default Value:	false
Attribute Group:	HTML [other HTML Group attributes]
Attribute Purpose:	HTML export post-processing
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	No
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

Obsolete: used to allow processing of tags while not treating headings (## heading) as ordered lists.

Do not use, only found in older TBXs.

No longer processed and function taken over by $HTMLMarkdown.

HTMLMarkupText

Attribute Data Type:	boolean [other boolean-type attributes]
Attribute Default Value:	true
Attribute Group:	HTML [other HTML Group attributes]
Attribute Purpose:	HTML export encoding
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	Yes
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

Ensures any HTML code (e.g. code examples) within the note text export as HTML entities rather than literal HTML code.

Thus a > character is exported as `&gt;` so as to show as > on screen and not be parsed as HTML code.

You would use a non-default value of `false` for any note where you have included HTML mark-up that has already been HTML 'entified' in the note text (and so a `&gt;` character in text is exported unchanged as `&gt;` thus rendering as source code, etc.).

$HTMLMarkupText can be set via the Export Inspector's ▸ Export tab, 'Markup text' box.

Whereas HTMLQuoteHTML just looks at mark-up code such as HTML/XML/PHP/etc. in the note's source text, HTMLMarkupText considers other aspects of the note. For example, if you set HTMLMarkupText to `false` macros are not expanded, styles are ignored, quick links are not created and so forth. As the attribute title implies you have turned mark-up off and you're just passing out raw text into the HTML export template. Depending on the latter's mark-up you may o may not see un-marked-up text in your browser.

Thus the default setting of `true` is almost invariably the correct setting for this attribute.

The attribute also controls, or has a part in, several display-related export functions (quick lists, auto-headings, etc.).

Note that $AutomaticIndent separately controls quick list layout within a note text window without affecting HTML export processing on quick list markers.

HTMLOrderedListEnd

Attribute Data Type:	string [other string-type attributes]
Attribute Default Value:	`</ol>`
Attribute Group:	HTML [other HTML Group attributes]
Attribute Purpose:	HTML export mark-up
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	Yes
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

Ending tag markup for ordered list generated by quick lists.
HTML ordered lists may be created by beginning new paragraphs with the '#' symbol. Ordered and regular (unordered) quick lists should not be mixed.
The starting tag is set via $HTMLOrderedListStart.
If the $HTMLListItemStart and $HTMLListItemEnd attributes are empty, all quick lists, as defined by lines starting with an * or a #, are disabled.
$HTMLOrderedListEnd can be set via the Export Inspector's ▸ List tab, 'Ordered' 'end' box.

HTMLOrderedListItemEnd

Attribute Data Type:	string [other string-type attributes]
Attribute Default Value:	`</li>`
Attribute Group:	HTML [other HTML Group attributes]
Attribute Purpose:	HTML export mark-up
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	No
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

Ending tag to use for opening ordered list items in the current note's text being exported to HTML.
The closing tag is set via $HTMLOrderedListItemStart.
If $HTMLOrderedListItemStart and $HTMLOrderedListItemEnd are empty, quick lists, as defined by lines starting with an * or a #, are disabled.

HTMLOrderedListItemStart

Attribute Data Type:	string [other string-type attributes]
Attribute Default Value:	`<li>`
Attribute Group:	HTML [other HTML Group attributes]
Attribute Purpose:	HTML export mark-up
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	No
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

Starting tag to use for opening ordered list items in the current note's text being exported to HTML.
The closing tag is set via $HTMLListItemEnd.
If $HTMLOrderedListItemStart and $HTMLOrderedListItemEnd are empty, quick lists, as defined by lines starting with an * or a #, are disabled.

HTMLOrderedListStart

Attribute Data Type:	string [other string-type attributes]
Attribute Default Value:	`<ol>`
Attribute Group:	HTML [other HTML Group attributes]
Attribute Purpose:	HTML export mark-up
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	Yes
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

Opening tag markup for ordered list generated by quick lists.
HTML ordered lists may be created by beginning new paragraphs with the '#' symbol. Ordered and regular (unordered) quick lists should not be mixed.
The closing tag is set via $HTMLOrderedListEnd. Changing this to the syntax `<ol type="a">` lets you alter the list item's 'bullet' type. The allowed values of `type` are:

- `A` capital letters: A, B, C, D, etc.
- `a` lower case letters: a, b, c, d, etc.
- `I` Capital Roman numerals: I, II, III, IV, etc.
- `i` Lower case roman numerals: i, ii, iii, iv, etc.
- `1` Arabic numerals: 1, 2, 3, 4, etc.

If the $HTMLListItemStart and $HTMLListItemEnd attributes are empty, all quick lists, as defined by lines starting with an * or a #, are disabled.
$HTMLOrderedListStart can be set via the Export Inspector's ▸ List tab, 'Ordered' 'start' box.

HTMLOverwriteImages

Attribute Data Type:	boolean [other boolean-type attributes]
Attribute Default Value:	false
Attribute Group:	HTML [other HTML Group attributes]
Attribute Purpose:	HTML export post-processing
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	No
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

Deprecated: Legacy support for per-v6 use.

Instructs Tinderbox not to overwrite existing images when exporting.

In some circumstances, exporting the compressed image from Tinderbox might be undesirable, e.g. when a more detailed, uncompressed image is wanted in the actual export. By setting $HTMLOverwriteImages and placing the uncompressed image file in the export folder, Tinderbox will avoid replacing the uncompressed file.

HTMLParagraphEnd

Attribute Data Type:	string [other string-type attributes]
Attribute Default Value:	`</p>`
Attribute Group:	HTML [other HTML Group attributes]
Attribute Purpose:	HTML export mark-up
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	Yes
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

Text and/or ending HTML mark-up to be included immediately after the end of all of the current note's paragraphs, except the first.

The opening value is set via $HTMLParagraphStart.

Export mark-up for all paragraphs succeeding the first are controlled by $HTMLFirstParagraphStart and $HTMLFirstParagraphEnd. Export mark-up for all paragraphs starting with a tab are controlled by $HTMLIndentedParagraphStart and $HTMLIndentedParagraphEnd.

$HTMLParagraphEnd can be set via the Export Inspector's ▸ Markup tab, 'Paragraph' 'end' box.

HTMLParagraphStart

Attribute Data Type:	string [other string-type attributes]
Attribute Default Value:	`<p>`
Attribute Group:	HTML [other HTML Group attributes]
Attribute Purpose:	HTML export mark-up
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	Yes
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

Text and/or starting HTML mark-up to be included immediately before all of the current note's paragraphs, except the first.

The closing value is set via $HTMLParagraphEnd.

Export mark-up for all paragraphs succeeding the first are controlled by $HTMLFirstParagraphStart and $HTMLFirstParagraphEnd. Export mark-up for all paragraphs starting with a tab are controlled by $HTMLIndentedParagraphStart and $HTMLIndentedParagraphEnd.

$HTMLParagraphStart can be set via the Export Inspector's ▸ Markup tab, 'Paragraph' 'start' box.

HTMLPreviewCommand

Attribute Data Type:	string [other string-type attributes]
Attribute Default Value:	(not set - empty string)
Attribute Group:	HTML [other HTML Group attributes]
Attribute Purpose:	HTML export post-processing
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	No
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

Stores path to an alternate export processor (e.g. Markdown)

If a path is specified, the stated application is used instead of the usual HTML mark-up generation when processing the (preview) output of ^text^.

See more on Markdown use in export and Markdown preview rendering (latter includes possible in-app values for this attribute).

HTMLQuoteHTML

Attribute Data Type:	boolean [other boolean-type attributes]
Attribute Default Value:	false
Attribute Group:	HTML [other HTML Group attributes]
Attribute Purpose:	HTML export encoding
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	Yes
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

Indicates that any valid HTML mark-up in the source text of the of the current note will be rendered as text (i.e. exported as HTML entities) instead of as HTML code.

$HTML can be set via the Export Inspector's ▸ Export tab,

For 'HTML mark-up' you can read any code using mark-up tag's written within in <> angle brackets, e.g. XML, PHP, etc. Thus, a `true` setting exports a '<br>' in the source text as '<br>' that shows as screen text whereas a `false` setting exports it as '<br>', i.e. HTML source code for a (visible) line break.

Values exported via ^value()^ and such will be parsed for HTML entities in addition to note text.

When set to `true`, where sections of mark-up include line breaks in the Tinderbox note, it does not output the line returns after each line of code, i.e. they form continuous text in the HTML output, *unless* the there is at least one character at the beginning or end of the line outside the code within the outermost < > characters.

For formatting discrete paragraphs/blocks of code, see the note on Exporting code samples.

HTMLStrikeEnd

Attribute Data Type:	string [other string-type attributes]
Attribute Default Value:	`</strike>`
Attribute Group:	HTML [other HTML Group attributes]
Attribute Purpose:	HTML export mark-up
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	Yes
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

Ending tag to use for opening strikethrough passages of the current note's text being exported to HTML.
The closing tag is set via $HTMLStrikeStart.
$HTMLStrikeEnd can be set via the Export Inspector's ▸ Style tab, 'Strike through' 'end' input box.

HTMLStrikeStart

Attribute Data Type:	string [other string-type attributes]
Attribute Default Value:	`<strike>`
Attribute Group:	HTML [other HTML Group attributes]
Attribute Purpose:	HTML export mark-up
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	Yes
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

Starting tag to use for opening strikethrough passages of the current note's text being exported to HTML.
The closing tag is set via $HTMLStrikeEnd.
$HTMLStrikeStart can be set via the Export Inspector's ▸ Style tab, 'Strike through' 'start' input box.

HTMLSubscriptEnd

Attribute Data Type:	string [other string-type attributes]
Attribute Default Value:	`</sub>`
Attribute Group:	HTML [other HTML Group attributes]
Attribute Purpose:	HTML export mark-up
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	No
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	9.5.2
Attribute Last Altered:	As at baseline

Ending tag for exported HTML markup for subscript text in notes.
The paired attribute is $HTMLSubscriptStart.

HTMLSubscriptStart

Attribute Data Type:	string [other string-type attributes]
Attribute Default Value:	`<sub>`
Attribute Group:	HTML [other HTML Group attributes]
Attribute Purpose:	HTML export mark-up
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	No
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	9.5.2
Attribute Last Altered:	As at baseline

Starting tag for exported HTML markup for subscript text in notes.
The paired attribute is $HTMLSubscriptEnd.

HTMLSuperscriptEnd

Attribute Data Type:	string [other string-type attributes]
Attribute Default Value:	`</sup>`
Attribute Group:	HTML [other HTML Group attributes]
Attribute Purpose:	HTML export mark-up
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	No
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	9.5.2
Attribute Last Altered:	As at baseline

Ending tag for exported HTML markup for superscript text in notes.
The paired attribute is $HTMLSuperscriptStart.

HTMLSuperscriptStart

Attribute Data Type:	string [other string-type attributes]
Attribute Default Value:	`<sup>`
Attribute Group:	HTML [other HTML Group attributes]
Attribute Purpose:	HTML export mark-up
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	No
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	9.5.2
Attribute Last Altered:	As at baseline

Starting tag for exported HTML markup for superscript text in notes.

The paired attribute is $HTMLSuperscriptEnd.

HTMLUnderlineEnd

Attribute Data Type:	string [other string-type attributes]
Attribute Default Value:	`</u>`
Attribute Group:	HTML [other HTML Group attributes]
Attribute Purpose:	HTML export mark-up
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	Yes
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

Ending tag for exported HTML mark-up for underlined text in notes.

The paired attribute is $HTMLUnderlineStart.

$HTMLUnderlineEnd can be set via the Export Inspector's ▸ Style tab, 'Underline' 'end' input box.

HTMLUnderlineStart

Attribute Data Type:	string [other string-type attributes]
Attribute Default Value:	`<u>`
Attribute Group:	HTML [other HTML Group attributes]
Attribute Purpose:	HTML export mark-up
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	Yes
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

Starting tag for exported HTML markup for underlined text in notes.

The paired attribute is $HTMLUnderlineEnd.

$HTMLUnderlineStart can be set via the Export Inspector's ▸ Style tab, 'Underline' 'start' input box.

ID

Attribute Data Type:	number [other number-type attributes]
Attribute Default Value:	(calculated)
Attribute Group:	General [other General Group attributes]
Attribute Purpose:	Calculated data
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	No
Attribute Read-Only?	Yes [other read-only attributes]
Attribute Intrinsic?	Yes [other intrinsic attributes]
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

The internal unique ID of the item and is a large arbitrary whole number (read-only, not sequential numbers).

Although set to no value (0) by default it is immediately populated with the items UID. The $ID UID number is unique within a TBX.

$ID opens a number of possibilities for export, including making HTML anchors for linking to includes in exported pages but note the newer $IDString offers a more robust alternative for new code.

An object dragged or copy/pasted to a different TB acquires a new $ID, as this ensures the UID is unique in the new location (i.e. its existing source $ID might already be allocated in the receiving document).

As the newer $IDString may be used interchangeably with $ID, $IDString is suggested as the more robust choice when referring to IDs. Few users, except those using much/complex action code need to work with IDs. It is suggested to use $IDString for new code, but do not rush to update all existing code from $ID to $IDString unless there is an indication it is needed.

Alaises and IDs

IDs of aliases *do not persist between sessions*. In other words, if a note X has an $ID of "3219936174" today, it will have the same ID next week, but aliases of X might have different IDs. Aliases created by agents may change IDs at any time. However, aliases inside inactive agents do retain their ID value, *unless/until* the agent is re-activated. As $IDString is calculated from $ID, the same limitations apply.

IDString

Attribute Data Type:	string [other string-type attributes]
Attribute Default Value:	(not set - empty string)
Attribute Group:	General [other General Group attributes]
Attribute Purpose:	Calculated data
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	No
Attribute Read-Only?	Yes [other read-only attributes]
Attribute Intrinsic?	Yes [other intrinsic attributes]
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

A more compact, string, alternative to $ID

From v9.5.0, IDString, provides a more compact alternative to $ID. It is auto-calculated from $ID and thus is not seen in the saved XML source code of TBX files As a String it is less liable to accidental mathematical transformation when used in action code expressions.

For example, a typical value of $ID might be `1666143079`, and its corresponding $IDString is `tbx:BjT1Nn`.

As $IDString may be used interchangeably with $ID, $IDString is suggested as the more robust choice when referring to IDs. Few users, except those using much/complex action code need to work with IDs. It is suggested to use $IDString fo
new code, but do not rush to update all existing code from $ID to $IDString unless there is an indication it is needed.

ImageCount

Attribute Data Type:	number [other number-type attributes]
Attribute Default Value:	(calculated)
Attribute Group:	Textual [other Textual Group attributes]
Attribute Purpose:	Calculated data
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	No
Attribute Read-Only?	Yes [other read-only attributes]
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

The number of images embedded in the note's text.

This can be useful to know when exporting notes with embedded images.

ImageSizeLimit

Attribute Data Type:	number [other number-type attributes]
Attribute Default Value:	1600
Attribute Group:	TextFormat [other TextFormat Group attributes]
Attribute Purpose:	Note text
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	No
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

The maximum image width for images placed in $Text.

New to v9.5.0.

Default value is `1600` (px).

InboundLinkCount

Attribute Data Type:	number [other number-type attributes]
Attribute Default Value:	(calculated)
Attribute Group:	General [other General Group attributes]
Attribute Purpose:	Calculated data
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	No
Attribute Read-Only?	Yes [other read-only attributes]
Attribute Intrinsic?	Yes [other intrinsic attributes]
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

Number of inbound links terminating at the current note (read-only).

See also $OutboundLinkCount.

The link count excludes any links of type "Prototype".

InteriorScale

Attribute Data Type:	number [other number-type attributes]
Attribute Default Value:	1.5
Attribute Group:	Map [other Map Group attributes]
Attribute Purpose:	Map configuration
Attribute Inherited from Preferences?	Yes
Attribute UI-configurable?	No
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

Factor by which notes inside a container's viewport are smaller than notes elsewhere in the current map.

The default value is 1.5, so if $InteriorScale is set to 2, then interior (child) notes are half their normal map size. $InteriorScale is inherited via a Map Preference: Map interior scale.

Larger magnifications (values) give somewhat more detail of interior notes, i.e. greater number of lower level notes visible within the 'viewport' area of container notes in Map View. Thus, to increase the number of lower level note icons seen, for a given container viewport size, increase the value of $InteriorScale.

IrisAngle

Attribute Data Type:	number [other number-type attributes]
Attribute Default Value:	0
Attribute Group:	Iris [other Iris Group attributes]
Attribute Purpose:	Experimental
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	No
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

Experimental attribute, not currently used.

IrisRadius

Attribute Data Type:	number [other number-type attributes]
Attribute Default Value:	1
Attribute Group:	Iris [other Iris Group attributes]
Attribute Purpose:	Experimental
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	No
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

Experimental attribute, not currently used.

IsAction

Attribute Data Type:	boolean [other boolean-type attributes]
Attribute Default Value:	false
Attribute Group:	General [other General Group attributes]
Attribute Purpose:	Action code
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	No
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

This identifies notes that contain action code.
These notes receive auto-completion support and action code syntax colouring.
This attribute is most likely to be set via the Action prototype, especially for internal stamp notes.

IsAdornment

Attribute Data Type:	boolean [other boolean-type attributes]
Attribute Default Value:	false
Attribute Group:	General [other General Group attributes]
Attribute Purpose:	Calculated data
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	No
Attribute Read-Only?	Yes [other read-only attributes]
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

Indicates that an object is an adornment.

IsAgent

Attribute Data Type:	boolean [other boolean-type attributes]
Attribute Default Value:	false
Attribute Group:	Textual [other Textual Group attributes]
Attribute Purpose:	Calculated data
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	No
Attribute Read-Only?	Yes [other read-only attributes]
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

Indicates if the object is an agent.
$IsAgent is `true` for agents and `false` for all other notes.
This attribute makes it easier to simply find—or exclude—agents when constructing an agent or conditional query.
Note: $IsAgent is not `true` for smart adornments as queries can't match adornments.

IsAlias

Attribute Data Type:	boolean [other boolean-type attributes]
Attribute Default Value:	false
Attribute Group:	General [other General Group attributes]
Attribute Purpose:	Calculated data
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	No
Attribute Read-Only?	Yes [other read-only attributes]
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

Indicates that an item is an alias (read-only)
The attribute can help when trying to find originals as distinct from their aliases. It also helps when querying for aliases.
It is likely this reads the state of the internal Boolean attribute $Alias.

ISBN

Attribute Data Type:	string [other string-type attributes]
Attribute Default Value:	(not set - empty string)
Attribute Group:	References [other References Group attributes]
Attribute Purpose:	Data synch
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	No
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

Used for Bookends reference import. Maps to RIS data tag 'SN'.

IsComposite

Attribute Data Type:	boolean [other boolean-type attributes]
Attribute Default Value:	false
Attribute Group:	Composites [other Composites Group attributes]
Attribute Purpose:	Composite configuration
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	No
Attribute Read-Only?	Yes [other read-only attributes]
Attribute Intrinsic?	Yes [other intrinsic attributes]
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

Defines if a note is part of a composite.

If `true` the note is currently part of a composite.

IsMultiple

Attribute Data Type:	boolean [other boolean-type attributes]
Attribute Default Value:	false
Attribute Group:	Composites [other Composites Group attributes]
Attribute Purpose:	Composite configuration
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	No
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

Indicates if a composite item's role may have multiple notes.

If `true` that role may occur more than once in the composite.

If $IsMultiple is *true*, when a note is moved into the composite touching that note, the moved note inherits the note's role.

IsPrototype

Attribute Data Type:	boolean [other boolean-type attributes]
Attribute Default Value:	false
Attribute Group:	General [other General Group attributes]
Attribute Purpose:	Special note type designator
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	Yes
Attribute Read-Only?	No
Attribute Intrinsic?	Yes [other intrinsic attributes]
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

Indicates this note is a prototype and can be used as such for by other notes.

A `true` value implies the note is available as a prototype.

Prototype values are inherited when the prototype is applied, normally at note creation. Subsequent editing of a prototype is not reflected through to notes based on that prototype; other methods must be used to update existing notes based on older prototype values.

$IsPrototype can be set via the Properties Inspector ▸ Prototype tab, 'Prototype' tick-box.

IsSeparator

Attribute Data Type:	boolean [other boolean-type attributes]
Attribute Default Value:	false
Attribute Group:	Outline [other Outline Group attributes]
Attribute Purpose:	Outline configuration
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	Yes
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	9.6.0
Attribute Last Altered:	As at baseline

Controls whether a note or agent is rendered as a separator in Outline view.

From v9.6.0, $IsSeparator replaces $Separator, which is now deprecated, but continues to work.

If notes are given no name, i.e. shown just as a line, a duplicate name warning is not issued even if reporting of such is turned on as a Warning Preference. The text of the separator is centred if it is smaller than the width of the separator label box.

$IsSeparator can be set via the Properties Inspector ▸ Prototype tab, 'Separator' tick-box.

Issue

Attribute Data Type:	string [other string-type attributes]
Attribute Default Value:	(not set - empty string)
Attribute Group:	References [other References Group attributes]
Attribute Purpose:	Data synch
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	No
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

Used for Bookends reference import. Maps to RIS data tag 'IS'.

IsTemplate

Attribute Data Type:	boolean [other boolean-type attributes]
Attribute Default Value:	false
Attribute Group:	HTML [other HTML Group attributes]
Attribute Purpose:	Special note type designator
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	Yes
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

Defines a note for use as an export template.

The attribute may also be set via the tick-box on the Properties Inspector's Prototype sub-tab or via action/rule code.

$IsTemplate can be set via the Properties Inspector ▸ Prototype tab, 'Template' tick-box.

Journal

Attribute Data Type:	string [other string-type attributes]
Attribute Default Value:	(not set - empty string)
Attribute Group:	References [other References Group attributes]
Attribute Purpose:	Data synch
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	No
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

Used for Bookends reference import. Maps to RIS data tag 'JA'.

KeyAttributeDateFormat

Attribute Data Type:	string [other string-type attributes]
Attribute Default Value:	(not set - empty string)
Attribute Group:	TextFormat [other TextFormat Group attributes]
Attribute Purpose:	Note KA
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	No
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

Deprecated: Sets a custom date format string for the note's Key Attributes.

Deprecated in favour of the $DisplayedAttributesDateFormat attribute.

By default, both the date and time are shown using the system's short formats for the current locale. Other formats may be chosen by changing the value of $KeyAttributeDateFormat. Suggested values include "L" and "l" to display only the date in long and short format respectively. This value will override the default set in the Text tab of Document Settings.

KeyAttributeFont

Attribute Data Type:	font [other font-type attributes]
Attribute Default Value:	IdealSansSSm-Book
Attribute Group:	TextFormat [other TextFormat Group attributes]
Attribute Purpose:	Note KA
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	No
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

Deprecated: Name of the font used for drawing Key Attributes.

Deprecated in favour of the DisplayedAttributesFont attribute.

The font is used for both title and value columns of the key attributes table.

By default, it is the same as $NameFont.

KeyAttributeFontSize

Attribute Data Type:	number [other number-type attributes]
Attribute Default Value:	11
Attribute Group:	TextFormat [other TextFormat Group attributes]
Attribute Purpose:	Note KA
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	No
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

Deprecated: Sets the font size used to draw the Displayed Attributes table.
Deprecated in favour of the $DisplayedAttributesFontSize attribute.
The stored value is the numerical value in points. The default value is 11 (point).

KeyAttributes

Attribute Data Type:	set [other set-type attributes]
Attribute Default Value:	(not set - empty string)
Attribute Group:	TextFormat [other TextFormat Group attributes]
Attribute Purpose:	Note KA
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	No
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

Deprecated: A Set of attribute names of those attributes to be displayed at the top of the current note's text pane.
Note that this is a list of attribute *names* and not references so do not use a $-prefix if editing the list by hand or via action code.
The key attributes table is drawn in $KeyAttributeFont (by default the same as $NameFont).
Deprecated in favour of the $DisplayedAttributes attribute. Displayed Attributes have replaced Key Attributes.
Key Attributes have no special status beyond the ability to display a user-selected table of 'key' attributes in a note's text pane.

LastFetched

Attribute Data Type:	date [other date-type attributes]
Attribute Default Value:	(calculated)
Attribute Group:	Net [other Net Group attributes]
Attribute Purpose:	Calculated data
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	No
Attribute Read-Only?	Yes [other read-only attributes]
Attribute Intrinsic?	Yes [other intrinsic attributes]
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

Date when the current note was last updated via auto-fetch from the web (read-only).

Latitude

Attribute Data Type:	number [other number-type attributes]
Attribute Default Value:	(not set - 0)
Attribute Group:	Places [other Places Group attributes]
Attribute Purpose:	Locational data
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	No
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

This holds the latitude of the $Address sourced via Automatic Geocoding.
Also see $Longitude.

LeafBase

Attribute Data Type:	number [other number-type attributes]
Attribute Default Value:	0.5
Attribute Group:	Map [other Map Group attributes]
Attribute Purpose:	Experimental
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	No
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

Deprecated: Use $Base instead.
Sets visual style of the base 'leaf' and 'banner' shapes.
The valid range is 0-1.
Now deprecated in favour of $Base.

LeafBend

Attribute Data Type:	number [other number-type attributes]
Attribute Default Value:	0
Attribute Group:	Map [other Map Group attributes]
Attribute Purpose:	Experimental
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	No
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

Deprecated: Use $Bend instead.

Sets visual style of the bend on 'leaf' and 'banner' shapes.

The valid range is -1 to 1.

Now deprecated in favour of $Bend.

LeafDirection

Attribute Data Type:	boolean [other boolean-type attributes]
Attribute Default Value:	false
Attribute Group:	Map [other Map Group attributes]
Attribute Purpose:	Experimental
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	No
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

Deprecated: Use $Direction instead.

Sets the orientation of some shapes.

Now deprecated in favour of $Direction.

LeafTip

Attribute Data Type:	number [other number-type attributes]
Attribute Default Value:	0.5
Attribute Group:	Map [other Map Group attributes]
Attribute Purpose:	Experimental
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	No
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

Deprecated: Use $Tip instead.

Sets visual style of the tip 'leaf' and 'banner' shapes.

The valid range is 0–1.

Now deprecated in favour of $Tip.

LeftMargin

Attribute Data Type:	string [other string-type attributes]
Attribute Default Value:	2
Attribute Group:	TextFormat [other TextFormat Group attributes]
Attribute Purpose:	Note text
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	No
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

Legacy-only feature to set left margin.

$LeftMargin and $RightMargin are legacy features for pre-v6 and allow the user to set margins for individual notes.

Below is for legacy reference only…

Margins are set in multiples of 9 pixels; a left margin of 4 is equivalent to 36 pixels (0.5" or about 12.5mm). The default may be set in the System Attribute Inspector.

When copying and pasting note text between TBX files with different margin settings, the pasted text takes the receiving note's margin settings but the note window must be closed and re-opened for this fact to be correctly displayed on screen. See also Pasting and Text Margins.

The left and right margins are always the same.

LineSpacing

Attribute Data Type:	number [other number-type attributes]
Attribute Default Value:	100
Attribute Group:	TextFormat [other TextFormat Group attributes]
Attribute Purpose:	Note text
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	Yes
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

Space between lines within a paragraph of a note.

To adjust the space between paragraphs, see $ParagraphSpacing.

$LineSpacing is expressed as a *percentage* of the line's nominal or recommended spacing. The default value of 100 gives "standard" line space, while a value of 200 results in "double spacing", i.e. "x2".

Dense text blocks may prove more readable with a $LineSpacing of 125 or 150 than with the familiar setting of 100.

$LineSpacing can be set via the Text Inspector ▸ Text tab, 'Line spacing' control.

LocalAttributes

Attribute Data Type:	set [other set-type attributes]
Attribute Default Value:	(not set - empty string)
Attribute Group:	General [other General Group attributes]
Attribute Purpose:	Calculated data
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	No
Attribute Read-Only?	Yes [other read-only attributes]
Attribute Intrinsic?	Yes [other intrinsic attributes]
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

A set of the attributes for this note which have a local value—a value that is not inherited to taken from the default. Attributes that are computed (like $WordCount) or intrinsic (like $Xpos) are never local attributes.

Lock

Attribute Data Type:	boolean [other boolean-type attributes]
Attribute Default Value:	false
Attribute Group:	Map [other Map Group attributes]
Attribute Purpose:	Map configuration
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	Yes
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

Locks a note or (more usually) an adornment to the Map to avoid being moved inadvertently.
Behaviour of this and the associated $ReadOnly attribute are discussed under Non-editable notes.

Longitude

Attribute Data Type:	number [other number-type attributes]
Attribute Default Value:	(not set - 0)
Attribute Group:	Places [other Places Group attributes]
Attribute Purpose:	Locational data
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	No
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

This holds the longitude of $Address sourced via Automatic Geocoding.

MapBackgroundAccentColor

Attribute Data Type:	color [other color-type attributes]
Attribute Default Value:	gray
Attribute Group:	Map [other Map Group attributes]
Attribute Purpose:	Map configuration
Attribute Inherited from Preferences?	Yes
Attribute UI-configurable?	Yes
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

Accent colour used for Map view background patterns.
This replaces (and maps old values of) $MapBackgroundColor2; the latter is deprecated.
The attribute's default is not set but it inherits the default of `gray` (a buff colour) from the Preferences Maps pane; the default is he same as that for $MapBackgroundColor.
$MapBackgroundAccentColor can be set via a map view tab's Map Settings pop-over, 'Accent' colour controls.

MapBackgroundColor

Attribute Data Type:	color [other color-type attributes]
Attribute Default Value:	gray
Attribute Group:	Map [other Map Group attributes]
Attribute Purpose:	Map configuration
Attribute Inherited from Preferences?	Yes
Attribute UI-configurable?	Yes
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

Colour of the background in Map views.
This colour is in fact used for for the background of all main views (Outline, Explorer, Chart, Treemap).
The attribute's default is not set but it inherits the default of `gray` (a buff colour) from the Preferences Maps pane.
A number of utility views also use $MapBackgroundColor (the document-level value, not the current note's) to draw the background of their list panes.
In Map views it is also used as the primary colour for background patterns.
$MapBackgroundColor can be set via a map view tab's Map Settings pop-over, 'Color' colour controls.

MapBackgroundColor2

Attribute Data Type:	color [other color-type attributes]
Attribute Default Value:	gray
Attribute Group:	Map [other Map Group attributes]
Attribute Purpose:	Map configuration
Attribute Inherited from Preferences?	Yes
Attribute UI-configurable?	Yes
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

Accent colour used for Map view background patterns (now called MapBackgroundAccentColor).
This attribute maps to $MapBackgroundAccentColor, and is deprecated in favour of the latter.
The attribute's default is not set but it inherits the default of `gray` (a buff colour) from the Preferences Maps pane; the default is he same as that for $MapBackgroundColor.
$MapBackgroundColor2 can be set via a map view tab's Map Settings pop-over, 'Accent' colour controls.

MapBackgroundFill

Attribute Data Type:	string [other string-type attributes]
Attribute Default Value:	(not set - empty string)
Attribute Group:	Map [other Map Group attributes]
Attribute Purpose:	Map configuration
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	Yes
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

The optional fill for the Map view's background.
$MapBackgroundFill can be set via a map view tab's Map Settings pop-over, 'Fill' pop-up control.
Default fill options are listed in the Fills pop-up menu, along with any custom fills.

MapBackgroundFillOpacity

Attribute Data Type:	number [other number-type attributes]
Attribute Default Value:	0.1
Attribute Group:	Map [other Map Group attributes]
Attribute Purpose:	Map configuration
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	Yes
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

Controls opacity used to draw the fill for the Map view's background.
The fill is set via $MapBackgroundFill
$MapBackgroundFillOpacity can be set via a map view tab's Map Settings pop-over, 'Opacity' slider control.

MapBackgroundPattern

Attribute Data Type:	string [other string-type attributes]
Attribute Default Value:	plain
Attribute Group:	Map [other Map Group attributes]
Attribute Purpose:	Map configuration
Attribute Inherited from Preferences?	Yes
Attribute UI-configurable?	Yes
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

This holds the pattern type for Map view background patterns.
The default "plain", is inherited from a Preference on the Maps pane.
$MapBackgroundPattern can be set via a map view tab's Map Settings pop-over.

MapBackgroundShadow

Attribute Data Type:	boolean [other boolean-type attributes]
Attribute Default Value:	true
Attribute Group:	Map [other Map Group attributes]
Attribute Purpose:	Map configuration
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	No
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

Controls whether map and outline views have a slight left margin shadow.
If `true` the shadow is drawn.

MapBodyTextColor

Attribute Data Type:	color [other color-type attributes]
Attribute Default Value:	automatic
Attribute Group:	Map [other Map Group attributes]
Attribute Purpose:	Map item text configuration
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	No
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

Colour of the note's body text when displayed in a map view's note icon.
The colour can be specified as a named Tinderbox colour or as a hex colour, i.e. 'black' or '#000000'.
Default: `automatic`. This results in either a black or white colour depending on the value of $Color.
The note's map icon's title colour is set via $NameColor.

MapBodyTextSize

Attribute Data Type:	number [other number-type attributes]
Attribute Default Value:	0
Attribute Group:	Map [other Map Group attributes]
Attribute Purpose:	Map item text configuration
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	No
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

Specific font size for the body text displayed within map icons (rectangular icons only).
From v8, the default is about 20% larger than in previous versions.
Some values have specific meanings:

- 0 (zero). The default value. The app chooses the text size automatically, based on scale vs. the size of the title.
- 1. This value suppresses display of body text, even if present.

For all other values, i.e. 2 or greater, use a point size value to choose an explicit font size, i.e. value of 12 will render the body text as 12 point.
Body text is never displayed in:

- adornments.
- shaped notes.

Any negative value also suppresses display of $Text in the map icon.

MapNameSize

Attribute Data Type:	number [other number-type attributes]
Attribute Default Value:	100
Attribute Group:	Map [other Map Group attributes]
Attribute Purpose:	Map item text configuration
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	Yes
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

Adjust the relative size of note title ($Name) text in the Map view.
The base font size of outline items (usually 14pt) is set in the Maps tab of Document Settings, or via the Text Inspector's Title tab (allowed values are described there)
This is a rename/replacement of $MapTextSize, as map view icons feature several different text sources.
Similarly, $OutlineTextSize has been renamed $OutlineNameSize.
The old attribute names are deprecated but will continue to function.

MapPrototypeColor

Attribute Data Type:	color [other color-type attributes]
Attribute Default Value:	#cceecc
Attribute Group:	Outline [other Outline Group attributes]
Attribute Purpose:	Outline configuration
Attribute Inherited from Preferences?	Yes
Attribute UI-configurable?	No
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

Deprecated: see $PrototypeHighlightColor
Not used in v6, retained for legacy compatibility.
Colour of the circle shown behind the Outline view note icons of prototype notes. Deprecated in favour of $PrototypeHighlightColor which is a clearer description of purpose.
The default is very light: if prototypes do not show a mark behind them try a darker value for this attribute. To avoid the mark showing, set the attribute the same as $MapBackgroundColor.
The default value of `#cceecc` is inherited from the Document Settings Outlines tab's Prototype highlight colour.

MapScrollX

Attribute Data Type:	number [other number-type attributes]
Attribute Default Value:	0
Attribute Group:	Map [other Map Group attributes]
Attribute Purpose:	Map configuration
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	No
Attribute Read-Only?	No
Attribute Intrinsic?	Yes [other intrinsic attributes]
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

The X-axis scroll position of child map in a container viewport.
The value is in map units. These co-ordinates are floating-point numbers, and use the same relative co-ordinates as $Xpos and $Ypos.
For $MapScrollX, position '0' is when the scroll bar is in the middle of the view's vertical window scroller.
In the parent's viewport $MapScrollX is drawn halfway across the interior of the container's icon (i.e. $Width/2) though depending on $TitleHeight the position itself may be obscured.
If $MapScrollX/Y are {0,0}, then a note where `$Xpos=0;$Ypos=0` will be around the middle of the screen.
This attribute works in conjunction with $MapScrollY.
The attribute can be set via actions so it is possible to pre-set a particular area of the parent map to be set to be onscreen when opened.

MapScrollY

Attribute Data Type:	number [other number-type attributes]
Attribute Default Value:	0
Attribute Group:	Map [other Map Group attributes]
Attribute Purpose:	Map configuration
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	No
Attribute Read-Only?	No
Attribute Intrinsic?	Yes [other intrinsic attributes]
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

The Y-axis scroll position of the child map in a container viewport.
The value is in map units. These co-ordinates are floating-point numbers, and use the same relative co-ordinates as $Xpos and $Ypos.
For $MapScrollY, position '0' is when the scroll bar is in the middle of the view's horizontal window scroller.
In the parent's viewport $MapScrollY is drawn halfway down the interior of the container's icon (i.e. $Height/2) though depending on $TitleHeight the position itself may be obscured.
If $MapScrollX/Y are {0,0}, then a note where `$Xpos=0;$Ypos=0` will be around the middle of the screen.
This attribute works in conjunction with $MapScrollX.
The attribute can be set via actions so it is possible to pre-set a particular area of the parent map to be set to be onscreen when opened.

MapTextSize

Attribute Data Type:	number [other number-type attributes]
Attribute Default Value:	100
Attribute Group:	Map [other Map Group attributes]
Attribute Purpose:	Map item text configuration
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	No
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

Deprecated: use $MapNameSize instead.
Adjust the relative size of note title ($Name) text in the Map view.
The base font size of outline items (usually 14pt) is set in the Maps tab of Document Settings, or via the Text Inspector's Title tab (allowed values are described there)

Modified

Attribute Data Type:	date [other date-type attributes]
Attribute Default Value:	(calculated)
Attribute Group:	General [other General Group attributes]
Attribute Purpose:	Calculated data
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	No
Attribute Read-Only?	Yes [other read-only attributes]
Attribute Intrinsic?	Yes [other intrinsic attributes]
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

Returns the time a note's text was last modified (read-only).
See also the $Created attribute: at first creation of the note both are the same.
This attribute tracks changes to:

- $Text.
- the value of any attribute value in the displayed attributes table or in Get Info: attributes

… with such changes updating $Modified.
Be aware that changing $Text or an attribute value in a stamp (including Quickstamp) or other form of action does **not** update $Modified.

mt_allow_comments

Attribute Data Type:	number [other number-type attributes]
Attribute Default Value:	1
Attribute Group:	Weblog [other Weblog Group attributes]
Attribute Purpose:	Weblog configuration
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	No
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

Deprecated: retained for pre-v6 use only.
Not used in v6+. Specifies, if 1, that MovableType weblogs allow comments.

mt_allow_pings

Attribute Data Type:	number [other number-type attributes]
Attribute Default Value:	1
Attribute Group:	Weblog [other Weblog Group attributes]
Attribute Purpose:	Weblog configuration
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	No
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

Deprecated: retained for pre-v6 use only.
Not used in v6+. Specifies, if 1, that MovableType weblogs allow pings to weblogs.com.

mt_convert_breaks

Attribute Data Type:	string [other string-type attributes]
Attribute Default Value:	(not set - empty string)
Attribute Group:	Weblog [other Weblog Group attributes]
Attribute Purpose:	Weblog configuration
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	No
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

Deprecated: retained for pre-v6 use only.
Not used in v6+. Specifies, if 1, that MovableType adds its own markup to discover paragraph breaks.

mt_keywords

Attribute Data Type:	string [other string-type attributes]
Attribute Default Value:	(not set - empty string)
Attribute Group:	Weblog [other Weblog Group attributes]
Attribute Purpose:	Weblog configuration
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	No
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

Deprecated: retained for pre-v6 use only.
Not used in v6+. Allows the user to specify one or more categories for the current note for use in MoveableType and related weblogs.

MyBoolean

Attribute Data Type:	boolean [other boolean-type attributes]
Attribute Default Value:	false
Attribute Group:	Sandbox [other Sandbox Group attributes]
Attribute Purpose:	Testing
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	No
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

Sample Boolean-type attribute for experimentation with Action code.
More on Boolean-type attributes.

MyColor

Attribute Data Type:	color [other color-type attributes]
Attribute Default Value:	(not set - coerced to #000000)
Attribute Group:	Sandbox [other Sandbox Group attributes]
Attribute Purpose:	Testing
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	No
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

Sample Color-type attribute for experimentation with Action code.
More on Color-type attributes.

MyDate

Attribute Data Type:	date [other date-type attributes]
Attribute Default Value:	(not set - never)
Attribute Group:	Sandbox [other Sandbox Group attributes]
Attribute Purpose:	Testing
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	No
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

Sample Date-type attribute for experimentation with Action code.
More on Date-type attributes.

MyDictionary

Attribute Data Type:	dictionary [other dictionary-type attributes]
Attribute Default Value:	(not set - empty string)
Attribute Group:	Sandbox [other Sandbox Group attributes]
Attribute Purpose:	Testing
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	No
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

Sample Dictionary-type attribute for experimentation with Action code.
More on Dictionary-type attributes.

MyInterval

Attribute Data Type:	interval [other interval-type attributes]
Attribute Default Value:	00:00
Attribute Group:	Sandbox [other Sandbox Group attributes]
Attribute Purpose:	Testing
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	No
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

Sample Interval-type attribute for experimentation with Action code.
More on Interval-type attributes.

MyList

Attribute Data Type:	list [other list-type attributes]
Attribute Default Value:	(not set - empty string)
Attribute Group:	Sandbox [other Sandbox Group attributes]
Attribute Purpose:	Testing
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	No
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

Sample List-type attribute for experimentation with Action code.
More on List-type attributes.

MyNumber

Attribute Data Type:	number [other number-type attributes]
Attribute Default Value:	(not set - 0)
Attribute Group:	Sandbox [other Sandbox Group attributes]
Attribute Purpose:	Testing
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	No
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

Sample Number-type attribute for experimentation with Action code.

More on Number-type attributes.

MySet

Attribute Data Type:	set [other set-type attributes]
Attribute Default Value:	(not set - empty string)
Attribute Group:	Sandbox [other Sandbox Group attributes]
Attribute Purpose:	Testing
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	No
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

Sample Set-type attribute for experimentation with Action code.

More on Set-type attributes.

MyString

Attribute Data Type:	string [other string-type attributes]
Attribute Default Value:	(not set - empty string)
Attribute Group:	Sandbox [other Sandbox Group attributes]
Attribute Purpose:	Testing
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	No
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

Sample String-type attribute for experimentation with Action code.

More on String-type attributes.

Name

Attribute Data Type:	string [other string-type attributes]
Attribute Default Value:	untitled
Attribute Group:	General [other General Group attributes]
Attribute Purpose:	Note title
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	Yes
Attribute Read-Only?	No
Attribute Intrinsic?	Yes [other intrinsic attributes]
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

Specifies the title of the note.

$Name is equivalent to the ^title^ export code in the context of exporting data. The $Name string can be one word or more and can include symbols etc., though the latter are not suggested for use if you're going to export to HTML as you may/will have issues to resolve with HTML entities in the output.

$Name of a selected object can easily be set/edited via the title box at the top of the text pane, or in some views by left-click-hold on the title text of the icon in the view pane.

Because Tinderbox recognises operators such as + and -, notes whose names begin with characters other than letters and numbers may sometimes be interpreted in unexpected ways.

For example, if a note is named "6*7", rules like

```
$Prototype=6*7
```

might be parsed as a multiplication with the result of 42.

```
$Prototype="6*7"
```

should have the expected effect.

The attributes $DisplayExpression and (read-only) $DisplayName allow the user the option of showing the original name or as a manipulated version, e.g. with some information such as a word count concatenated on the end. Use of the new features is non-intrusive to existing use such as in actions & rules as $Name always retains the name as before any expression is applied to it.

Does a note name need to be unique?

No, and $Path or in the case of both same-titled notes being in the same container, $ID can be used to reference discretely each duplicate-named note.

Is $Name really intrinsic?

If the intrinsic setting of $Name seems egregious, recall that intrinsic status reflects the XML setting ' canInherit'. When a note acts as prototype, it is generally desirable that the inheriting note has a different title, thus $Name is intrinsic.

However, $Name differs from general intrinsic behaviour in that it is always the same for the original note and all its aliases. In other words, it is not possible to set a different $Name for an alias, whereas it is possible to have a different $Height or $Width. As $DisplayExpression is evaluated in the current context which may differ for an original and alias (e.g. if in different containers) offering and opportunity for discrete $DisplayName labels using contextual inputs.

NameAlignment

Attribute Data Type:	string [other string-type attributes]
Attribute Default Value:	left
Attribute Group:	Map [other Map Group attributes]
Attribute Purpose:	Map item text configuration
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	Yes
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

Specifies the justification of the (title and subtitle) in notes and adornments.
Allowed values are:

- left
- center
- right

The attribute default is 'left'.
$NameAlignment can be set via the Text Inspector ▸ Title tab, 'Alignment' pop-up control.
Subtitles have no separate alignment control but instead inherit this setting.

NameBold

Attribute Data Type:	boolean [other boolean-type attributes]
Attribute Default Value:	false
Attribute Group:	Map [other Map Group attributes]
Attribute Purpose:	Map item text configuration
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	Yes
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

Controls use of bold type for the note name in map views (and all main views).
Changing the default value of $NameBold is convenient for bolding all note names.
$NameBold can be set via the Text Inspector ▸ Title tab, 'Bold' tick-box control.

NameColor

Attribute Data Type:	color [other color-type attributes]
Attribute Default Value:	automatic
Attribute Group:	Map [other Map Group attributes]
Attribute Purpose:	Map item text configuration
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	Yes
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

Colour used for the title of the note in Map views.
The colour can be specified as a named Tinderbox colour or as a hex colour, i.e. 'black' or '#000000'.
Default: automatic. This results in either a black or white colour depending on the value of $Color.
The colour of body text displayed in a map note icon is set via $MapBodyTextColor.
This colour may also 'transparent'. This option is intended to help those wishing to use images to form a note icon's contents in Map view. If the note title is transparent, when the note is selected, the note title will still be visible in the text pan
$NameColor can be set via the Text Inspector ▸ Title tab, 'Color' colour controls.

NameFont

Attribute Data Type:	font [other font-type attributes]
Attribute Default Value:	IdealSansSSm-Book
Attribute Group:	Map [other Map Group attributes]
Attribute Purpose:	Map item text configuration
Attribute Inherited from Preferences?	Yes
Attribute UI-configurable?	No
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

Font used for rendering the $Name of a note in the Map view.
It is also used to render the note name in most views: e.g. Outline, Chart, Treemap, Roadmap, Find, Path, Common Words, etc. Here, 'note' implies a note, a container note or an agent. Be aware that adornments may optionally use $AdornmentFont for the same purpose, enabling all adornments to use a different font as opposed to notes/agents.
The attribute's default is not set but is inherited as Ideal Sans (was Helvetica Neue pre-v7, Lucida Grande pre-v6, Geneva pre-v2.3.0) from the Preferences Maps pane's 'Note label font' option.

NameLeading

Attribute Data Type:	number [other number-type attributes]
Attribute Default Value:	0
Attribute Group:	Map [other Map Group attributes]
Attribute Purpose:	Map item text configuration
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	No
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

Controls leading between lines of a multi-line title display
This is pertinent where the title is long and my wrap.
Possible values:

- -1. Specifies that the space between lines should be chosen by the font ($NameFont). Negative values also affect leading in titles of Map view icons.
- 0. Specifies that the lines of the name should be set as tightly as possible.
- Any other value (greater than zero) adds additional space between lines. Positive values of $NameLeading represent a multiple of the natural line height. For example, $NameLeading of 1.5 adds a half-line of extra space between lines of the title, and $NameLeading of 0.9 sets the title 10% tighter than normal.

In outlines, the space between different titles is controlled by a Map Preference.

NameStrike

Attribute Data Type:	boolean [other boolean-type attributes]
Attribute Default Value:	false
Attribute Group:	Map [other Map Group attributes]
Attribute Purpose:	Map item text configuration
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	Yes
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

Render note title in strike-through.

If `true` the title is struck through.

More detail.

$NameStrike can be set via the Text Inspector ▸ Title tab, 'Strikethrough' tick-box control.

NeverComposite

Attribute Data Type:	boolean [other boolean-type attributes]
Attribute Default Value:	true
Attribute Group:	Composites [other Composites Group attributes]
Attribute Purpose:	Composite configuration
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	No
Attribute Read-Only?	No
Attribute Intrinsic?	Yes [other intrinsic attributes]
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

Exclude a note from all composites.

By setting the default value of $NeverComposite to `true`, composites can be turned off throughout a document.

In new documents, the default value of $NeverComposite is initially `true`. Prior to v9 the default was `false`. Pre-existing documents will retain their current default.

NLNames

Attribute Data Type:	set [other set-type attributes]
Attribute Default Value:	(not set - empty string)
Attribute Group:	AI [other AI Group attributes]
Attribute Purpose:	Natural Language Processing
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	No
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

A set of personal names found in the text using Natural Language Processing.

NLOrganizations

Attribute Data Type:	set [other set-type attributes]
Attribute Default Value:	(not set - empty string)
Attribute Group:	AI [other AI Group attributes]
Attribute Purpose:	Natural Language Processing
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	No
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

A set of the names of organisations found in the text using Natural Language Processing.

NLPlaces

Attribute Data Type:	set [other set-type attributes]
Attribute Default Value:	(not set - empty string)
Attribute Group:	AI [other AI Group attributes]
Attribute Purpose:	Natural Language Processing
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	No
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

A set of place names found in the text using Natural Language Processing.

NLTags

Attribute Data Type:	set [other set-type attributes]
Attribute Default Value:	(not set - empty string)
Attribute Group:	AI [other AI Group attributes]
Attribute Purpose:	Natural Language Processing
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	No
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

A set-type attribute to hold annotations automatically generated using Natural Language Processing.
The first such annotation adds the tag 'plan' to notes that Tinderbox believes might represent a planning note, such as "remember to deposit the cheque" or "remind the freshers to begin planning their module essays".

NoSpellling

Attribute Data Type:	boolean [other boolean-type attributes]
Attribute Default Value:	false
Attribute Group:	TextFormat [other TextFormat Group attributes]
Attribute Purpose:	Note text
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	Yes
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

Allows notes to be ignored during spell checks.
if a document is set to ' check spelling as you type', setting $NoSpelling to `true` will cause that note to be ignored.
Default is 'false', i.e. the note *is* included in spell checks to maintain previous behaviour.
$NoSpelling can be set for the current note via the Text Inspector ▸ Text tab, 'Bold' tick-box control.

NotesFolder

Attribute Data Type:	string [other string-type attributes]
Attribute Default Value:	(not set - empty string)
Attribute Group:	Watch [other Watch Group attributes]
Attribute Purpose:	Data synch
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	No
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

The name of an Apple Notes folder used by a watched folder.
Individual synced Notes within the watched folder use $NotesID and $NotesModified.

NotesID

Attribute Data Type:	string [other string-type attributes]
Attribute Default Value:	(not set - empty string)
Attribute Group:	Watch [other Watch Group attributes]
Attribute Purpose:	Data synch
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	No
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

A unique source ID of Notes imported via a Notes watched folder.
The source Notes folder is held in $NotesFolder.

NotesModified

Attribute Data Type:	date [other date-type attributes]
Attribute Default Value:	(not set - never)
Attribute Group:	Watch [other Watch Group attributes]
Attribute Purpose:	Data synch
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	No
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

The latest modification date/time of a Note imported/synced via a Notes or Finder watched folder.
The source Notes folder is held in $NotesFolder.

NoteURL

Attribute Data Type:	string [other string-type attributes]
Attribute Default Value:	(not set - empty string)
Attribute Group:	Net [other Net Group attributes]
Attribute Purpose:	Data synch
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	No
Attribute Read-Only?	Yes [other read-only attributes]
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

Pseudo-url (tinderbox://) URL for current note.

A read-only attribute exposing the pseudo-url tinderbox:// allowing access the current note from other programs.

This attribute is transferred when pasting notes to DEVONThink v2.8.8+.

OnAdd

Attribute Data Type:	action [other action-type attributes]
Attribute Default Value:	(not set - empty string)
Attribute Group:	General [other General Group attributes]
Attribute Purpose:	Action code
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	Yes
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

Action code expression executed when a note becomes child of the current note, be it new or moved there by dragging.

Adornments also offer 'OnAdd' action support. Although it takes a string argument, the string must be valid action syntax.

$OnAdd actions are applied to each item of a multiple-item drag, even if the dragged notes originate in a different file. The $OnAdd action is applied only once, when a note is created inside a container, or is moved into a container from some other container. This stops $OnAdd being re-applied for actions like moving a note on a Map and where $OnAdd may have iterative effects on a note if re-applied. Pasting a note into a container runs the $OnAdd action.

The container's $OnAdd action acts on both newly-made notes and agents.

For agents, $OnAdd is replaced by $AgentAction (though not in smart adornments)

Adornments show a slightly different behaviour. Here, the $OnAdd is re-triggered if a note is moved manually within the adornment, i.e. never leaves it, as the move effectively takes the note out of the adornment (start of move) and *adds* it back at the end of the move. However, if notes are moved automatically within an adornment due to adornment sorting, or by an external action altering $Xpos/$Ypos, the adornment's $OnAdd is **not** run again.

$OnAdd (or $AgentAction for agents) for the current note can be set via the Action Inspector ▸ Action tab, code input box.

OnJoin

Attribute Data Type:	action [other action-type attributes]
Attribute Default Value:	(not set - empty string)
Attribute Group:	Composites [other Composites Group attributes]
Attribute Purpose:	Composite configuration
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	No
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

An action performed when new composite members first touch existing members.

Notes may have an $OnJoin action. When a note that was not previously a member of a composite is dragged to touch a note with an $OnJoin action, the $OnJoin action is performed on the dragged note. If a note touches more than one not in the composite, each $OnJoin action is performed in turn.

In the context of an $OnJoin action, designator ' this' references the note joining the composite whilst ' that' references the note within composite to which it is now adjacent.

OnRemove

Attribute Data Type:	action [other action-type attributes]
Attribute Default Value:	(not set - empty string)
Attribute Group:	General [other General Group attributes]
Attribute Purpose:	Action code
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	No
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

Action code expression executed when performed when a note is removed from this container.

The OnRemove action for agents is applied to the *original* note, since the alias will be deleted. Note, too, that agents may remove notes and add them again at any time.

When cutting a note, the OnRemove action of its container is applied *before* the note is copied to the pasteboard.

$OnRemove for the current note can be set via the Action Inspector ▸ Remove tab, code input box.

OnVisit

Attribute Data Type:	action [other action-type attributes]
Attribute Default Value:	(not set - empty string)
Attribute Group:	Storyspace [other Storyspace Group attributes]
Attribute Purpose:	Storyspace synch
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	No
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

In Storyspace, an action performed when the reader visits the note.

This is honoured in Tinderbox and can be used as an 'OnSelection' event trigger.

Opacity

Attribute Data Type:	number [other number-type attributes]
Attribute Default Value:	100
Attribute Group:	Appearance [other Appearance Group attributes]
Attribute Purpose:	Map item general configuration
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	No
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

Opacity of a note icon in map view.

The default opacity value of 100 leaves notes opaque, while an opacity of zero allows the note interior to be transparent.

Translucency may be useful in some maps with complex link networks, and also permits interesting visual effects.

N.B. Borders are not translucent.

Containers and agents are never translucent. If a note was translucent, later becomes a container, and later yet becomes a simple note once more, the note's $Opacity will again control its opacity.

Notes using some shapes—"cloud" and "bubble"—are drawn without borders when translucent.

The Interior inspector's, Pattern ▸ translucent option toggles an Opacity value of 50/100.

Both $TitleOpacity and $SubtitleOpacity are independent of $Opacity. However, the if higher than the value of $Opacity, the latter value is used when drawing the note.

Organization

Attribute Data Type:	string [other string-type attributes]
Attribute Default Value:	(not set - empty string)
Attribute Group:	People [other People Group attributes]
Attribute Purpose:	Person detail
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	No
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

Intended for storing the a person's company or organisation name.

OutboundLinkCount

Attribute Data Type:	number [other number-type attributes]
Attribute Default Value:	(calculated)
Attribute Group:	General [other General Group attributes]
Attribute Purpose:	Calculated data
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	No
Attribute Read-Only?	Yes [other read-only attributes]
Attribute Intrinsic?	Yes [other intrinsic attributes]
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

Number of outbound links originating from the current note (read-only).

The count includes both both outbound basic and text links. Discrete totals are also calculated for outbound basic links ($PlainLinkCount) and for outbound text links ($TextLinkCount). However, any Web links are counted completely separately in $WebLinkCount.

See also $InboundLinkCount.

The link count always excludes any links of type "Prototype".

OutlineBackgroundColor

Attribute Data Type:	color [other color-type attributes]
Attribute Default Value:	automatic
Attribute Group:	Outline [other Outline Group attributes]
Attribute Purpose:	Outline configuration
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	No
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

Colour used as background colour of the selected note(s) listing in Outline view.

The colour is carried across the width of the view including checkbox and column display areas.

With the default value of "automatic" the colour used is simply the same as $MapBackgroundColor; the inside outline checkboxes is always shown in grey.

OutlineColorSwatch

Attribute Data Type:	boolean [other boolean-type attributes]
Attribute Default Value:	false
Attribute Group:	Outline [other Outline Group attributes]
Attribute Purpose:	Outline configuration
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	No
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	9.5.2

Use a colour swatch in Outline view instead of a badge.

Controls whether a note in Outline View shows a colour swatch between the note icon and its title. If `true` the swatch is drawn in $Color.

It occupies the same space as a badge and is suppressed if a $Badge is set

This attribute enables outlines to show simple progress bars. If the latter pattern is used, the swatch is drawn in $Color and $AccentColor.

Note: this feature is deprecated from v9.5.2, in favour of using flags in outlines.

OutlineDepth

Attribute Data Type:	number [other number-type attributes]
Attribute Default Value:	(calculated)
Attribute Group:	General [other General Group attributes]
Attribute Purpose:	Calculated data
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	No
Attribute Read-Only?	Yes [other read-only attributes]
Attribute Intrinsic?	Yes [other intrinsic attributes]
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

Number of outline levels between a note and the document root (read-only).

Actually, it is 1 + the number as 'top-level' notes have $OutlineDepth of 1, notes inside a top-level container have $OutlineDepth of 2, and so forth.

Aliases do not inherit this property from their original but have their own intrinsic value.

OutlineNameSize

Attribute Data Type:	number [other number-type attributes]
Attribute Default Value:	100
Attribute Group:	Outline [other Outline Group attributes]
Attribute Purpose:	Outline configuration
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	Yes
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

Adjust the relative size of note title ($Name) text in Outline/Chart/Timeline views.

The base font size of outline items (usually 14pt) is set in the Maps tab of Document Settings, or via the Text Inspector's Title tab (allowed values are described there).

This setting does not affect the line weight of separators, only the text height of the line they occupy (to allow for a text label even if not used).

This is a rename/replacement of $OutlineTextSize.

The rename mirrors the renaming of $MapTextSize to $MapNameSize.

The old names are deprecated but will continue to function.

OutlineOrder

Attribute Data Type:	number [other number-type attributes]
Attribute Default Value:	(calculated)
Attribute Group:	General [other General Group attributes]
Attribute Purpose:	Calculated data
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	No
Attribute Read-Only?	Yes [other read-only attributes]
Attribute Intrinsic?	Yes [other intrinsic attributes]
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

Sequential position in the overall document hierarchy (read-only).

It can be used by agents to judge the position of a note within the document. Note that some items not displayed in the Outline view, such as adornments, are still included in the $OutlineOrder. This should be borne in mind if doing calculations based on comparing sibling $OutlineOrder values as the 'next' item may not be a note (and there is no way to test this fact).

The order of notes within the current container is shown by $SiblingOrder.

The lowest number is drawn at the front in a map. If notes with outline orders #1 and #5 overlap, part of #5 will be behind the icon for #1.

The note menu command to move a note forward/backward or all the way to the back/front will alter both $OutlineOrder and $SiblingOrder for the affected note(s).

Aliases do not inherit this property from their original but have their own value.

OutlineTextSize

Attribute Data Type:	number [other number-type attributes]
Attribute Default Value:	100
Attribute Group:	Outline [other Outline Group attributes]
Attribute Purpose:	Outline configuration
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	No
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

Deprecated: use $OutlineTextSize instead.

Adjust the relative size of note title ($Name) text in Outline/Chart/Timeline views.

The base font size of outline items (usually 14pt) is set in the Maps tab of Document Settings, or via the Text Inspector's Title tab (allowed values are described there).

This setting does not affect the line weight of separators, only the text height of the line they occupy (to allow for a text label even if not used).

Pages

Attribute Data Type:	string [other string-type attributes]
Attribute Default Value:	(not set - empty string)
Attribute Group:	References [other References Group attributes]
Attribute Purpose:	Data synch
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	No
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

Used for Bookends reference import. Maps to RIS data tag 'SP'.

ParagraphSpacing

Attribute Data Type:	string [other string-type attributes]
Attribute Default Value:	8
Attribute Group:	TextFormat [other TextFormat Group attributes]
Attribute Purpose:	Note text
Attribute Inherited from Preferences?	Yes
Attribute UI-configurable?	Yes
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

Sets paragraph spacing for individual notes.
The default value of $ParagraphSpacing is set in the Text pane of Document Settings.
$ParagraphSpacing can be set for the current note via the Text Inspector ▸ Text tab, 'Paragraph Spacing' control.

Participants

Attribute Data Type:	set [other set-type attributes]
Attribute Default Value:	(not set - empty string)
Attribute Group:	Events [other Events Group attributes]
Attribute Purpose:	
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	No
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	9.5.2
Attribute Last Altered:	As at baseline

A listing of an event's participants.
See Calendar event drag-drop.

Path

Attribute Data Type:	string [other string-type attributes]
Attribute Default Value:	(calculated)
Attribute Group:	General [other General Group attributes]
Attribute Purpose:	Calculated data
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	No
Attribute Read-Only?	Yes [other read-only attributes]
Attribute Intrinsic?	Yes [other intrinsic attributes]
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

Contains the path from the root of the document to the note (read-only).
For example, if the document contains the following notes:

```
To Do
    Bank
    Post Office
    Garden
        Weeding
Done
    Tax
```

then the $Path of note 'Weeding' would be:
`/To Do/Garden/Weeding`
Note that this is the full path, including the current notes name. A full path always starts with a '/' character.
$Path is identified as intrinsic.

Pattern

Attribute Data Type:	string [other string-type attributes]
Attribute Default Value:	plain
Attribute Group:	Appearance [other Appearance Group attributes]
Attribute Purpose:	Map item general configuration
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	Yes
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

Optional pattern to be drawn on the face of icons in Map view.
$Pattern may be set for the current note via the Appearance Inspector ▸ Interior tab, 'Pattern' control: for plot-style $Pattern values us the Appearance Inspector ▸ Plot tab, 'Pattern' control instead.
These patterns look best when the two colours are similar but not identical. The keywords are *case-sensitive*. The current default is '**plain**'.
If $Pattern is **plain** no pattern is applied and the note face is drawn in solid $Color.
If $Pattern is **lines**, the face of the note is filled with alternative horizontal lines of $Color and $AccentColor.
If $Pattern is **gradient**, the face of the note is filled with a linear colour gradient, top to bottom, from $Color to $AccentColor.
If $Pattern is **diagonal**, the face of the note is filled with a linear colour gradient, running from upper-left to lower-right corner, from $Color to $AccentColor.
If $Pattern is **cylinder**, the face of the note is filled with a cylindrical gradient, running from $Color at the top through $AccentColor at the midpoint, and then to $Color again at the bottom.
If $Pattern is **radial**, the face of the note is a graduation from $Color at 1/3 in from top-left through $AccentColor toward all edges.
If $Pattern is not set (empty string), no pattern is applied and equating to **plain**.
The remaining seven patterns cannot be set via the Pattern menu. Rather they are set via manual editing of this attribute (Info view, Displayed Attributes) or via actions, rules, stampstc., acting upon it. These patterns are:

- bar(). The draws as a horizontal 'progress bar', using $Color and $AccentColor. The 'progress' block is drawn in $AccentColor.
- vbar(). The draws as a vertical 'progress bar', using $Color and $AccentColor. The 'progress' block is drawn in $AccentColor.
- 5 patterns only used for Map view container plots:
  - bargraph().
  - plot().
  - xyplot().
  - ring().
  - pie().

PlainLinkCount

Attribute Data Type:	number [other number-type attributes]
Attribute Default Value:	(calculated)
Attribute Group:	General [other General Group attributes]
Attribute Purpose:	Calculated data
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	No
Attribute Read-Only?	Yes [other read-only attributes]
Attribute Intrinsic?	Yes [other intrinsic attributes]
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

Holds the number of outbound basic links.

Also see $TextLinkCount, $WebLinkCount, $OutboundLinkCount.

PlotBackgroundColor

Attribute Data Type:	color [other color-type attributes]
Attribute Default Value:	white
Attribute Group:	Map [other Map Group attributes]
Attribute Purpose:	Map configuration
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	Yes
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

Colour of the plot background when a map container plot is configured.

If the container has a pattern such as plot/bargraph/xyplot), before the graph is drawn the space behind the plot is filled with the colour $PlotBackgroundColor with opacity of $PlotBackgroundOpacity. Previously the background was that of the child map.

$PlotBackgroundColor for the current note may be set via the Appearance Inspector ▸ Plot tab, 'Background' colour controls.

PlotBackgroundOpacity

Attribute Data Type:	number [other number-type attributes]
Attribute Default Value:	50
Attribute Group:	Map [other Map Group attributes]
Attribute Purpose:	Map configuration
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	Yes
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

Opacity of background colour to the plot when a map container plot is configured.

If the container has a pattern such as plot/bargraph/xyplot), before the graph is drawn, the space behind the plot is filled with the colour $PlotBackgroundColor with opacity of $PlotBackgroundOpacity. Previously the background was that of the child map.

$PlotBackgroundOpacity for the current note may be set via the Appearance Inspector ▸ Plot tab, 'Background' slider control.

PlotColor

Attribute Data Type:	color [other color-type attributes]
Attribute Default Value:	light green
Attribute Group:	Appearance [other Appearance Group attributes]
Attribute Purpose:	Map item general configuration
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	Yes
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

Colour used to draw plots in containers.

$PlotColor for the current note may be set via the Appearance Inspector ▸ Plot tab, 'Plot Color' colour controls.

PlotColorList

Attribute Data Type:	list [other list-type attributes]
Attribute Default Value:	2;7;3;8;4;9;0;5;1;6
Attribute Group:	Map [other Map Group attributes]
Attribute Purpose:	Map item general configuration
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	No
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

A list of colours to be used in this note's pie charts.

The default value is the list `[2;7;3;8;4;9;0;5;1;6]`, using the default built-in numbered greyscale colours. The first colour designates the colour of the first segment, and remaining colours in the list are used in rotation until all segments have been drawn. If the list contains fewer than two colours, the list `[black;white]` is used instead.

As a List-type attribute, this can take a list of values of any defined system/user colour names, such as might be generated by `collect(children,$Color)`.

PostalCode

Attribute Data Type: string [other string-type attributes]
Attribute Default Value: (not set - empty string)
Attribute Group: Places [other Places Group attributes]
Attribute Purpose: Locational data
Attribute Inherited from Preferences? No
Attribute UI-configurable? No
Attribute Read-Only? No
Attribute Intrinsic? No
Attribute First Added: Baseline
Attribute Last Altered: As at baseline

The Postal Code field of an address.

The automatic geocoding tries to supply these fields automatically when looking up the supplied $Address.

The date type is a string as not all countries use number-only postal/zip codes.

PosterCSS

Attribute Data Type: string [other string-type attributes]
Attribute Default Value: (not set - empty string)
Attribute Group: Poster [other Poster Group attributes]
Attribute Purpose: Poster configuration
Attribute Inherited from Preferences? No
Attribute UI-configurable? No
Attribute Read-Only? No
Attribute Intrinsic? No
Attribute First Added: 9.6.0
Attribute Last Altered: As at baseline

Holds CSS rules to apply to the poster.

New to v9.6.0, this holds CSS rules that are applied to the note's poster, whether generated via a template or called via a URL.

PosterLabels

Attribute Data Type: list [other list-type attributes]
Attribute Default Value: (not set - empty string)
Attribute Group: Poster [other Poster Group attributes]
Attribute Purpose: Poster configuration
Attribute Inherited from Preferences? No
Attribute UI-configurable? No
Attribute Read-Only? No
Attribute Intrinsic? No
Attribute First Added: 9.6.0
Attribute Last Altered: As at baseline

A utility attribute for passing information to a poster's visualisation tool.

New to v9.6.0, this is an assistive option for poster configuration and has no special significance to Tinderbox.

PosterSettings

Attribute Data Type: string [other string-type attributes]
Attribute Default Value: (not set - empty string)
Attribute Group: Poster [other Poster Group attributes]
Attribute Purpose: Poster configuration
Attribute Inherited from Preferences? No
Attribute UI-configurable? No
Attribute Read-Only? No
Attribute Intrinsic? No
Attribute First Added: 9.6.0
Attribute Last Altered: As at baseline

A utility attribute for passing information to a poster's visualisation tool.

New to v9.6.0, this is an assistive option for poster configuration and has no special significance to Tinderbox.

PosterTemplate

Attribute Data Type: string [other string-type attributes]
Attribute Default Value: (not set - empty string)
Attribute Group: Poster [other Poster Group attributes]
Attribute Purpose: Poster configuration
Attribute Inherited from Preferences? No
Attribute UI-configurable? No
Attribute Read-Only? No
Attribute Intrinsic? No
Attribute First Added: 9.6.0
Attribute Last Altered: As at baseline

The template note used to define a poster.

New to v9.6.0, $PosterTemplate specifies a template note, much as $HTMLExportTemplate does. This template is applied to the note, and the result is displayed on the poster affixed to the note.

PosterURL

Attribute Data Type:	URL [other URL-type attributes]
Attribute Default Value:	(not set - empty string)
Attribute Group:	Poster [other Poster Group attributes]
Attribute Purpose:	Poster configuration
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	No
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	9.6.0
Attribute Last Altered:	As at baseline

The URL used to call a poster visualisation

New to v9.6.0, $PosterURL specifies a URL to be displayed in the current note's poster.

The expectation is that $PosterURL will be much less common than $PosterTemplate. If a note has *both* a $PosterTemplate and a $PosterURL, the $PosterURL is ignored.

PosterX

Attribute Data Type:	list [other list-type attributes]
Attribute Default Value:	(not set - empty string)
Attribute Group:	Poster [other Poster Group attributes]
Attribute Purpose:	Poster configuration
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	No
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	9.6.0
Attribute Last Altered:	As at baseline

A utility attribute for passing information to a poster's visualisation tool.

New to v9.6.0, this is an assistive option for poster configuration and has no special significance to Tinderbox.

PosterY

Attribute Data Type:	list [other list-type attributes]
Attribute Default Value:	(not set - empty string)
Attribute Group:	Poster [other Poster Group attributes]
Attribute Purpose:	Poster configuration
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	No
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	9.6.0
Attribute Last Altered:	As at baseline

A utility attribute for passing information to a poster's visualisation tool.

New to v9.6.0, this is an assistive option for poster configuration and has no special significance to Tinderbox.

PosterZoom

Attribute Data Type:	number [other number-type attributes]
Attribute Default Value:	1.0
Attribute Group:	Poster [other Poster Group attributes]
Attribute Purpose:	Poster configuration
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	No
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	9.6.0
Attribute Last Altered:	As at baseline

A zoom value applied to posters sourced via a URL

Default value is `1.0`.

New to v9.6.0, $PosterZoom is a numeric attribute representing the magnification applied to posters loaded through $PosterURL. Its default value is 1.0, representing the page's normal size. A $PosterZoom of 0.5 would display the source page at half original screen size.

Private

Attribute Data Type:	boolean [other boolean-type attributes]
Attribute Default Value:	false
Attribute Group:	General [other General Group attributes]
Attribute Purpose:	Special note type designator
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	No
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

Controls whether a prototype appears in the Map view prototype tab menu or the Outline view pop-up list.

When set as Private, the note is removed from the popup menu of the Properties Inspector's Prototype sub-tab. This lets TBX document authors reserve some prototypes for special or internal use.

Other uses of "private" may be added in the future.

Prototype

Attribute Data Type:	string [other string-type attributes]
Attribute Default Value:	(not set - empty string)
Attribute Group:	General [other General Group attributes]
Attribute Purpose:	General data
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	Yes
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

The name of the current note's prototype note (if there is one).
The value may be the name of another prototype.
$Prototype for the current note may be set via the Properties Inspector ▸ Prototype tab, 'Prototype' pop-up control.

PrototypeBequeathsChildren

Attribute Data Type:	boolean [other boolean-type attributes]
Attribute Default Value:	true
Attribute Group:	General [other General Group attributes]
Attribute Purpose:	Special note type designator
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	No
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

Controls whether a prototype's children are inherited by notes using that prototype.
Thus this attribute only has meaning for notes for which $IsPrototype is also set to `true`.
Children created by this method are not removed from inheriting notes if they subsequently cease to use the prototype.
By default (`true`) all new prototypes will bequeath children and all descendants.
For prototypes that do not/will not have children this attribute has no effect on general function. However, if you like to nest your prototypes, do set this attribute to `false`.
Do not bequeath children that are themselves prototypes. Instead, create separate prototypes *outside* the bequeathed layout and set the source notes in the latter to use a prototype.
More on prototypes and prototype bequeathal.

PrototypeHighlightColor

Attribute Data Type:	color [other color-type attributes]
Attribute Default Value:	automatic
Attribute Group:	Outline [other Outline Group attributes]
Attribute Purpose:	Outline configuration
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	Yes
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

Colour of the circle shown behind the Outline view note icons of prototype notes.
The default is very light: if prototypes do not show a mark behind them try a darker value for this attribute. To avoid the mark showing, set the attribute the same as $MapBackgroundColor.
The default value of `#cceecc` is inherited from the Document Setting's Outline tab's Prototype highlight colour setting.
This attribute is a direct replacement for the older $MapPrototypeColor attribute.

PublicationCity

Attribute Data Type:	string [other string-type attributes]
Attribute Default Value:	(not set - empty string)
Attribute Group:	References [other References Group attributes]
Attribute Purpose:	Data synch
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	No
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

Used for Bookends reference import. Maps to RIS data tag 'CY'.

PublicationYear

Attribute Data Type:	string [other string-type attributes]
Attribute Default Value:	(not set - empty string)
Attribute Group:	References [other References Group attributes]
Attribute Purpose:	Data synch
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	No
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

Used for Bookends reference import. Maps to RIS data tag 'PY'.

Publisher

Attribute Data Type:	string [other string-type attributes]
Attribute Default Value:	(not set - empty string)
Attribute Group:	References [other References Group attributes]
Attribute Purpose:	Data synch
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	No
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

Used for Bookends reference import. Maps to RIS data tag 'PB'.

Range

Attribute Data Type:	number [other number-type attributes]
Attribute Default Value:	0
Attribute Group:	Places [other Places Group attributes]
Attribute Purpose:	Locational data
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	No
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

This represents the approximate range of a geographic adornment in kilometres.
Number-type attribute, in the Places group

RawData

Attribute Data Type:	string [other string-type attributes]
Attribute Default Value:	(not set - empty string)
Attribute Group:	Net [other Net Group attributes]
Attribute Purpose:	Import configuration
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	No
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

An experimental attribute. It should be treated as an internal field for Tinderbox's private use.
It is listed here simply to discourage inquisitive tinkering. As an experimental attribute it may disappear, be renamed or take on a new purpose in later versions.
Purpose is experimental in support of RSS and AutoFetch to hold the raw data should the planned fetch fail.

ReadCount

Attribute Data Type:	number [other number-type attributes]
Attribute Default Value:	(calculated)
Attribute Group:	General [other General Group attributes]
Attribute Purpose:	Calculated data
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	No
Attribute Read-Only?	Yes [other read-only attributes]
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

Tracks the number of times that a note is opened (read-only).
The count is cumulative from the creation of the note.
This attribute is now deprecated and not created in v6+; if already present from use in older version, it is no longer updated. In older versions it tracked the opening of a note's text window.

ReadOnly

Attribute Data Type:	boolean [other boolean-type attributes]
Attribute Default Value:	false
Attribute Group:	Textual [other Textual Group attributes]
Attribute Purpose:	Note text
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	No
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

Controls UI-based editing of the text of a note ($Text) or any contents of the displayed attributes table.
Behaviour of this and the associated $Lock attribute are discussed under Non-editable notes.

ReferenceDictionary

Attribute Data Type:	dictionary [other dictionary-type attributes]
Attribute Default Value:	(not set - empty string)
Attribute Group:	References [other References Group attributes]
Attribute Purpose:	Data synch
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	No
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

Provides a dictionary of RIS information from imported references.

ReferenceRIS

Attribute Data Type:	string [other string-type attributes]
Attribute Default Value:	(not set - empty string)
Attribute Group:	References [other References Group attributes]
Attribute Purpose:	Data synch
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	No
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

Used for Bookends reference import. Holds to full RIS data supplied.
This is the case for dragged RIS files or for a reference dragged from Bookends and possible other reference manager apps.
See RIS import.

ReferenceTitle

Attribute Data Type:	string [other string-type attributes]
Attribute Default Value:	(not set - empty string)
Attribute Group:	References [other References Group attributes]
Attribute Purpose:	Data synch
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	No
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

Used for Bookends reference import. Maps to RIS data tag 'T1'
Note: maps 'T1' except for CHAP (Chapter) type references.

ReferenceURL

Attribute Data Type:	URL [other URL-type attributes]
Attribute Default Value:	(not set - empty string)
Attribute Group:	References [other References Group attributes]
Attribute Purpose:	Data synch
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	No
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

Used for Bookends reference import. Maps to RIS data tag 'UR'.

RefFormat

Attribute Data Type:	string [other string-type attributes]
Attribute Default Value:	(not set - empty string)
Attribute Group:	References [other References Group attributes]
Attribute Purpose:	Data synch
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	No
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

Used for Bookends reference import.

RefKeywords

Attribute Data Type:	set [other set-type attributes]
Attribute Default Value:	(not set - empty string)
Attribute Group:	References [other References Group attributes]
Attribute Purpose:	Data synch
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	No
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

Used for Bookends reference import. Maps to RIS data tag 'KW'.

RefType

Attribute Data Type:	string [other string-type attributes]
Attribute Default Value:	(not set - empty string)
Attribute Group:	References [other References Group attributes]
Attribute Purpose:	Data synch
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	No
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

Used for Bookends reference import. Maps to value of RIS data tag 'TY'.
See RIS import.

Requirements

Attribute Data Type:	action [other action-type attributes]
Attribute Default Value:	(not set - empty string)
Attribute Group:	Storyspace [other Storyspace Group attributes]
Attribute Purpose:	Storyspace compatibility
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	No
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

Storyspace: writing space entry logic
In Storyspace, this holds the requirement logic to be met before a writing space can be visited in read mode.
Evaluating $Requirements tests if the desired linked space can be accessed. If the test is evaluates `false`, the link is not followed, even if the links guard field is satisfied.
This logic is **not** tested in Tinderbox.

ResetAction

Attribute Data Type:	action [other action-type attributes]
Attribute Default Value:	(not set - empty string)
Attribute Group:	Storyspace [other Storyspace Group attributes]
Attribute Purpose:	Storyspace compatibility
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	No
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

Storyspace: reset action.
In Storyspace, the action to be evaluated if the user selects the cover sheet in read mode.
This logic is **not** tested in Tinderbox.

RightMargin

Attribute Data Type:	string [other string-type attributes]
Attribute Default Value:	2
Attribute Group:	TextFormat [other TextFormat Group attributes]
Attribute Purpose:	Note text
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	No
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

Legacy-only feature to set left margin.
$RightMargin and $LeftMargin are legacy features to allow the user to set margins for individual notes.
Note that the margins are set in multiples of 9 pixels; a left margin of 4 is equivalent to 36 pixels (0.5" or about 12.5mm).
Below is for legacy reference only…
As left and right margins are now the same and $RightMargin is ignored and retained only for legacy compatibility purposes.
Prior to v6.6.0, the default is set in the Text pane of the Preferences.
When copying and pasting note text between TBX files with different margin settings, the pasted text takes the receiving note's margin settings but the note window must be closed and re-opened for this fact to be correctly displayed on screen. See also Pasting and Text Margins.

Role

Attribute Data Type:	string [other string-type attributes]
Attribute Default Value:	(not set - empty string)
Attribute Group:	Composites [other Composites Group attributes]
Attribute Purpose:	Composite configuration
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	No
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

The (optional) role within the composite of an individual note.
If a note has a $Role but not $Subtitle, the Role is displayed where the Subtitle would normally appear.

RSSChannelTemplate

Attribute Data Type:	file [other file-type attributes]
Attribute Default Value:	RSSChannel
Attribute Group:	Net [other Net Group attributes]
Attribute Purpose:	Import configuration
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	No
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

Specifies the name of the template used for RSS channels when importing ('AutoFetching') as RSS feed.
Not present in new documents, but it may be present in documents created on older application versions; its use, if present is deprecated.
The default template is stored in the Tinderbox application package (though not in its templates folder) and is:

```
^title^
^link^
^subtitle^
^description^
```

RSSItemLimit

Attribute Data Type:	number [other number-type attributes]
Attribute Default Value:	0
Attribute Group:	Net [other Net Group attributes]
Attribute Purpose:	Import configuration
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	No
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

Restrict the size of any feed being read into Tinderbox.
Not present in new documents, but it may be present in documents created on older application versions; its use, if present is deprecated.
Some news feeds syndicated a large number of items, and your design might have space for only a few.
The default value, zero, imposes no limit on the number of items imported.

RSSItemTemplate

Attribute Data Type:	file [other file-type attributes]
Attribute Default Value:	RSSItem
Attribute Group:	Net [other Net Group attributes]
Attribute Purpose:	Import configuration
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	No
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

Specifies the template used for RSS items with importing ('AutoFetching') an RSS feed.
Not present in new documents, but it may be present in documents created on older application versions; its use, if present is deprecated.
The default template is stored in the Tinderbox application package (though not in its templates folder) and is:

```
^title^
^link^
^description^
```

Rule

Attribute Data Type:	action [other action-type attributes]
Attribute Default Value:	(not set - empty string)
Attribute Group:	General [other General Group attributes]
Attribute Purpose:	Action code
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	Yes
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

The rule code that the note executes.
The string must be valid action syntax.
The rule can be overridden in the current note (i.e. still stored but not executed) by setting $RuleDisabled.
$Rule for the current note can be set via the Action Inspector ▸ Rule tab, code box.

RuleDisabled

Attribute Data Type:	boolean [other boolean-type attributes]
Attribute Default Value:	false
Attribute Group:	General [other General Group attributes]
Attribute Purpose:	Action code
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	Yes
Attribute Read-Only?	No
Attribute Intrinsic?	Yes [other intrinsic attributes]
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

Allows the $Rule for a specific note to be disabled. It is intrinsic and thus not inherited.

Atypically for inheritance in Tinderbox, it is **not** inherited from a note's prototype (unlike most other attributes), partly because a primary use is suppressing code running in prototypes.

$RuleDisabled is convenient when a rule may make significant changes to the note (or other parts of the document). Setting such a rule via a prototype can be problematic as the intent is only to rule to rule in notes inheriting the rule. Thus it can be useful to suppress $Rule activation in prototypes. Previously, this meant placing the Rule in an 'if($Prototype)' test which was cumbersome and meant every inheriting note had to run the addition if test adding load to the action code cycle. That workaround can be discarded and the attribute set to `true` in the Prototype. Because, atypically, this attribute is not inherited from a prototype, the prototype's setting does not affect the Display Expression in notes using the prototype.

A similar control is supplied for display expressions: $DisplayExpressionDisabled.

$RuleDisabled for the current note can be set via the Action Inspector ▸ Rule tab, 'enabled' tick-box.

ScreenHeight

Attribute Data Type:	number [other number-type attributes]
Attribute Default Value:	(not set - 0)
Attribute Group:	Poster [other Poster Group attributes]
Attribute Purpose:	Poster configuration
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	No
Attribute Read-Only?	Yes [other read-only attributes]
Attribute Intrinsic?	No
Attribute First Added:	9.6.0
Attribute Last Altered:	As at baseline

Records the poster's height in pixels to aid template configuration

The default value is `0`, and is *read-only*.

Added in v9.6.0, this attribute aids in configuring the HTML in poster templates. The pixel value is for map view at the magnification currently in use.

ScreenWidth

Attribute Data Type:	number [other number-type attributes]
Attribute Default Value:	(not set - 0)
Attribute Group:	Poster [other Poster Group attributes]
Attribute Purpose:	Poster configuration
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	No
Attribute Read-Only?	Yes [other read-only attributes]
Attribute Intrinsic?	No
Attribute First Added:	9.6.0
Attribute Last Altered:	As at baseline

Records the poster's width in pixels to aid template configuration

The default value is `0`, and is *read-only*.

Added in v9.6.0, this attribute aids in configuring the HTML in poster templates. The pixel value is for map view at the magnification currently in use.

ScrivenerID

Attribute Data Type:	string [other string-type attributes]
Attribute Default Value:	(not set - empty string)
Attribute Group:	Scrivener [other Scrivener Group attributes]
Attribute Purpose:	Data synch
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	No
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

Used for Scrivener Import.

ScrivenerKeywords

Attribute Data Type:	set [other set-type attributes]
Attribute Default Value:	(not set - empty string)
Attribute Group:	Scrivener [other Scrivener Group attributes]
Attribute Purpose:	Data synch
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	No
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

Used for Scrivener Import.

ScrivenerLabel

Attribute Data Type:	string [other string-type attributes]
Attribute Default Value:	(not set - empty string)
Attribute Group:	Scrivener [other Scrivener Group attributes]
Attribute Purpose:	Data synch
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	No
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

Used for Scrivener Import.

ScrivenerLabelID

Attribute Data Type:	number [other number-type attributes]
Attribute Default Value:	(not set - 0)
Attribute Group:	Scrivener [other Scrivener Group attributes]
Attribute Purpose:	Data synch
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	No
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

Used for Scrivener Import.

ScrivenerNote

Attribute Data Type:	string [other string-type attributes]
Attribute Default Value:	(not set - empty string)
Attribute Group:	Scrivener [other Scrivener Group attributes]
Attribute Purpose:	Data synch
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	No
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

Used for Scrivener Import.

ScrivenerStatus

Attribute Data Type:	string [other string-type attributes]
Attribute Default Value:	(not set - empty string)
Attribute Group:	Scrivener [other Scrivener Group attributes]
Attribute Purpose:	Data synch
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	No
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

Used for Scrivener Import.

ScrivenerStatusID

Attribute Data Type:	number [other number-type attributes]
Attribute Default Value:	(not set - 0)
Attribute Group:	Scrivener [other Scrivener Group attributes]
Attribute Purpose:	Data synch
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	No
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

Used for Scrivener Import.

ScrivenerType

Attribute Data Type:	string [other string-type attributes]
Attribute Default Value:	(not set - empty string)
Attribute Group:	Scrivener [other Scrivener Group attributes]
Attribute Purpose:	Data synch
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	No
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

Used for Scrivener Import.

Searchable

Attribute Data Type:	boolean [other boolean-type attributes]
Attribute Default Value:	true
Attribute Group:	General [other General Group attributes]
Attribute Purpose:	Agent configuration
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	No
Attribute Read-Only?	No
Attribute Intrinsic?	Yes [other intrinsic attributes]
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

This allows fine-tuning of searches.

The default value is `true`; for adornments is it always `false`. The attribute is intrinsic.

SelectionCount

Attribute Data Type:	number [other number-type attributes]
Attribute Default Value:	(calculated)
Attribute Group:	General [other General Group attributes]
Attribute Purpose:	Calculated data
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	No
Attribute Read-Only?	Yes [other read-only attributes]
Attribute Intrinsic?	Yes [other intrinsic attributes]
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

Tracks the number of times that a note is selected in a view window (read-only).
The count is cumulative from the creation of the note.
The count is incremented regardless of the view type. Moving a note on a map will involve a selection. Thus the count *may* not reflect the number of times the user selected the note in order to engage with its contents.

Sentiment

Attribute Data Type:	number [other number-type attributes]
Attribute Default Value:	0
Attribute Group:	AI [other AI Group attributes]
Attribute Purpose:	Natural Language Processing
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	No
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

The average sentiment score set via analysis of $Text.
Sentiment analysis is part of the Tagger process.
The average is derived from the per-paragraph values stored in $Sentiments.

Sentiments

Attribute Data Type:	list [other list-type attributes]
Attribute Default Value:	0
Attribute Group:	AI [other AI Group attributes]
Attribute Purpose:	Natural Language Processing
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	No
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

A list of the sentiment score analysed of each paragraph of $Text.
Sentiment analysis is part of the Tagger process.
The average sentiment for the entire note is stored in $Sentiment.

Separator

Attribute Data Type:	boolean [other boolean-type attributes]
Attribute Default Value:	false
Attribute Group:	Outline [other Outline Group attributes]
Attribute Purpose:	Outline configuration
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	Yes
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	9.6.0

Deprecated: controls whether a note or agent is rendered as a separator in Outline view.
From v9.6.0, this is replaced by the more explicit $IsSeparator.
If notes are given no name, i.e. shown just as a line, a duplicate name warning is not issued even if reporting of such is turned on as a Warning Preference. The text of the separator is centred if it is smaller than the width of the separator label box.
$Separator can be set via the Properties Inspector ▸ Prototype tab, 'Separator' tick-box.

Shadow

Attribute Data Type:	boolean [other boolean-type attributes]
Attribute Default Value:	false
Attribute Group:	Appearance [other Appearance Group attributes]
Attribute Purpose:	Map item shadow configuration
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	Yes
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

Controls whether an object uses drop shadows when rendered in maps.
When Shadow is `false`, the following attributes are ignored: $ShadowBlur, $ShadowColor, $ShadowDistance.
$Shadow for the current note can be set via the Appearance Inspector ▸ Shadow tab, 'Shadow' tick-box.

ShadowBlur

Attribute Data Type: number [other number-type attributes]
Attribute Default Value: 10
Attribute Group: Appearance [other Appearance Group attributes]
Attribute Purpose: Map item shadow configuration
Attribute Inherited from Preferences? No
Attribute UI-configurable? Yes
Attribute Read-Only? No
Attribute Intrinsic? No
Attribute First Added: Baseline
Attribute Last Altered: As at baseline

Blurriness of the object's map icon shadow as a numeric value.
The allowed range is 0-100.
This attribute is ignored if $Shadow is `false`.
$ShadowBlur for the current note can be set via the Appearance Inspector ▸ Shadow tab, 'Blur' control.

ShadowColor

Attribute Data Type: color [other color-type attributes]
Attribute Default Value: black
Attribute Group: Appearance [other Appearance Group attributes]
Attribute Purpose: Map item shadow configuration
Attribute Inherited from Preferences? No
Attribute UI-configurable? Yes
Attribute Read-Only? No
Attribute Intrinsic? No
Attribute First Added: Baseline
Attribute Last Altered: As at baseline

Hue of the shadow drawn for the object's map icon.
This attribute is ignored if $Shadow is `false`.
$ShadowColor for the current note can be set via the Appearance Inspector ▸ Shadow tab, 'Color' colour controls.

ShadowDistance

Attribute Data Type: number [other number-type attributes]
Attribute Default Value: 5
Attribute Group: Appearance [other Appearance Group attributes]
Attribute Purpose: Map item shadow configuration
Attribute Inherited from Preferences? No
Attribute UI-configurable? Yes
Attribute Read-Only? No
Attribute Intrinsic? No
Attribute First Added: Baseline
Attribute Last Altered: As at baseline

The numerical offset of the map icons drop-shadow (in pixels?).
The allowed range of values is 0-100.
This attribute is ignored if $Shadow is `false`.
$ShadowDistance for the current note can be set via the Appearance Inspector ▸ Shadow tab, 'Distance' control.

Shape

Attribute Data Type: string [other string-type attributes]
Attribute Default Value: (not set - empty string)
Attribute Group: Appearance [other Appearance Group attributes]
Attribute Purpose: Map item general configuration
Attribute Inherited from Preferences? No
Attribute UI-configurable? Yes
Attribute Read-Only? No
Attribute Intrinsic? No
Attribute First Added: Baseline
Attribute Last Altered: As at baseline

Stores the shape used to draw the note's icon in Map view.
The value is normally set via the Shape sub-menu of the Note menu (which lists allowed values) though it can always be set directly via the attribute. The attribute default of [no value] equates to 'rectangle' in terms of the shape displayed in Map view.
$Shape name values are case-sensitive.
$Shape for the current note can be set via the Appearance Inspector ▸ Interior tab, 'Shape' control.

ShowTitle

Attribute Data Type: boolean [other boolean-type attributes]
Attribute Default Value: false
Attribute Group: TextFormat [other TextFormat Group attributes]
Attribute Purpose: Note text
Attribute Inherited from Preferences? Yes
Attribute UI-configurable? No
Attribute Read-Only? No
Attribute Intrinsic? No
Attribute First Added: Baseline
Attribute Last Altered: As at baseline

Deprecated: retained for pre-v6 use only.
Specifies pre-v6 whether the $DisplayName of the note is displayed at the top of note text windows.
Now deprecated.

SiblingOrder

Attribute Data Type:	number [other number-type attributes]
Attribute Default Value:	(calculated)
Attribute Group:	General [other General Group attributes]
Attribute Purpose:	Calculated data
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	No
Attribute Read-Only?	No
Attribute Intrinsic?	Yes [other intrinsic attributes]
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

Sequential sibling order number (1-based).
The $SiblingOrder of the first child in a container is '1'. The $SiblingOrder of the next child is '2', and so on.
The sibling order is effectively the the $OutlineOrder but only (numbered) for the current container.
The lowest number is drawn at the front in a map. If notes with sibling orders #1 and #5 overlap, part of #5 will be behind the icon for #1.
The note menu command to move a note forward/backward or all the way to the back/front will alter both $OutlineOrder and $SiblingOrder for the affected note(s).
Separators and adornments are included in sibling order. Note that some early versions of tinderbox, adornments used not to be included in sibling order.
Regardless of $SiblingOrder values, in map view all adornments are drawn behind all notes/container/agents. It may be easier to think of the two sets of data as if being drawn on separate map overlays
Aliases do not inherit this property from their original but have their own intrinsic value.
From v9.5.0, $SiblingOrder is editable allowing actions to re-organise sibling items Previously it was read only). Of course, if a $Sort value is set for the container, the latter will re-organise any custom sibling ordering. To set first/last sibling:

- First. Set to any number 1 or less and the note will become the eldest, i.e. first, sibling.
- Last. Set to any number $ChildCount(parent) or greater and, the note will become the youngest, i.e. last, sibling.

SimplenoteKey

Attribute Data Type:	string [other string-type attributes]
Attribute Default Value:	(not set - empty string)
Attribute Group:	Watch [other Watch Group attributes]
Attribute Purpose:	Data synch
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	No
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

Used to support for Simplenote synchronisation.
For synched notes this attribute holds the Simplenote UID for the note.
This is legacy feature as support for Simplenote sync was discontinued from v8.8.0.

SimplenoteModified

Attribute Data Type:	date [other date-type attributes]
Attribute Default Value:	(not set - never)
Attribute Group:	Watch [other Watch Group attributes]
Attribute Purpose:	Data synch
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	No
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

Used to support Simplenote synchronisation.
The attribute keeps track of the effective modification date according to the Simplenote server.
This is legacy feature as support for Simplenote sync was discontinued from v8.8.0.

SimplenoteSync

Attribute Data Type:	number [other number-type attributes]
Attribute Default Value:	(not set - 0)
Attribute Group:	Watch [other Watch Group attributes]
Attribute Purpose:	Data synch
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	No
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

Used internally by Tinderbox and should not be altered by users.
This is legacy feature as support for Simplenote sync was discontinued from v8.8.0.

SimplenoteTags

Attribute Data Type:	set [other set-type attributes]
Attribute Default Value:	(not set - empty string)
Attribute Group:	Watch [other Watch Group attributes]
Attribute Purpose:	Data synch
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	No
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

Holds the Simplenote [sic] tags associated with a Simplenote-synced note (i.e. one in the /Simplenote container).
This is legacy feature as support for Simplenote sync was discontinued from v8.8.0.

SimplenoteVersion

Attribute Data Type:	number [other number-type attributes]
Attribute Default Value:	(not set - 0)
Attribute Group:	Watch [other Watch Group attributes]
Attribute Purpose:	Data synch
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	No
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

Used internally by Tinderbox and should not be altered by users.
This is legacy feature as support for Simplenote sync was discontinued from v8.8.0.

SmartLinks

Attribute Data Type:	boolean [other boolean-type attributes]
Attribute Default Value:	true
Attribute Group:	TextFormat [other TextFormat Group attributes]
Attribute Purpose:	Note text
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	Yes
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

This is `true` if $Text automatically recognises URLs.
It can be modified using Edit ▸ Substitutions ▸ SmartLinks.

SmartQuotes

Attribute Data Type:	boolean [other boolean-type attributes]
Attribute Default Value:	true
Attribute Group:	TextFormat [other TextFormat Group attributes]
Attribute Purpose:	Note text
Attribute Inherited from Preferences?	Yes
Attribute UI-configurable?	Yes
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

Allow note-level setting of use of automatic smart quotes.
Controls automatic setting (via the Apple RTF frameworks) of styling of ' smart quotes' and ' smart dashes'.
This can also be set at document level via the Document Settings Text tab. As $SmartQuotes default inherits from the latter it allows for an easy document-wide toggle for the effect.
$SmartQuotes can be set via the Text Inspector ▸ Text tab, 'Smart Quotes' tick-box control. It can also be set via Format menu ▸ Text ▸ Smart Quotes

Sort

Attribute Data Type:	string [other string-type attributes]
Attribute Default Value:	(not set - empty string)
Attribute Group:	Sorting [other Sorting Group attributes]
Attribute Purpose:	Sort configuration
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	Yes
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

Specifies the primary sorting order for notes within containers, agents and smart adornments.
The order is based on the *name* of an existing system or user attribute. The name is cited without a $-prefix as this attribute stores the desired attribute's name, not its value:

CORRECT: `$Sort = "SomeAttribute"` (case-sensitive)
FAIL: `$Sort = $SomeAttribute`

In default sorting A-Z precedes a-z, thus the sort order Aardvark, Boy, aardvark [sic], though this can be altered by $SortAlso which offers three choices of sort logic.
A secondary sort is also allowed, such as where two notes have equal values under the main sort criterion. The secondary sort is set via $SortAlso.
On the Action Inspector's Sort sub-tab, the list of available choices is shown in a $Sort pop-up menu in the upper set of controls
If unsure of allowed attributes on which to sort, the best place to set a sort is via the note's naming panel (Enter key). If exposed via $DisplayedAttributes, the value pop-up will only show values already sorted on within the document.
To reverse the sort order based on the given attribute, use $SortBackward.

SortAlso

Attribute Data Type:	string [other string-type attributes]
Attribute Default Value:	(not set - empty string)
Attribute Group:	Sorting [other Sorting Group attributes]
Attribute Purpose:	Sort configuration
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	Yes
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

Specifies the secondary sorting order for notes within containers, agents and smart adornments.
The order is based on the *name* of an existing system or user attribute. The name is cited without a $-prefix as this attribute stores the desired attribute's name, not its value:

CORRECT: `$SortAlso = "SomeAttribute"` (case-sensitive)
FAIL: `$SortAlso = $SomeAttribute`

In fact, the latter will work **if** the value $ SomeAttribute(this) is a valid attribute name.
In default sorting A-Z precedes a-z, thus the sort order Aardvark, Boy, aardvark [sic], though this can be altered by $SortAlsoTransform which offers three choices of sort logic.

A secondary sort is useful where two notes have equal values under the main sort criterion. The primary sort is set via $Sort.
On the Action Inspector's Sort sub-tab, the list of available choices is shown in a $Sort pop-up menu in the lower set of controls.
If unsure of allowed attributes on which to sort, the best place to set a sort is via the note's naming panel (Enter key). If exposed via $DisplayedAttributes, the value pop-up will only show values already sorted on within the document.
To reverse the sort order based on the given attribute, use $SortBackwardAlso.

SortAlsoTransform

Attribute Data Type:	string [other string-type attributes]
Attribute Default Value:	case-sensitive
Attribute Group:	Sorting [other Sorting Group attributes]
Attribute Purpose:	Sort configuration
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	Yes
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

Holds the sort transform value for secondary sorts of containers and agents.
The primary sort transform is stored in $SortTransform.
On the Action Inspector's Sort sub-tab, the list of available choices is shown in a Sort Transform pop-up menu.

SortBackward

Attribute Data Type:	boolean [other boolean-type attributes]
Attribute Default Value:	false
Attribute Group:	Sorting [other Sorting Group attributes]
Attribute Purpose:	Sort configuration
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	Yes
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

Specifies whether the sorting order for $Sort is backwards.
If `true` the $Sort order is reversed.
For example, Z → A instead of A → Z, etc.; thus the sort order aardvark, Boy, Aardvark [sic].
$SortBackward can be set for the current note using the Action Inspector ▸ Sort tab, 'reverse' tick-box, for the upper set of controls.

SortBackwardAlso

Attribute Data Type:	boolean [other boolean-type attributes]
Attribute Default Value:	false
Attribute Group:	Sorting [other Sorting Group attributes]
Attribute Purpose:	Sort configuration
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	Yes
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

Specifies whether the sorting order for $SortAlso is backwards.
If `true` the $SortAlso order is reversed.
For example, Z → A instead of A → Z, etc.; thus the sort order aardvark, Boy, Aardvark [sic].
$SortBackwardAlso can be set for the current note using the Action Inspector ▸ Sort tab, 'reverse' tick-box, for the lower set of controls.

SortTransform

Attribute Data Type:	string [other string-type attributes]
Attribute Default Value:	case-sensitive
Attribute Group:	Sorting [other Sorting Group attributes]
Attribute Purpose:	Sort configuration
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	Yes
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

Holds the sort transform value for sorts of containers and agents.
The secondary sort transform is stored in $SortAlsoTransform.
On the Action Inspector's Sort sub-tab, the list of available choices is shown in a Sort Transform pop-up menu.

SourceCreated

Attribute Data Type:	date [other date-type attributes]
Attribute Default Value:	(not set - never)
Attribute Group:	References [other References Group attributes]
Attribute Purpose:	Data synch
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	No
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

Used to hold creation date of DEVONthink imports.

SourceModifed

Attribute Data Type:	date [other date-type attributes]
Attribute Default Value:	(not set - never)
Attribute Group:	References [other References Group attributes]
Attribute Purpose:	Data synch
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	No
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

Used to hold the modification date of DEVONthink imports.

SourceURL

Attribute Data Type:	URL [other URL-type attributes]
Attribute Default Value:	(not set - empty string)
Attribute Group:	Watch [other Watch Group attributes]
Attribute Purpose:	Data synch
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	No
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

Holds the source URL (if any) of an imported DEVONthink item.

See more on Tinderbox use with DEVONthink.

Legacy:

Originally, it held the source URL (if any) of an item imported via an Evernote watched folder.

The watched Evernote object's data is defined in the watched folder's $EvernoteNotebook.

StartDate

Attribute Data Type:	date [other date-type attributes]
Attribute Default Value:	(not set - never)
Attribute Group:	Events [other Events Group attributes]
Attribute Purpose:	Timeline event configuration
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	No
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

A date attribute provided to facilitate organisation of tasks.

See also $DueDate and $EndDate.

For Timeline views, events with only $EndDate or an $EndDate before $StartDate will not plot correctly. If using an $EndDate, a $StartDate must be supplied. An alternate Date-type attribute to use can be supplied via $TimelineStartAttribute

State

Attribute Data Type:	string [other string-type attributes]
Attribute Default Value:	(not set - empty string)
Attribute Group:	Places [other Places Group attributes]
Attribute Purpose:	Locational data
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	No
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

The State field of an address.

The automatic geocoding tries to supply these fields automatically when looking up the supplied $Address.

Sticky

Attribute Data Type:	boolean [other boolean-type attributes]
Attribute Default Value:	false
Attribute Group:	Map [other Map Group attributes]
Attribute Purpose:	Map configuration
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	Yes
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

Controls whether moving an adornment also drags any notes wholly or partially on top of it.

This attribute only affects adornments, setting it in other objects such as notes has no effect.

If an adornment is sticky, objects (including other adornments, notes, agents, and containers) that overlap the adornment will move when the adornment is repositioned. This makes it much easier to work with moving sections of very large, complex maps.

Note that, while moving a sticky adornment moves notes on top of it, items on a 'sticky' adornment can still be moved independently, either on the adornment or to move them off it. In the latter case the object is no longer 'stuck' to the adornment. Moving an item off a sticky adornment has no effect on the adornment itself.

The icons used for $Sticky state can be customised.

$Sticky can be (re-)set several ways:

- Document Inspector ▸ More tab, 'Sticky' tick-box.
- Clicking the push-pin item on a selected adornment. This icon is only displayed by adornment and only when selected. The default state is `false` (i.e. not ticked) in which state the pushpin is greyed-out. When the icon is clicked (or set via other means) the icon turns blue. The sticky icon is not seen if the adornment is part of a multiple selection.
- Get Info ▸ attributes ▸ Map group: set $Sticky.
- Set $Sticky via action code.

Subtitle

Attribute Data Type:	string [other string-type attributes]
Attribute Default Value:	(not set - empty string)
Attribute Group:	General [other General Group attributes]
Attribute Purpose:	Map item subtitle configuration
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	Yes
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

Holds the optional subtitle text that is displayed beneath the note title in map view.

This subtitle, typically drawn in smaller type, provides means to give additional information about the note. The subtitle can be set or changed through the $Subtitle attribute.

The subtitle may most easily be set in the Inspector's name pane or directly into the attribute via code or Info view. Subtitles may also be set or edited by clicking and holding an existing subtitle, or by clicking and holding beneath the note's name even if there is no current subtitle. The mechanism is the same as Edit-in-Place.

Subtitles only appear in maps, and appear only if the note affords sufficient space. ~~However, the subtitle is displayed on hover in outlines if the note has no Hover Expression~~ (feature is currently withdrawn). In a map both $Subtitle and $HoverExpression can be displayed (assuming they both have a value).

Other attributes allow fine tuning of the look of subtitles:

- $SubtitleOpacity (a percentage, defaults to 75).
- $SubtitleSize (a percentage of the main title size, defaults to 80).
- $SubtitleColor (defaults to automatic, in which case $NameColor is used).
- Alignment of subtitle text is controlled via the note title's $NameAlignment, ensuring both title and subtitle always use the same alignment.

$Subtitle can be set for the current note via the Text Inspector ▸ Subtitle tab, 'Subtitle' text box.

SubtitleColor

Attribute Data Type:	color [other color-type attributes]
Attribute Default Value:	automatic
Attribute Group:	Map [other Map Group attributes]
Attribute Purpose:	Map item subtitle configuration
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	Yes
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

Sets the colour of $Subtitle text.

The default setting is automatic which is the same as the $NameColor default.

$SubtitleColor can be set for the current note via the Text Inspector ▸ Subtitle tab, 'Color' colour controls.

SubtitleOpacity

Attribute Data Type:	number [other number-type attributes]
Attribute Default Value:	50
Attribute Group:	Map [other Map Group attributes]
Attribute Purpose:	Map item subtitle configuration
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	Yes
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

Controls the opacity with which the text of $Subtitle is drawn.

Both $TitleOpacity and $SubtitleOpacity are independent of $Opacity. However, the if higher than the value of $Opacity, the latter value is used when drawing the note.

$Subtitle can be set for the current note via the Text Inspector ▸ Subtitle tab, 'Opacity' slider control.

SubtitleSize

Attribute Data Type:	number [other number-type attributes]
Attribute Default Value:	75
Attribute Group:	Map [other Map Group attributes]
Attribute Purpose:	Map item subtitle configuration
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	Yes
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

The size of $Subtitle text in Map view.

The number is a percentage of the text size of the $Name label.

$SubtitleSize can be set for the current note via the Text Inspector ▸ Subtitle tab, 'Size' control.

SyntaxHighlighting

Attribute Data Type:	string [other string-type attributes]
Attribute Default Value:	(not set - empty string)
Attribute Group:	TextFormat [other TextFormat Group attributes]
Attribute Purpose:	Note text
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	No
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

Stores the note's desired highlighter type.

TableExpression

Attribute Data Type:	action [other action-type attributes]
Attribute Default Value:	(not set - empty string)
Attribute Group:	Map [other Map Group attributes]
Attribute Purpose:	Action code
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	Yes
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

Expression for display of summary displays in map container icons.

Displayed item is for the container's child notes; descendants are not included.

For example, if $TableExpression is

```
$Name+"|"+$WordCount
```

Tinderbox will draw a two-column table containing the $Name and $WordCount of the first few children of the container. The "|" character (often called a vertical bar or *pipe*) separates the columns of the table.

Optional column headings can be specified via $TableHeading.

The expression code may be conditional. This lists all children with $Name and $Sibling order data displayed:

```
if($Prototype != "Attributes"){$Name+"|"+$SiblingOrder}
```

This conditional version ignores any children that are agents:

```
if(!$AgentQuery){$Name+"|"+$SiblingOrder}
```

This version only lists items of prototype type "Event":

```
if($Prototype += "Event"){$Name+"|"+$SiblingOrder}
```

$TableExpression can be set for the current container via the Summary Table Properties pop-over.

TableHeading

Attribute Data Type:	string [other string-type attributes]
Attribute Default Value:	(not set - empty string)
Attribute Group:	Map [other Map Group attributes]
Attribute Purpose:	Map item general configuration
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	Yes
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

Optional column headings for $TableExpression summary displays in containers.

Columns should be separated by a "|". Thus:

```
$TableHeading="Word Count|Value"
```

would be used as heading for a list of $WordCount values.

Tinderbox uses the pipe (|) character to parse out the individual column titles from the overall string. Remember that if using attribute names as column titles, you want to refer to the actual name and not a reference to it, so do not use a $- prefix.

The data for such tables is specified via TableExpression.

N.B. when entering such a value via Info view, do not include the enclosing quotes.

$TableHeading can be set for the current container via the Summary Table Properties pop-over.

Tabs

Attribute Data Type:	string [other string-type attributes]
Attribute Default Value:	0.25;0.25;0.25;0.25;
Attribute Group:	TextFormat [other TextFormat Group attributes]
Attribute Purpose:	Note text
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	No
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

A semi-colon delimited list of intervals between tab stops, measured in inches.

Tab stops are thus adjustable. For example, if $Tabs is

```
1;3;0.5
```

…the first tab stop will be one inch from the left edge. The second tab stop will be three inches to the right of the first, four inches from the edge. The third tab stops will be placed at intervals of 0.5".

Although stored in the form of a List, the attribute is String-type.

In older files the default tabs settings may vary.

Tab stops are measured from the left margin of the text.

The document $Tabs value defines the tab stops for the document's default text paragraph style.

Tags

Attribute Data Type:	set [other set-type attributes]
Attribute Default Value:	(not set - empty string)
Attribute Group:	References [other References Group attributes]
Attribute Purpose:	General data
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	No
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

A set of free-form tags that you can use to describe note topics, context, or other useful attributes.

When notes are copied/pasted to DEVONThink v2.8.8+, the $Tags values will be imported and applied as DevonThink tags.

Telephone

Attribute Data Type:	string [other string-type attributes]
Attribute Default Value:	(not set - empty string)
Attribute Group:	People [other People Group attributes]
Attribute Purpose:	Person detail
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	No
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

Intended for storing a person's phone number.

Tinderbox extracts the first phone number found in the $Text and automatically stores it in $Telephone.

Text

Attribute Data Type:	string [other string-type attributes]
Attribute Default Value:	(not set - empty string)
Attribute Group:	Textual [other Textual Group attributes]
Attribute Purpose:	Note text
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	Yes
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

The text, i.e. body copy, of any note or other object.

The word count of $Text is given by $WordCount and the character count of $Text is given by $TextLength.

The contents of $Text can be altered both by typing into the text pane of a note text window or via action code methods. If using the latter method, novice users should be very careful about appending to $Text as repeating actions can unintentionally rapidly lead to very large amounts of data. Setting $Text via an action code results in new text using the $TextFont font.

If a note is set to use $AutoFetch its $ReadOnly attribute is set to `true`, rendering the text un-editable (either turn off auto-fetch or alter $ReadOnly's value to edit $Text again).

$Text for the current note can be set or edited via the $Text area of the Text tab of the Text pane.

TextAlign

Attribute Data Type:	string [other string-type attributes]
Attribute Default Value:	(not set - empty string)
Attribute Group:	TextFormat [other TextFormat Group attributes]
Attribute Purpose:	Note text
Attribute Inherited from Preferences?	Yes
Attribute UI-configurable?	No
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

Deprecated: retained for pre-v6 use only.

In pre-v6…

Justification style for $Text display on map icons.

The underlying default value is an empty string. The Text Preferences default is left and thus the inherited default. Values allowed: "left", "center", "right".

This alignment is also reflected in note text displayed in map note icons.

TextBackgroundColor

Attribute Data Type:	color [other color-type attributes]
Attribute Default Value:	#ffffff
Attribute Group:	TextFormat [other TextFormat Group attributes]
Attribute Purpose:	Note text
Attribute Inherited from Preferences?	Yes
Attribute UI-configurable?	No
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

Background colour for $Text displayed in map icons.

The attribute default is not set but inherits `#ffffff` (white) from the Preferences Text pane's Background colour option.

TextColor

Attribute Data Type: color [other color-type attributes]
Attribute Default Value: #000000
Attribute Group: TextFormat [other TextFormat Group attributes]
Attribute Purpose: Note text
Attribute Inherited from Preferences? Yes
Attribute UI-configurable? Yes
Attribute Read-Only? No
Attribute Intrinsic? No
Attribute First Added: Baseline
Attribute Last Altered: As at baseline

Default display colour of $Text.

This colour may be modified by further text editing; it is the 'default' colour for new text in that note.

IMPORTANT: Changes to this attribute affect *only those notes with no current $Text* . Once anything has been added to $Text, inheritance is broken. Thereafter $Text uses the relevant attribute settings as at the time the text was added; they are effectively 'baked' into the $Text. The only way to 'reset' text with the wrong $TextColor value is to select the text and apply plain text formatting (Style menu). **WARNING**: setting plain text resets **all** $Text styling attributes. Is is not possib to 'reset' only font but not colour, text size, bolding, etc., as might be otherwise intuited.

A color-type attribute. The attribute default is not set but inherits `#000000` (black) from the Document Settings' Text pane's Text color option.

$TextColor for the selected note can be set via the Text Inspector ▸ Text tab, 'Color' colour controls.

TextColorBlue

Attribute Data Type: color [other color-type attributes]
Attribute Default Value: bright blue
Attribute Group: TextFormat [other TextFormat Group attributes]
Attribute Purpose: Note text
Attribute Inherited from Preferences? No
Attribute UI-configurable? No
Attribute Read-Only? No
Attribute Intrinsic? No
Attribute First Added: Baseline
Attribute Last Altered: As at baseline

Controls the exact colour applied via Format ▸ Style ▸ Blue.

Changing this attributes does not change text colours previously applied, but affects future applications of this style.

TextColorGray

Attribute Data Type: color [other color-type attributes]
Attribute Default Value: warm gray
Attribute Group: TextFormat [other TextFormat Group attributes]
Attribute Purpose: Note text
Attribute Inherited from Preferences? No
Attribute UI-configurable? No
Attribute Read-Only? No
Attribute Intrinsic? No
Attribute First Added: Baseline
Attribute Last Altered: As at baseline

Controls the exact colour applied via Format ▸ Style ▸ Gray.

Changing this attributes does not change text colours previously applied, but affects future applications of this style.

TextColorGreen

Attribute Data Type: color [other color-type attributes]
Attribute Default Value: bright green
Attribute Group: TextFormat [other TextFormat Group attributes]
Attribute Purpose: Note text
Attribute Inherited from Preferences? No
Attribute UI-configurable? No
Attribute Read-Only? No
Attribute Intrinsic? No
Attribute First Added: Baseline
Attribute Last Altered: As at baseline

---

the exact colour applied via Format ▸ Style ▸ Green.

Changing this attributes does not change text colours previously applied, but affects future applications of this style.

TextColorRed

Attribute Data Type: color [other color-type attributes]
Attribute Default Value: bright red
Attribute Group: TextFormat [other TextFormat Group attributes]
Attribute Purpose: Note text
Attribute Inherited from Preferences? No
Attribute UI-configurable? No
Attribute Read-Only? No
Attribute Intrinsic? No
Attribute First Added: Baseline
Attribute Last Altered: As at baseline

---

the exact colour applied via Format ▸ Style ▸ Red.

Changing this attributes does not change text colours previously applied, but affects future applications of this style.

TextExportTemplate

Attribute Data Type:	string [other string-type attributes]
Attribute Default Value:	outline
Attribute Group:	Textual [other Textual Group attributes]
Attribute Purpose:	Text export file configuration
Attribute Inherited from Preferences?	Yes
Attribute UI-configurable?	No
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

Deprecated: retained for pre-v6 use only.

Specifies the name of the template to be used for text export (as opposed to HTML export).

TextFont

Attribute Data Type:	font [other font-type attributes]
Attribute Default Value:	MercurySSm-Book
Attribute Group:	TextFormat [other TextFormat Group attributes]
Attribute Purpose:	Note text
Attribute Inherited from Preferences?	Yes
Attribute UI-configurable?	Yes
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

Default display font for new notes

The default value is inherited from Preferences, Text, Font. The default is *MercurySSm-Book* (at 16pt).

This attribute allows a class of notes (e.g. via a prototype) to inherit a specific body text font via a prototype.

$TextFont is easily overridden simply by selecting a font from the Font palette.

IMPORTANT: Changes to this attribute affect *only those notes with no current $Text* . Once anything has been added to $Text, inheritance is broken. Thereafter $Text uses the relevant attribute settings as at the time the text was added; they are effectively 'baked' into the $Text. The only way to 'reset' text with the wrong $TextFont value is to select the text and apply plain text formatting (Style menu). **WARNING**: setting plain text resets **all** $Text styling attributes. It is not possible to 'reset' only font but not colour, text size, bolding, etc., as might be otherwise intuited.

$TextFont is used for any Text set via an action or rule.

The $TextFont for the current note—or $Text selection therein, can be set using Format menu ▸ Font menu ▸ Show Fonts and using the OS Fonts palette to select the desired font name/face/size.

TextFontSize

Attribute Data Type:	number [other number-type attributes]
Attribute Default Value:	16
Attribute Group:	TextFormat [other TextFormat Group attributes]
Attribute Purpose:	Note text
Attribute Inherited from Preferences?	Yes
Attribute UI-configurable?	Yes
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

Sets the default text font size for new notes.

The value is inherited from Document settings, Text font.

This attribute allows a set of notes (e.g. Code Examples or Shopping List) to inherit a specific body text font size via a prototype.

$TextFontSize is easily overridden simply by selecting a font size from the Font palette.

IMPORTANT: Changes to this attribute affect *only those notes with no current $Text* . Once anything has been added to $Text, inheritance is broken. Thereafter $Text uses the relevant attribute settings as at the time the text was added; they are effectively 'baked' into the $Text. The only way to 'reset' text with the wrong $TextFontSize value is to select the text and use menu Format ▸ Style ▸ Standard Size.

The $TextFontSize for the current note - or $Text selection therein, can be set using Format menu ▸ Font menu ▸ Show Fonts and using the OS Fonts palette to select the desired font size.

TextHighlightBlue

Attribute Data Type:	color [other color-type attributes]
Attribute Default Value:	#3399FF
Attribute Group:	TextFormat [other TextFormat Group attributes]
Attribute Purpose:	Note text
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	No
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

the colour applied by Format ▸ Style ▸ Highlight ▸ blue highlight.

Changing this attribute does not change any text highlights previously applied, but only affects future applications of this style.

TextHighlightGreen

Attribute Data Type:	color [other color-type attributes]
Attribute Default Value:	#33FF66
Attribute Group:	TextFormat [other TextFormat Group attributes]
Attribute Purpose:	Note text
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	No
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

the colour applied by Format ▸ Style ▸ Highlight ▸ green highlight.

Changing this attribute does not change any text highlights previously applied, but only affects future applications of this style.

TextHighlightMagenta

Attribute Data Type:	color [other color-type attributes]
Attribute Default Value:	#ff66ff
Attribute Group:	TextFormat [other TextFormat Group attributes]
Attribute Purpose:	Note text
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	No
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

the colour applied by Format ▸ Style ▸ Highlight ▸ magenta highlight.
Changing this attribute does not change any text highlights previously applied, but only affects future applications of this style.

TextHighlightRed

Attribute Data Type:	color [other color-type attributes]
Attribute Default Value:	#FF5533
Attribute Group:	TextFormat [other TextFormat Group attributes]
Attribute Purpose:	Note text
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	No
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

This lets you control the colour applied by Format ▸ Style ▸ Highlight ▸ red highlight.
Changing this attribute does not change any text highlights previously applied, but only affects future applications of this style.

TextHighlightYellow

Attribute Data Type:	color [other color-type attributes]
Attribute Default Value:	#ffff00
Attribute Group:	TextFormat [other TextFormat Group attributes]
Attribute Purpose:	Note text
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	No
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

This lets you control the colour applied by Format ▸ Style ▸ Highlight ▸ yellow highlight.
Changing this attribute does not change any text highlights previously applied, but only affects future applications of this style.

TextLength

Attribute Data Type:	number [other number-type attributes]
Attribute Default Value:	(not set - 0)
Attribute Group:	Textual [other Textual Group attributes]
Attribute Purpose:	Calculated data
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	No
Attribute Read-Only?	Yes [other read-only attributes]
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

The number of characters (character count) of the current note (read-only).
See also $WordCount for a count of discrete words in the note's text.
More detail on how character counts are made.

TextLinkCount

Attribute Data Type:	number [other number-type attributes]
Attribute Default Value:	(calculated)
Attribute Group:	General [other General Group attributes]
Attribute Purpose:	Calculated data
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	No
Attribute Read-Only?	Yes [other read-only attributes]
Attribute Intrinsic?	Yes [other intrinsic attributes]
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

Holds the number of outbound text links (i.e. within $Text).
The value is (internal) text links *plus* (external) web links. Thus if you need the count of only text-type links, use an expression like `$TextInternalLinkCount = $TextLinkCount-$WebLinkCount`. Thus, if a note has two text links and one web link, the note's $TextLinkCount value would be 3 but user attribute $TextInternalLinkCount would have a value of 2.
Also see $PlainLinkCount, $WebLinkCount, $OutboundLinkCount.

TextPaneRatio

Attribute Data Type:	number [other number-type attributes]
Attribute Default Value:	0.5
Attribute Group:	TextFormat [other TextFormat Group attributes]
Attribute Purpose:	General data
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	No
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

Deprecated: unused retained for v6 use only.
Custom ratio on main view to text view for the current note.
Default value is 0.5, i.e. 50% split. Value range is 0-100.
If a custom $TextPaneWidth is set, the latter overrides this setting.
Currently, the splitter pane location is stored in the per-tab XML metadata of the document window, and is not exposed via an attribute.

TextPaneWidth

Attribute Data Type:	number [other number-type attributes]
Attribute Default Value:	-1
Attribute Group:	TextFormat [other TextFormat Group attributes]
Attribute Purpose:	General data
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	No
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

Deprecated: unused retained for v6 use only.
Stores a custom width for the note's text pane.
A value of -1 leaves the text pane unmoved, and inheriting the split set/inherited via $TextPaneRatio. Dragging the document window's vertical splitter pane automatically sets $TextPaneWidth.
The splitter pane location is stored in the per-tab XML metadata of the document window, and is not exposed via an attribute.

TextSidebar

Attribute Data Type:	boolean [other boolean-type attributes]
Attribute Default Value:	true
Attribute Group:	TextFormat [other TextFormat Group attributes]
Attribute Purpose:	Note text
Attribute Inherited from Preferences?	Yes
Attribute UI-configurable?	No
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

Deprecated: retained for pre-v6 use only.
Stores the pre-v6 display state (shown/hidden) of a note's text window sidebar. In the current application, the old sidebar is replaced by the displayed attributes table, and this attribute's feature is replaced by $HideDisplayedAttributes.

TextWindowHeight

Attribute Data Type:	number [other number-type attributes]
Attribute Default Value:	250
Attribute Group:	TextFormat [other TextFormat Group attributes]
Attribute Purpose:	Note window configuration
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	No
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

Last-used height of note's stand-alone window.
Resizing the note's text window automatically sets this attribute, so that the text window for each note remembers its last-used height. The new value is saved when the window is closed or document is closed with the text window still open.

TextWindowWidth

Attribute Data Type:	number [other number-type attributes]
Attribute Default Value:	475
Attribute Group:	TextFormat [other TextFormat Group attributes]
Attribute Purpose:	Note window configuration
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	No
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

Last-used width of note's stand-alone window.
Resizing the note's text window automatically sets this attribute, so that the text window for each note remembers its last-used width. The new value is saved when the window is closed or document is closed with the text window still open.

TimelineAliases

Attribute Data Type:	boolean [other boolean-type attributes]
Attribute Default Value:	true
Attribute Group:	Events [other Events Group attributes]
Attribute Purpose:	Timeline configuration
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	Yes
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

Control plotting of in-scope aliases in the timeline.
Default is `true`, in which case in-scope aliases are included in the timeline.
$TimelineAliases for the current timeline (tab) view can be set via the Timeline Settings pop-over, 'Alaises' tick-box.

TimelineBand

Attribute Data Type:	number [other number-type attributes]
Attribute Default Value:	0
Attribute Group:	Events [other Events Group attributes]
Attribute Purpose:	Timeline event configuration
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	No
Attribute Read-Only?	No
Attribute Intrinsic?	Yes [other intrinsic attributes]
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

Assigns an event to a specific vertical band in Timeline view.
By default all events are in band 0, but can be set to another band by drag-drop or via action code.
Although the attribute accepts any number, the valid range is whole numbers 0–1300. 1300 is not a hard limit and that many bands should not be needed, but above that number timeline band labels are clipped or not drawn at all.
This attribute is intrinsic for aliases, allowing aliases to be paced in a different timeline band to their original.

TimelineBandLabelColor

Attribute Data Type:	color [other color-type attributes]
Attribute Default Value:	black
Attribute Group:	Events [other Events Group attributes]
Attribute Purpose:	Timeline configuration
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	Yes
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

Holds the colour used to draw band labels in a timeline.
When set locally it applies to the current container's Timeline view.
$TimelineBandLabelColor for the current timeline (tab) view can be set via the Timeline Settings pop-over, 'Label Color' colour controls.

TimelineBandLabelOpacity

Attribute Data Type:	number [other number-type attributes]
Attribute Default Value:	25
Attribute Group:	Events [other Events Group attributes]
Attribute Purpose:	Timeline configuration
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	Yes
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

Stores an opacity value for drawing band labels in the current container's Timeline view.
$Timeline for the current timeline (tab) view can be set via the Timeline Settings pop-over, '' control.

TimelineBandLabels

Attribute Data Type:	list [other list-type attributes]
Attribute Default Value:	(not set - empty string)
Attribute Group:	Events [other Events Group attributes]
Attribute Purpose:	Timeline configuration
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	Yes
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

Holds the assigned labels for different Timeline view bands and is a semicolon-delimited string.
For example, the string "Paris;London;Madrid" would label three bands, "Paris", "London", and "Madrid". Labels are drawn onto the timeline in the order stored; in the example band number 0 will be labelled "Paris".
$TimelineBandLabels for the current timeline (tab) view can be set via the Timeline Settings pop-over, 'Labels' input box.

TimelineColor

Attribute Data Type:	color [other color-type attributes]
Attribute Default Value:	lighter warm gray
Attribute Group:	Events [other Events Group attributes]
Attribute Purpose:	Timeline configuration
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	Yes
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

Used to colour every other (even-numbered) timeline band within a Timeline view.
A desaturated tint of the colour, rather than the exact colour name/value specified, is used to avoid undue contrast.
$TimelineColor for the current timeline (tab) view can be set via the Timeline Settings pop-over, 'Band Color' colour controls.

TimelineDescendants

Attribute Data Type:	boolean [other boolean-type attributes]
Attribute Default Value:	true
Attribute Group:	Events [other Events Group attributes]
Attribute Purpose:	Timeline configuration
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	Yes
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

Control scope to direct children in the timeline.
Default is `true`, which results in all descendants being plotted.
$TimelineDescendants for the current timeline (tab) view can be set via the Timeline Settings pop-over, 'Descendants' tick-box.

TimelineEnd

Attribute Data Type:	date [other date-type attributes]
Attribute Default Value:	(not set - never)
Attribute Group:	Events [other Events Group attributes]
Attribute Purpose:	Timeline configuration
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	Yes
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

Optional end date for container's Timeline.
Notes with a $EndDate later than this are listed in the "Out of Range" section of the view. By default, this attribute is not set.
The setting works at container level, thus different containers can have different plotted date ranges.
$Timeline for the current timeline (tab) view can be set via the Timeline Settings pop-over, 'End' date input box.

TimelineEndAttribute

Attribute Data Type:	string [other string-type attributes]
Attribute Default Value:	(not set - empty string)
Attribute Group:	Events [other Events Group attributes]
Attribute Purpose:	Timeline configuration
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	Yes
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

Alternative Date-type attribute of Timeline's event $EndDate.
Use the attribute name, without a $-prefix.
$TimelineEndAttribute for the current timeline (tab) view can be set via the Timeline Settings pop-over, 'Attributes: End' pop-up control.

TimelineGridColor

Attribute Data Type:	color [other color-type attributes]
Attribute Default Value:	#009090
Attribute Group:	Events [other Events Group attributes]
Attribute Purpose:	Timeline configuration
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	No
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

Colour of timeline vertical guidelines for selected note in Timeline view.

TimelineMarker

Attribute Data Type:	boolean [other boolean-type attributes]
Attribute Default Value:	false
Attribute Group:	Events [other Events Group attributes]
Attribute Purpose:	Timeline configuration
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	No
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

Draw persistent note vertical markers in Timeline view.

Controls whether marker lines, in a tint of $AccentColor, are drawn from events to the Timeline view's scale persist when a note is not currently selected.

The marker lines are drawn from the left edge of events. The default is `false`, with lines only shown for selected items.

TimelineScaleColor

Attribute Data Type:	color [other color-type attributes]
Attribute Default Value:	cool gray
Attribute Group:	Events [other Events Group attributes]
Attribute Purpose:	Timeline configuration
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	Yes
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

Primary colour in the gradient used for time scale of Timeline view.

The default is 'cool gray'. The scale uses a gradient to fade to $TimelineScaleColor2.

$TimelineScaleColor for the current timeline (tab) view can be set via the Timeline Settings pop-over, 'Scale Color' colour controls.

TimelineScaleColor2

Attribute Data Type:	color [other color-type attributes]
Attribute Default Value:	(not set - empty string)
Attribute Group:	Events [other Events Group attributes]
Attribute Purpose:	Timeline configuration
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	No
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

Primary colour in the gradient used for time scale of Timeline view.

The default is inherited from the note's $MapBackgroundColor (which is in turn inherited from the Map Preference for background colour) which results in '#f2f2e6'. The scale uses a gradient to fade from $TimelineScaleColor. The default means the top edge of the scale bar is the same as the background of the main content portion of the map.

TimelineStart

Attribute Data Type:	date [other date-type attributes]
Attribute Default Value:	(not set - never)
Attribute Group:	Events [other Events Group attributes]
Attribute Purpose:	Timeline configuration
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	Yes
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

Optional start date for container's Timeline.

Notes with a $StartDate earlier than this are listed in the "out of Range" section of the view.

By default, this attribute is not set.

The setting works at container level, thus different containers can have different plotted date ranges.

$TimelineStart for the current timeline (tab) view can be set via the Timeline Settings pop-over, 'Start' date input box.

TimelineStartAttribute

Attribute Data Type:	string [other string-type attributes]
Attribute Default Value:	(not set - empty string)
Attribute Group:	Events [other Events Group attributes]
Attribute Purpose:	Timeline configuration
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	Yes
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

Alternative Date-type attribute of Timeline's event $StartDate.

Use the attribute name, without a $-prefix.

$TimelineStartAttribute for the current timeline (tab) view can be set via the Timeline Settings pop-over, 'Attributes: Start' pop-up control.

Tip

Attribute Data Type:	number [other number-type attributes]
Attribute Default Value:	0.5
Attribute Group:	Map [other Map Group attributes]
Attribute Purpose:	Map item general configuration
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	Yes
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

Controls the shape of arrow and leaf shapes.

The valid range is 0–1.

Replaces the deprecated $LeafTip.

For notes whose $Shape employs this setting, the $Tip value for the selected note can be altered using the grey dot control to the bottom of the note, in an up-down direction.

TitleBackgroundColor

Attribute Data Type:	color [other color-type attributes]
Attribute Default Value:	#000000
Attribute Group:	TextFormat [other TextFormat Group attributes]
Attribute Purpose:	Note title
Attribute Inherited from Preferences?	Yes
Attribute UI-configurable?	No
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

Deprecated: retained for pre-v6 use only.

Specifies the background colour for the title pane in text windows (not used in v6+).

TitleFont

Attribute Data Type:	font [other font-type attributes]
Attribute Default Value:	LucidaGrande
Attribute Group:	TextFormat [other TextFormat Group attributes]
Attribute Purpose:	Note title
Attribute Inherited from Preferences?	Yes
Attribute UI-configurable?	No
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

Deprecated: retained for pre-v6 use only.

Specifies the name of the font used for displaying the current note's title in the title pane of the note's text window (ignored in v6+).

TitleForegroundColor

Attribute Data Type:	color [other color-type attributes]
Attribute Default Value:	#ffffff
Attribute Group:	TextFormat [other TextFormat Group attributes]
Attribute Purpose:	Note title
Attribute Inherited from Preferences?	Yes
Attribute UI-configurable?	No
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

Deprecated: retained for pre-v6 use only.

Specifies the foreground colour for the title pane in text windows (not used in v6+).

TitleHeight

Attribute Data Type:	number [other number-type attributes]
Attribute Default Value:	0
Attribute Group:	Map [other Map Group attributes]
Attribute Purpose:	Map item text configuration
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	No
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

Sets height of the title bar in a map container/agent map icon, in map units.

$TitleHeight is only used by agents and container notes and ignored by other objects. It is expressed in relative map units, as for $Height and $Width.

For a (selected) container, a note or agent, a splitter resize mouse icon is shown when the mouse is over the boundary of the title area and the map viewport; click and drag (hand icon) the mouse up or down to alter the height of the title area.

The default value of 0 (or a set value of 0) allows the icon to use Tinderbox's normal auto-resizing where the title area grows as the overall icon area does.

If $TitleHeight is set to other than zero, then resizing the overall size of the container has no effect on the height of the title area.

To 'hide' the viewport area of a container, set the $TitleHeight to equal $Height. This makes the map icon look like a normal note except the two bottom corners (note) or top corners (agent) are rounded. While the viewport is hidden, it is not possible to double-click drill down but its shortcut (Cmd+Opt+Return) will work.

TitleOpacity

Attribute Data Type:	number [other number-type attributes]
Attribute Default Value:	100
Attribute Group:	Map [other Map Group attributes]
Attribute Purpose:	Map item text configuration
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	No
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

Controls the opacity used when drawing the $Name onto a map icon.

The default is 100, totally opaque.

Both $TitleOpacity and $SubtitleOpacity are independent of $Opacity. However, the if higher than the value of $Opacity, the latter value is used when drawing the note.

Tot

Attribute Data Type:	set [other set-type attributes]
Attribute Default Value:	(not set - empty string)
Attribute Group:	Watch [other Watch Group attributes]
Attribute Purpose:	Data synch
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	No
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

A Set indicating which Tot notes are watched.

The default value is an empty string.

Twitter

Attribute Data Type:	string [other string-type attributes]
Attribute Default Value:	(not set - empty string)
Attribute Group:	People [other People Group attributes]
Attribute Purpose:	Person detail
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	No
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

Holds a person's Twitter account name.

UpdateTextLinksAfterRename

Attribute Data Type:	boolean [other boolean-type attributes]
Attribute Default Value:	true
Attribute Group:	Textual [other Textual Group attributes]
Attribute Purpose:	Data synch
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	No
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

Optionally disable automatic update to text link anchors edits.

Default: `true`

After a note is renamed, the app automatically updates text links if (a) the text link's destination is the renamed note, and (b) the text link's anchor text is the old name of the renamed note. Though this is often the desired behaviour, but may in some instances not be desirable.

Thus, an option to disable automatic renaming is available in Document Settings: Text: Update after renaming. But automatic renaming may also be disabled by setting this attribute.

In large, well-linked document that do not use note titles as link anchor text, this option is of less use as it is essentially unneeded so nugatory background work for Tinderbox to be doing.

URL

Attribute Data Type:	URL [other URL-type attributes]
Attribute Default Value:	(not set - empty string)
Attribute Group:	Net [other Net Group attributes]
Attribute Purpose:	Import configuration
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	No
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

URL to be called if the current notes fetches its text content from the web.

As well as 'http' URLs, 'mailto' addresses and other such protocols can be used, including pseudo protocols used by some desktop app. The $ViewInBrowser attribute controls whether the fetched content is shown in the note or in a web browser window.

If using a local path to a file, e.g. to enable AutoFetch use, you must use the file:// protocol rather than a bare file path.

Prior to v5 this was a string data type attribute.

UUID

Attribute Data Type:	string [other string-type attributes]
Attribute Default Value:	(not set - empty string)
Attribute Group:	Watch [other Watch Group attributes]
Attribute Purpose:	Data synch
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	No
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

Unique ID of a DEVONThink item imported via a DEVONThink watched folder.

ViewInBrowser

Attribute Data Type:	boolean [other boolean-type attributes]
Attribute Default Value:	false
Attribute Group:	Net [other Net Group attributes]
Attribute Purpose:	Import configuration
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	No
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

Opening the note's also open note's $URL in a web browser.
If `true`, and if $URL is a valid network or file URL, then Tinderbox will automatically open that URL when the note is selected.

Visits

Attribute Data Type:	number [other number-type attributes]
Attribute Default Value:	(not set - 0)
Attribute Group:	Storyspace [other Storyspace Group attributes]
Attribute Purpose:	Storyspace synch
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	No
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

In StorySpace, the number of times the reader has visited this note.
One story method is to ensure a writing space has been visited at least N before some outcomes (new links) are enabled).

Volume

Attribute Data Type:	string [other string-type attributes]
Attribute Default Value:	(not set - empty string)
Attribute Group:	References [other References Group attributes]
Attribute Purpose:	Data synch
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	No
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

Used for Bookends reference import. Maps to RIS data tag 'VL'.

WatchFolder

Attribute Data Type:	file [other file-type attributes]
Attribute Default Value:	(not set - empty file string)
Attribute Group:	Watch [other Watch Group attributes]
Attribute Purpose:	Data synch
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	No
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

The local file path to a Finder watched folder.

WebLinkCount

Attribute Data Type:	number [other number-type attributes]
Attribute Default Value:	(calculated)
Attribute Group:	General [other General Group attributes]
Attribute Purpose:	Calculated data
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	No
Attribute Read-Only?	Yes [other read-only attributes]
Attribute Intrinsic?	Yes [other intrinsic attributes]
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

Holds the number of outbound links originating from the current note (read-only).

Formerly, these were lumped into $OutboundLinkCount.

Also see $TextLinkCount, $PlainLinkCount.

WeblogPostID

Attribute Data Type:	string [other string-type attributes]
Attribute Default Value:	(not set - empty string)
Attribute Group:	Weblog [other Weblog Group attributes]
Attribute Purpose:	Weblog configuration
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	No
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

Deprecated: retained for pre-v6 use only.

Specifies the ID for weblog APIs such as MovableType, Blogger and Radio Userland.

Width

Attribute Data Type:	number [other number-type attributes]
Attribute Default Value:	3
Attribute Group:	Map [other Map Group attributes]
Attribute Purpose:	Map item general configuration
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	No
Attribute Read-Only?	No
Attribute Intrinsic?	Yes [other intrinsic attributes]
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

The width of a note icon in the Map view, in map units.

See also $Height.

Aliases do not inherit this property from their original but have their own value.

WordCount

Attribute Data Type:	number [other number-type attributes]
Attribute Default Value:	(not set - 0)
Attribute Group:	Textual [other Textual Group attributes]
Attribute Purpose:	Calculated data
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	No
Attribute Read-Only?	Yes [other read-only attributes]
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	9.5.2

The number of discrete words in the current note's $Text (read-only).

See $TextLength for the total character count).

$WordCount is not language aware. Essentially it is tuned for the heuristics of English language. Therefore it may not work well with languages that use word-spacing conventions drastically unlike English.

For those needing a word count in a non-English language/script/locale, one approach is to consider String.split(regex), populating the regex with word break characters that may differ from English. Note, in this context punctuation doesn't matter only the alternatives to spaces/non-breaking spaces that a given language may use.

Word count is as much science as art. For instance, is 'UNESCO' one word, six words—or seven if you include the 'and' omitted from the acronym. Within a single country or a single language tradition there may be consensus on such points but it may not hold true more widely.

However, for English text, $WordCount should be considered accurate.

From v9.5.2, $WordCount now works better in Chinese and several other languages. Word count, even in English still remains something of an art. For some languages, such as Chinese, $TextLength—the character count of $Text—may be more useful, if only one type of (language) script is being used in the note. The word count for mixed-script text is always likely to have some inaccuracy as language typing to sub-string level, e.g. discrete parts of $Text, does not occur.

Legacy note

In very early versions the attribute name was 'Wordcount' and the program allows either spelling. However, the old version is deprecated.

Xpos

Attribute Data Type:	number [other number-type attributes]
Attribute Default Value:	0
Attribute Group:	Map [other Map Group attributes]
Attribute Purpose:	Map configuration
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	No
Attribute Read-Only?	No
Attribute Intrinsic?	Yes [other intrinsic attributes]
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

Note's X-axis map co-ordinates (in map units).

$Xpos = 0 is the middle of the X-axis of the map, with negative values moving left, positive values moving right. See Map Co-ordinates.

See also $Ypos.

Aliases do not inherit this property from their original but have their own value.

Ypos

Attribute Data Type:	number [other number-type attributes]
Attribute Default Value:	0
Attribute Group:	Map [other Map Group attributes]
Attribute Purpose:	Map configuration
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	No
Attribute Read-Only?	No
Attribute Intrinsic?	Yes [other intrinsic attributes]
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

Note's Y-axis map co-ordinates (in map units).

$Ypos = 0 is the middle of the Y-axis of the map, with negative values moving up [sic], positive values moving down. Note the Y-axis value orientation is the reverse of normal Cartesian Y co-ordinates, but does makes sense in the context of a visual top-left-to-bottom-right visual layout. See Map Co-ordinates.

See also $Xpos.

Aliases do not inherit this property from their original but have their own value.

Ziplinks

Attribute Data Type:	boolean [other boolean-type attributes]
Attribute Default Value:	true
Attribute Group:	TextFormat [other TextFormat Group attributes]
Attribute Purpose:	Note text
Attribute Inherited from Preferences?	No
Attribute UI-configurable?	No
Attribute Read-Only?	No
Attribute Intrinsic?	No
Attribute First Added:	Baseline
Attribute Last Altered:	As at baseline

This enables Text link creation via the Ziplinks method functionality in a note.

The default value is `true`.

It can be useful to disable the feature in notes using code examples or square brackets. Some built-in prototypes, such as HTML and Code have $Ziplinks set to `false` for just such a reason.

System Attribute Groups within Tinderbox

A Tinderbox documents's *pre-defined* System Attributes are organised in various groups; being pre-defined, they are always available from outset in a new TBX document. The groupings have grown unequally over time, but broadly they describe attributes with a common context or purpose:

- Agent Attributes
- AI Attributes
- Appearance Attributes
- Composites Attributes
- Events Attributes
- General Attributes
- Grid Attributes
- HTML Attributes
- Internal Attributes (4)
- Iris Attributes
- Map Attributes
- Net Attributes
- Outline Attributes
- People Attributes
- Places Attributes
- Poster Attributes
- References Attributes
- Sandbox Attributes
- Scrivener Attributes
- Sorting Attributes
- Storyspace Attributes
- TextFormat Attributes
- Textual Attributes
- Watch Attributes
- Weblog Attributes
- User Attributes

One further group is hidden in the UI but visible in the documents source XML:

- Internal (not accessible via the UI)

Agent Attributes

These store data related to Agent function: the query string, action code, etc.

- $AgentAction
- $AgentCaseSensitive
- $AgentPriority
- $AgentQuery
- $CleanupAction

AI Attributes

These store data related to the Artificial Intelligence (AI) and Neural-Linguistic Programming (NLP) function: the query string, action code, etc.

- $DominantLanguage
- $NLNames
- $NLOrganizations
- $NLPlaces
- $NLTags
- $Sentiment
- $Sentiments

Appearance Attributes

These store data related to a note or other objects appearance in Map (and some other views).

- $AccentColor
- $Badge
- $BadgeMonochrome
- $BadgeSize
- $Border
- $BorderBevel
- $BorderColor
- $BorderDash
- $CaptionAlignment
- $CaptionColor
- $CaptionFont
- $CaptionOpacity
- $CaptionSize
- $Color
- $Color2
- $Flags
- $Opacity
- $Pattern
- $PlotColor
- $Shadow
- $ShadowBlur
- $ShadowColor
- $ShadowDistance
- $Shape

Composites Attributes

These store data related to a note functioning as a composite in Map (and some other views).

- $Associates
- $IsComposite
- $IsMultiple
- $NeverComposite
- $OnJoin
- $Role

Events Attributes

These store date data related to tasks set up by the user.

- $DueDate
- $EndDate
- $Participants
- $StartDate
- $TimelineAliases
- $TimelineBand
- $TimelineBandLabelColor
- $TimelineBandLabelOpacity
- $TimelineBandLabels
- $TimelineColor
- $TimelineDescendants
- $TimelineEnd
- $TimelineEndAttribute
- $TimelineGridColor
- $TimelineMarker
- $TimelineScaleColor
- $TimelineScaleColor2
- $TimelineStart
- $TimelineStartAttribute

General Attributes

Generic information about notes. This group has the largest proportion of calculated, read-only, values.

- $AdornmentCount
- $Aliases
- $Caption
- $Checked
- $ChildCount
- $ClusterTerms
- $Container
- $Created
- $CreatedFrom
- $Creator
- $DescendantCount
- $DisplayExpression
- $DisplayExpressionDisabled
- $DisplayName
- $Edict
- $EdictDisabled
- $File
- $HoverBackgroundColor
- $HoverExpression
- $HoverImage
- $HoverOpacity
- $ID
- $IDString
- $InboundLinkCount
- $IsAction
- $IsAdornment
- $IsAlias
- $IsPrototype
- $LocalAttributes
- $Modified
- $Name
- $OnAdd
- $OnRemove
- $OutboundLinkCount
- $OutlineDepth
- $OutlineOrder
- $Path
- $PlainLinkCount
- $Private
- $Prototype
- $PrototypeBequeathsChildren
- $ReadCount
- $Rule
- $RuleDisabled
- $Searchable
- $SelectionCount
- $SiblingOrder
- $Subtitle
- $TextLinkCount
- $WebLinkCount

Grid Attributes

Settings to customise map view adornment grids.

- $GridColor
- $GridColumns
- $GridLabelFont
- $GridLabels
- $GridLabelSize
- $GridOpacity
- $GridRows

HTML Attributes

Settings relating to HTML export configuration and mark-up.

- $HTMLBoldEnd
- $HTMLBoldStart
- $HTMLCloud1End
- $HTMLCloud1Start
- $HTMLCloud2End
- $HTMLCloud2Start
- $HTMLCloud3End
- $HTMLCloud3Start
- $HTMLCloud4End
- $HTMLCloud4Start
- $HTMLCloud5End
- $HTMLCloud5Start
- $HTMLCodeEnd
- $HTMLCodeStart
- $HTMLDontExport
- $HTMLEntities
- $HTMLExportAfter
- $HTMLExportBefore
- $HTMLExportChildren
- $HTMLExportCommand
- $HTMLExportExtension
- $HTMLExportFileName
- $HTMLExportFileNameSpacer
- $HTMLExportPath
- $HTMLExportTemplate
- $HTMLFileNameLowerCase
- $HTMLFileNameMaxLength
- $HTMLFirstParagraphEnd
- $HTMLFirstParagraphStart
- $HTMLFont
- $HTMLFontSize
- $HTMLImageEnd
- $HTMLImageStart
- $HTMLIndentedParagraphEnd
- $HTMLIndentedParagraphStart
- $HTMLItalicEnd
- $HTMLItalicStart
- $HTMLLinkExtension
- $HTMLListEnd
- $HTMLListItemEnd
- $HTMLListItemStart
- $HTMLListStart
- $HTMLMarkDown
- $HTMLMarkdown
- $HTMLMarkupText
- $HTMLOrderedListEnd
- $HTMLOrderedListItemEnd
- $HTMLOrderedListItemStart
- $HTMLOrderedListStart
- $HTMLOverwriteImages
- $HTMLParagraphEnd
- $HTMLParagraphStart
- $HTMLPreviewCommand
- $HTMLQuoteHTML
- $HTMLStrikeEnd
- $HTMLStrikeStart
- $HTMLSubscriptEnd
- $HTMLSubscriptStart
- $HTMLSuperscriptEnd
- $HTMLSuperscriptStart
- $HTMLUnderlineEnd
- $HTMLUnderlineStart
- $IsTemplate

Internal Attributes (4)

The 'Internal' group of attributes is hidden from the user and is only encountered if working with the XML source of a saved TBX file. This group cannot be accessed via the UI, such as via the documents Inspector/System tab. Do note attempt to export or use action script on these items. The group is comprised of following known attributes:

- $Alias. Unsigned type (only used by this attribute). The Alias is used by aliases only and holds $ID of the alias' original note.
- $EntryScript. String type. Purpose undocumented. At some point the app may have allowed action code to be run at doc open/close and the attribute may be for this (legacy support?).
- $ExitScript. String type. Purpose undocumented. See the previous bullet.
- $WindowPlace. [Unknown] type. Purpose undocumented, but may be used to seed values for individual window position/state stored in the TBX.

The purpose of the other attributes is unknown.

Iris Attributes

Experimental attributes. Not currently used. Ignore.

- $IrisAngle
- $IrisRadius

Map Attributes

Information relating to visual display of notes, agents and adornments. Although title 'Map' these display settings are also used to varying degrees in other sorts of Tinderbox Views.

- $AdornmentFont
- $Base
- $Bend
- $Direction
- $Fill
- $FillOffsetY
- $FillOpacity
- $Height
- $HoverFont
- $InteriorScale
- $LeafBase
- $LeafBend
- $LeafDirection
- $LeafTip
- $Lock
- $MapBackgroundAccentColor
- $MapBackgroundColor
- $MapBackgroundColor2
- $MapBackgroundFill
- $MapBackgroundFillOpacity
- $MapBackgroundPattern
- $MapBackgroundShadow
- $MapBodyTextColor
- $MapBodyTextSize
- $MapNameSize
- $MapScrollX
- $MapScrollY
- $MapTextSize
- $NameAlignment
- $NameBold
- $NameColor
- $NameFont
- $NameLeading
- $NameStrike
- $PlotBackgroundColor
- $PlotBackgroundOpacity
- $PlotColorList
- $Sticky
- $SubtitleColor
- $SubtitleOpacity
- $SubtitleSize
- $TableExpression
- $TableHeading
- $Tip
- $TitleHeight
- $TitleOpacity
- $Width
- $Xpos
- $Ypos

Net Attributes

For data derived from the Net, e.g. RSS, notes drawing content from web pages, etc.

- $AutoFetch
- $AutoFetchCommand
- $LastFetched
- $NoteURL
- $RawData
- $RSSChannelTemplate
- $RSSItemLimit
- $RSSItemTemplate
- $URL
- $ViewInBrowser

Outline Attributes

These store data related to appearance in Outline view.

- $IsSeparator
- $MapPrototypeColor
- $OutlineBackgroundColor
- $OutlineColorSwatch
- $OutlineNameSize
- $OutlineTextSize
- $PrototypeHighlightColor
- $Separator

People Attributes

This group offers a series of attributes to support the storage of information about people.
If an existing document already uses these names as user attributes, the attributes will be moved from the User attribute pane to the People group of System attributes.
Some items were moved to the new Places group in v8.0.4.

- $Email
- $EmailSubject
- $EmailTemplate
- $FullName
- $Organization
- $Telephone
- $Twitter

Places Attributes

This group offers a series of attributes related to geographical location data in notes. Previously, these attributes were in the People group:

- $Address
- $City
- $Country
- $District
- $FormattedAddress
- $GeocodedAddress
- $Latitude
- $Longitude
- $PostalCode
- $Range
- $State

Poster Attributes

This group offers a series of attributes to support the storage of information about use of posters.

- $PosterCSS
- $PosterLabels
- $PosterSettings
- $PosterTemplate
- $PosterURL
- $PosterX
- $PosterY
- $PosterZoom
- $ScreenHeight
- $ScreenWidth

References Attributes

This group holds a series of attributes to support the exchange of information with Bookends on the web.

- $Abstract
- $AccessDate
- $ArticleTitle
- $Author2
- $Author3
- $Author4
- $Authors
- $BookTitle
- $CallNumber
- $DOI
- $Edition
- $ISBN
- $Issue
- $Journal
- $Pages
- $PublicationCity
- $PublicationYear
- $Publisher
- $ReferenceDictionary
- $ReferenceRIS
- $ReferenceTitle
- $ReferenceURL
- $RefFormat
- $RefKeywords
- $RefType
- $SourceCreated
- $SourceModifed
- $Tags
- $Volume

Sandbox Attributes

This contains a set of 9 text attributes of varying data types (Boolean, Color, Date, List, Number, Set, String). The attribute names use a common prefix 'My', e.g. $MyString. These attributes are very useful for initial testing and incremental formalisation of projects. For pre-v6 documents which may have same-named/typed attributes already defined as user attributes, these will still work as before but said attributes will be accessed via the Sandbox group of system attributes.

- $MyBoolean
- $MyColor
- $MyDate
- $MyDictionary
- $MyInterval
- $MyList
- $MyNumber
- $MySet
- $MyString

Scrivener Attributes

This group holds a series of attributes to support the exchange of information with Scrivener.

- $ScrivenerID
- $ScrivenerKeywords
- $ScrivenerLabel
- $ScrivenerLabelID
- $ScrivenerNote
- $ScrivenerStatus
- $ScrivenerStatusID
- $ScrivenerType

Sorting Attributes

Flags for sorting container content. Although the document is not a visible object in Maps or Outlines it can be selected, to allow sorting top-level notes. In the top level of the view deselect all items and then open the Info view dialog (Cmd+Opt+I, or Note menu, Get Info) and setting the sort attributes for the document itself.

- $Sort
- $SortAlso
- $SortAlsoTransform
- $SortBackward
- $SortBackwardAlso
- $SortTransform

Storyspace Attributes

This group holds a series of attributes to support the exchange of information with Storyspace v3.

- $BeforeVisit
- $ChosenWord
- $Deck
- $OnVisit
- $Requirements
- $ResetAction
- $Visits

TextFormat Attributes

Information relating to visual display of a note's data in Note text windows.

- $AutomaticIndent
- $CodeFont
- $DisplayedAttributes
- $DisplayedAttributesDateFormat
- $DisplayedAttributesFont
- $DisplayedAttributesFontSize
- $HideDisplayedAttributes
- $HideKeyAttributes
- $HideTitle
- $ImageSizeLimit
- $KeyAttributeDateFormat
- $KeyAttributeFont
- $KeyAttributeFontSize
- $KeyAttributes
- $LeftMargin
- $LineSpacing
- $NoSpellling
- $ParagraphSpacing
- $RightMargin
- $ShowTitle
- $SmartLinks
- $SmartQuotes
- $SyntaxHighlighting
- $Tabs
- $TextAlign
- $TextBackgroundColor
- $TextColor
- $TextColorBlue
- $TextColorGray
- $TextColorGreen
- $TextColorRed
- $TextFont
- $TextFontSize
- $TextHighlightBlue
- $TextHighlightGreen
- $TextHighlightMagenta
- $TextHighlightRed
- $TextHighlightYellow
- $TextPaneRatio
- $TextPaneWidth
- $TextSidebar
- $TextWindowHeight
- $TextWindowWidth
- $TitleBackgroundColor
- $TitleFont
- $TitleForegroundColor
- $Ziplinks

Textual Attributes

Relating to body text, text export settings and general text-related data. For legacy compatibility reasons, the group contains a hidden $Wordcount which maps to the current $WordCount attribute.

- $EstimatedNoteSize
- $ImageCount
- $IsAgent
- $ReadOnly
- $Text
- $TextExportTemplate
- $TextLength
- $UpdateTextLinksAfterRename
- $WordCount

Watch Attributes

This group holds a series of attributes to support the exchange of information with the web (and via that the iPhone/iPad) using Simplenote, DEVONThink, Evernote, Apple Notes, and Finder folders. In older versions, this group was called 'Simplenote' but was renamed to better reflect its scope of use.

This group also contains 12 hidden Scrivener-related attributes which are needed for under-the-hood synch requirements. Their exact roles are not documented and should not be of concern to most users. In TBX source XML they can be identified by the code string `visibleInEditor="0"`.

- $DEVONthinkGroup
- $DEVONthinkLabel
- $EvernoteNotebook
- $NotesFolder
- $NotesID
- $NotesModified
- $SimplenoteKey
- $SimplenoteModified
- $SimplenoteSync
- $SimplenoteTags
- $SimplenoteVersion
- $SourceURL
- $Tot
- $UUID
- $WatchFolder

Weblog Attributes

Data for configuring Weblog use.

- $mt_allow_comments
- $mt_allow_pings
- $mt_convert_breaks
- $mt_keywords
- $WeblogPostID

User Attributes

Any attributes added to a TBX by a user are added to the the 'User' group, which is empty by default.

Unusual attributes

These listings describe attributes whose behaviour may be unusual compared to the norm:

- Read-only system attributes
- Attributes inherited from Document Settings
- Attributes which are intrinsic

(What is does 'intrinsic' imply?)
There is also a full listing of all system attributes, although it omits the internal group (see above) as these are essentially off-limits to the user.

Read-only system attributes

N.B. This is not a grouping within Info view.
Attribute definition-level defaults for read/only attributes are not revealed by the Attributes window. However, the defaults can be revealed by double clicking the Attribute name, not value, in a note's information window.
A listing of system (built-in) attributes that may be read but not altered is given below. All other system attributes may be edited by the user.

- $AdornmentCount
- $Aliases
- $Associates
- $ChildCount
- $Created
- $Creator
- $DescendantCount
- $DisplayName
- $EstimatedNoteSize
- $HTMLExportPath
- $ID
- $IDString
- $ImageCount
- $InboundLinkCount
- $IsAdornment
- $IsAgent
- $IsAlias
- $IsComposite
- $LastFetched
- $LocalAttributes
- $Modified
- $NoteURL
- $OutboundLinkCount
- $OutlineDepth
- $OutlineOrder
- $Path
- $PlainLinkCount
- $ReadCount
- $ScreenHeight
- $ScreenWidth
- $SelectionCount
- $TextLength
- $TextLinkCount
- $WebLinkCount
- $WordCount

Attributes inherited from Document Settings

N.B. This is not a grouping within Info view.
A number of attributes are not defined with a default within the attributes dialog but instead draw their default from program or document level preferences. Attributes of this type are listed below.

- $AutomaticIndent
- $Creator
- $HTMLExportExtension
- $HTMLExportTemplate
- $InteriorScale
- $MapBackgroundAccentColor
- $MapBackgroundColor
- $MapBackgroundColor2
- $MapBackgroundPattern
- $MapPrototypeColor
- $NameFont
- $ParagraphSpacing
- $ShowTitle
- $SmartQuotes
- $TextAlign
- $TextBackgroundColor
- $TextColor
- $TextExportTemplate
- $TextFont
- $TextFontSize
- $TextSidebar
- $TitleBackgroundColor
- $TitleFont
- $TitleForegroundColor

Attributes which are intrinsic

Attributes that are intrinsic to all objects and thus not inherited, or shared, by aliases.

This list is created by a setting in the source XML of `canInherit="0"`, as explained in the section on TBX structure. That is not a formal identification; if in doubt about a prototype's intrinsic status check with Tech Support (not the forum - this is internal program information).

Attributes of this type are listed below.

- $Associates
- $CleanupAction
- $Container
- $Created
- $CreatedFrom
- $DisplayExpressionDisabled
- $EdictDisabled
- $Flags
- $Height
- $ID
- $IDString
- $InboundLinkCount
- $IsComposite
- $IsPrototype
- $LastFetched
- $LocalAttributes
- $MapScrollX
- $MapScrollY
- $Modified
- $Name
- $NeverComposite
- $OutboundLinkCount
- $OutlineDepth
- $OutlineOrder
- $Path
- $PlainLinkCount
- $RuleDisabled
- $Searchable
- $SelectionCount
- $SiblingOrder
- $TextLinkCount
- $TimelineBand
- $WebLinkCount
- $Width
- $Xpos
- $Ypos

Attributes grouped by purpose

This section groups attributes based on their purpose within Tinderbox. These grouping as as made by aTbRef as opposed to the app itself:

Action code related attributes

These attributes store data related to Actions or are used for testing out actions. The current listings here map each system attribute to a single purpose, though experienced users will see that some could be placed under several groupings. Notwithstanding the one-to-one mapping to groupings here is intended to help newer users find attributes related to a type of purpose or work:

- $AgentAction
- $AgentQuery
- $CreatedFrom
- $DisplayExpression
- $DisplayExpressionDisabled
- $Edict
- $EdictDisabled
- $HoverExpression
- $IsAction
- $OnAdd
- $OnRemove
- $Rule
- $RuleDisabled
- $TableExpression

Agent configuration attributes

These attributes are used to configure agents:

- $AgentCaseSensitive
- $AgentPriority
- $CleanupAction
- $Searchable

Calculated data attributes

These attributes hold calculated values and are generally read-only (see a full list of read-only system attributes):

- $AdornmentCount
- $Aliases
- $ChildCount
- $Created
- $Creator
- $DescendantCount
- $DisplayName
- $EstimatedNoteSize
- $ID
- $IDString
- $ImageCount
- $InboundLinkCount
- $IsAdornment
- $IsAgent
- $IsAlias
- $LastFetched
- $LocalAttributes
- $Modified
- $OutboundLinkCount
- $OutlineDepth
- $OutlineOrder
- $Path
- $PlainLinkCount
- $ReadCount
- $SelectionCount
- $SiblingOrder
- $TextLength
- $TextLinkCount
- $WebLinkCount
- $WordCount

Composite configuration attributes

These attributes relate to the configuration of composites:

- $Associates
- $IsComposite
- $IsMultiple
- $NeverComposite
- $OnJoin
- $Role

Data synch attributes

These attributes are used when synching data with other apps such as Bookends, Scrivener and Simplenote:

- $Abstract
- $AccessDate
- $ArticleTitle
- $Author2
- $Author3
- $Author4
- $Authors
- $BookTitle
- $CallNumber
- $DEVONthinkGroup
- $DEVONthinkLabel
- $DOI
- $Edition
- $EvernoteNotebook
- $ISBN
- $Issue
- $Journal
- $NotesFolder
- $NotesID
- $NotesModified
- $NoteURL
- $Pages
- $PublicationCity
- $PublicationYear
- $Publisher
- $ReferenceDictionary
- $ReferenceRIS
- $ReferenceTitle
- $ReferenceURL
- $RefFormat
- $RefKeywords
- $RefType
- $ScrivenerID
- $ScrivenerKeywords
- $ScrivenerLabel
- $ScrivenerLabelID
- $ScrivenerNote
- $ScrivenerStatus
- $ScrivenerStatusID
- $ScrivenerType
- $SimplenoteKey
- $SimplenoteModified
- $SimplenoteSync
- $SimplenoteTags
- $SimplenoteVersion
- $SourceCreated
- $SourceModifed
- $SourceURL
- $Tot
- $UpdateTextLinksAfterRename
- $UUID
- $Volume
- $WatchFolder

Experimental attributes

These are attributes adjudged by Eastgate to be 'experimental' in nature. Their purpose may change and they may not be retained in future versions:

- $IrisAngle
- $IrisRadius
- $LeafBase
- $LeafBend
- $LeafDirection
- $LeafTip

General data attributes

These attribute relate to miscellaneous general data about a note:

- $Checked
- $Container
- $File
- $HTMLFont
- $HTMLFontSize
- $Prototype
- $Tags
- $TextPaneRatio
- $TextPaneWidth

Export purposes

There are a number of export-related purposes:

- HTML export encoding attributes
- HTML export file configuration attributes
- HTML export mark-up attributes
- HTML export post-processing attributes
- HTML export scope attributes
- Text export file configuration attributes

HTML export encoding attributes

These attributes control the general processing of note data during HTML export, when using the ^text^ export code.

- $HTMLCodeEnd
- $HTMLCodeStart
- $HTMLEntities
- $HTMLExportTemplate
- $HTMLMarkupText
- $HTMLQuoteHTML

HTML export file configuration attributes

These attributes control the nature of the exported filename for a note.

- $EmailSubject
- $EmailTemplate
- $HTMLExportExtension
- $HTMLExportFileName
- $HTMLExportFileNameSpacer
- $HTMLExportPath
- $HTMLFileNameLowerCase
- $HTMLFileNameMaxLength

HTML export mark-up attributes

These attributes control the specific HTML (or other) mark-up applied to various parts of the $Text structure (e.g. paragraphs) and style (e.g. italics).

- $HTMLBoldEnd
- $HTMLBoldStart
- $HTMLCloud1End
- $HTMLCloud1Start
- $HTMLCloud2End
- $HTMLCloud2Start
- $HTMLCloud3End
- $HTMLCloud3Start
- $HTMLCloud4End
- $HTMLCloud4Start
- $HTMLCloud5End
- $HTMLCloud5Start
- $HTMLExportAfter
- $HTMLExportBefore
- $HTMLFirstParagraphEnd
- $HTMLFirstParagraphStart
- $HTMLImageEnd
- $HTMLImageStart
- $HTMLIndentedParagraphEnd
- $HTMLIndentedParagraphStart
- $HTMLItalicEnd
- $HTMLItalicStart
- $HTMLListEnd
- $HTMLListItemEnd
- $HTMLListItemStart
- $HTMLListStart
- $HTMLOrderedListEnd
- $HTMLOrderedListItemEnd
- $HTMLOrderedListItemStart
- $HTMLOrderedListStart
- $HTMLParagraphEnd
- $HTMLParagraphStart
- $HTMLStrikeEnd
- $HTMLStrikeStart
- $HTMLSubscriptEnd
- $HTMLSubscriptStart
- $HTMLSuperscriptEnd
- $HTMLSuperscriptStart
- $HTMLUnderlineEnd
- $HTMLUnderlineStart

HTML export post-processing attributes

These attributes control post-processing applied to exported notes.

- $HTMLExportCommand
- $HTMLLinkExtension
- $HTMLMarkdown
- $HTMLMarkDown
- $HTMLOverwriteImages
- $HTMLPreviewCommand

HTML export scope attributes

These attributes control whether a given note, or its children, are exported

- $HTMLDontExport
- $HTMLExportChildren

Text export file configuration attributes

Not used post v5, these attributes control legacy plain text export.

- $TextExportTemplate

Import configuration attributes

These attributes relate to AutoFetch and RSS use:

- $AutoFetch
- $AutoFetchCommand
- $RawData
- $RSSChannelTemplate
- $RSSItemLimit
- $RSSItemTemplate
- $URL
- $ViewInBrowser

Locational data attributes

These attributes relate to the geographical location of notes:

- $Address
- $City
- $Country
- $District
- $FormattedAddress
- $GeocodedAddress
- $Latitude
- $Longitude
- $PostalCode
- $Range
- $State

Map configuration attributes

These attributes control the general configuration of a map view:

- $AdornmentFont
- $ClusterTerms
- $InteriorScale
- $Lock
- $MapBackgroundAccentColor
- $MapBackgroundColor
- $MapBackgroundColor2
- $MapBackgroundFill
- $MapBackgroundFillOpacity
- $MapBackgroundPattern
- $MapBackgroundShadow
- $MapScrollX
- $MapScrollY
- $PlotBackgroundColor
- $PlotBackgroundOpacity
- $Sticky
- $Xpos
- $Ypos

Map item purposes

There are a large number of attributes relating to the look and layout of map items. Be aware that some of these, e.g. badges apply to other view types as well as map view:

- Map item badge configuration attributes
- Map item border configuration attributes
- Map item caption configuration attributes
- Map item general configuration attributes
- Map item grid configuration attributes
- Map item shadow configuration attributes
- Map item subtitle configuration attributes
- Map item text configuration attributes

Map item badge configuration attributes

These attributes control the configuration of an item's badge in a map view.

- $Badge
- $BadgeMonochrome
- $BadgeSize

Map item border configuration attributes

These attributes control the configuration of an item's border in a map view.

- $Border
- $BorderBevel
- $BorderColor
- $BorderDash

Map item caption configuration attributes

These attributes control the configuration of an item's caption in a map view.

- $Caption
- $CaptionAlignment
- $CaptionColor
- $CaptionFont
- $CaptionOpacity
- $CaptionSize

Map item general configuration attributes

These attributes control the general visual configuration of an item in a map view such a shape, colour, tables and plots.

- $AccentColor
- $Base
- $Bend
- $Color
- $Color2
- $Direction
- $Fill
- $FillOffsetY
- $FillOpacity
- $Flags
- $Height
- $HoverBackgroundColor
- $HoverFont
- $HoverImage
- $HoverOpacity
- $Opacity
- $Pattern
- $PlotColor
- $PlotColorList
- $Shape
- $TableHeading
- $Tip
- $Width

Map item grid configuration attributes

These attributes control the configuration of an adornment item's grid in a map view.

- GridColor
- GridColumns
- GridLabelFont
- GridLabels
- GridLabelSize
- GridOpacity
- GridRows

Map item shadow configuration attributes

These attributes control the configuration of an item's drop shadow in a map view.

- $Shadow
- $ShadowBlur
- $ShadowColor
- $ShadowDistance

Map item subtitle configuration attributes

These attributes control the configuration of an item's subtitle in a map view.

- $Subtitle
- $SubtitleColor
- $SubtitleOpacity
- $SubtitleSize

Map item text configuration attributes

These attributes control the configuration of the look of an item's title and text in a map view.

- $MapBodyTextColor
- $MapBodyTextSize
- $MapNameSize
- $MapTextSize
- $NameAlignment
- $NameBold
- $NameColor
- $NameFont
- $NameLeading
- $NameStrike
- $TitleHeight
- $TitleOpacity

Natural Language Processing attributes

These attributes hold data extracted using Natural Language Processing:

- $DominantLanguage
- $NLNames
- $NLOrganizations
- $NLPlaces
- $NLTags
- $Sentiment
- $Sentiments

Note Displayed Attributes attributes

These attributes relate to configuring note Displayed Attributes:

- $DisplayedAttributes
- $DisplayedAttributesDateFormat
- $DisplayedAttributesFont
- $DisplayedAttributesFontSize
- $HideDisplayedAttributes

Note Key Attributes attributes - replaced by Displayed Attributes

These attributes are deprecated as they are now totally replaced by the Displayed Attributes attributes with the same purpose.
These attributes relate to configuring note key attributes:

- $HideKeyAttributes
- $KeyAttributeDateFormat
- $KeyAttributeFont
- $KeyAttributeFontSize
- $KeyAttributes

Note text attributes

These attributes store data related to how text is displayed.

- $AutomaticIndent
- $CodeFont
- $ImageSizeLimit
- $LeftMargin
- $LineSpacing
- $NoSpellling
- $ParagraphSpacing
- $ReadOnly
- $RightMargin
- $ShowTitle
- $SmartLinks
- $SmartQuotes
- $SyntaxHighlighting
- $Tabs
- $Text
- $TextAlign
- $TextBackgroundColor
- $TextColor
- $TextColorBlue
- $TextColorGray
- $TextColorGreen
- $TextColorRed
- $TextFont
- $TextFontSize
- $TextHighlightBlue
- $TextHighlightGreen
- $TextHighlightMagenta
- $TextHighlightRed
- $TextHighlightYellow
- $TextSidebar
- $Ziplinks

Note title attributes

Attributes referring to the note title. Most of these only have legacy use.

- $HideTitle
- $Name
- $TitleBackgroundColor
- $TitleFont
- $TitleForegroundColor

Note window configuration attributes

Attributes referring to configuration of the pre-v6 note's text window. Most of these only have legacy use.

- $TextWindowHeight
- $TextWindowWidth

Outline configuration attributes

These attributes control the general configuration of an outline view:

- $IsSeparator
- $MapPrototypeColor
- $OutlineBackgroundColor
- $OutlineColorSwatch
- $OutlineNameSize
- $OutlineTextSize
- $PrototypeHighlightColor
- $Separator

Person detail attributes

These attributes store details relating to a person or organisation:

- $Email
- $FullName
- $Organization
- $Telephone
- $Twitter

Poster configuration attributes

These attributes store details relating to configuring posters. LINK TO POSTERS ARTICLE
These attributes are:

- $PosterCSS
- $PosterLabels
- $PosterSettings
- $PosterTemplate
- $PosterURL
- $PosterX
- $PosterY
- $PosterZoom
- $ScreenHeight
- $ScreenWidth

Sort configuration attributes

These attributes are used to configure container and agent sorting:

- $Sort
- $SortAlso
- $SortAlsoTransform
- $SortBackward
- $SortBackwardAlso
- $SortTransform

Special note type designator attributes

These attributes store data indicating a note is to be used in a non-typical way:

- $IsPrototype
- $IsTemplate
- $Private
- $PrototypeBequeathsChildren

Test Attributes

These attributes, from the system ' Sandbox' group provide attributes of varying type for rapid testing of code:

- $MyBoolean
- $MyColor
- $MyDate
- $MyDictionary
- $MyInterval
- $MyList
- $MyNumber
- $MySet
- $MyString

Timeline configuration attributes

These attributes control the general configuration of a timeline view:

- $TimelineAliases
- $TimelineBandLabelColor
- $TimelineBandLabelOpacity
- $TimelineBandLabels
- $TimelineColor
- $TimelineDescendants
- $TimelineEnd
- $TimelineEndAttribute
- $TimelineGridColor
- $TimelineMarker
- $TimelineScaleColor
- $TimelineScaleColor2
- $TimelineStart
- $TimelineStartAttribute

Timeline event configuration attributes

These attributes control the general configuration of event items in a timeline view:

- $DueDate
- $EndDate
- $StartDate
- $TimelineBand

Weblog configuration attributes

Not used post v5, these attributes are retained for legacy purposes:

- $mt_allow_comments
- $mt_allow_pings
- $mt_convert_breaks
- $mt_keywords
- $WeblogPostID

Deprecated attributes

These attributes are not longer used but may be present in documents created under older versions of Tinderbox:

- $AutomaticIndent
- $Color2
- $EvernoteNotebook
- $FormattedAddress
- $HideKeyAttributes
- $HTMLMarkDown
- $HTMLOverwriteImages
- $KeyAttributeDateFormat
- $KeyAttributeFont
- $KeyAttributeFontSize
- $KeyAttributes
- $LeafBase
- $LeafBend
- $LeafDirection
- $LeafTip
- $MapBackgroundColor2
- $MapPrototypeColor
- $MapTextSize
- $mt_allow_comments
- $mt_allow_pings
- $mt_convert_breaks
- $mt_keywords
- $OutlineColorSwatch
- $OutlineTextSize
- $RSSChannelTemplate
- $RSSItemLimit
- $RSSItemTemplate
- $Separator
- $ShowTitle
- $SimplenoteKey
- $SimplenoteModified
- $SimplenoteSync
- $SimplenoteTags
- $SimplenoteVersion
- $TextAlign
- $TextExportTemplate
- $TextPaneRatio
- $TextPaneWidth
- $TextSidebar
- $TitleBackgroundColor
- $TitleFont
- $TitleForegroundColor
- $WeblogPostID

Attributes settable via the UI

These attributes can be set/edited via UI elements such as the Inspector, Document Settings or view settings:

- $AccentColor
- $AgentAction
- $AgentPriority
- $AgentQuery
- $Badge
- $BadgeSize
- $Base
- $Bend
- $Border
- $BorderBevel
- $BorderColor
- $BorderDash
- $Caption
- $CaptionAlignment
- $CaptionColor
- $CaptionFont
- $CaptionOpacity
- $CaptionSize
- $Checked
- $Color
- $Color2
- $Direction
- $DisplayExpression
- $DisplayExpressionDisabled
- $EdictDisabled
- $GridColor
- $GridColumns
- $GridLabelFont
- $GridLabels
- $GridLabelSize
- $GridOpacity
- $GridRows
- $HideTitle
- $HoverExpression
- $HoverImage
- $HoverOpacity
- $HTMLBoldEnd
- $HTMLBoldStart
- $HTMLCodeEnd
- $HTMLCodeStart
- $HTMLDontExport
- $HTMLExportAfter
- $HTMLExportBefore
- $HTMLExportChildren
- $HTMLExportExtension
- $HTMLExportFileName
- $HTMLExportTemplate
- $HTMLFirstParagraphEnd

- $HTMLFirstParagraphStart
- $HTMLIndentedParagraphEnd
- $HTMLIndentedParagraphStart
- $HTMLItalicEnd
- $HTMLItalicStart
- $HTMLListEnd
- $HTMLListItemEnd
- $HTMLListItemStart
- $HTMLListStart
- $HTMLMarkupText
- $HTMLOrderedListEnd
- $HTMLOrderedListStart
- $HTMLParagraphEnd
- $HTMLParagraphStart
- $HTMLQuoteHTML
- $HTMLStrikeEnd
- $HTMLStrikeStart
- $HTMLUnderlineEnd
- $HTMLUnderlineStart
- $IsPrototype
- $IsSeparator
- $IsTemplate
- $LineSpacing
- $Lock
- $MapBackgroundAccentColor
- $MapBackgroundColor
- $MapBackgroundColor2
- $MapBackgroundFill
- $MapBackgroundFillOpacity
- $MapBackgroundPattern
- $MapNameSize
- $Name
- $NameAlignment
- $NameBold
- $NameColor
- $NameStrike
- $NoSpellling
- $OnAdd
- $OutlineNameSize
- $ParagraphSpacing
- $Pattern
- $PlotBackgroundColor
- $PlotBackgroundOpacity
- $PlotColor
- $Prototype
- $PrototypeHighlightColor
- $Rule
- $RuleDisabled
- $Separator
- $Shadow
- $ShadowBlur
- $ShadowColor
- $ShadowDistance
- $Shape
- $SmartLinks
- $SmartQuotes
- $Sort
- $SortAlso
- $SortAlsoTransform
- $SortBackward
- $SortBackwardAlso
- $SortTransform
- $Sticky
- $Subtitle
- $SubtitleColor
- $SubtitleOpacity
- $SubtitleSize
- $TableExpression
- $TableHeading
- $Text
- $TextColor
- $TextFont
- $TextFontSize
- $TimelineAliases
- $TimelineBandLabelColor
- $TimelineBandLabelOpacity
- $TimelineBandLabels
- $TimelineColor
- $TimelineDescendants
- $TimelineEnd
- $TimelineEndAttribute
- $TimelineScaleColor
- $TimelineStart
- $TimelineStartAttribute
- $Tip

Editing attribute values

A Tinderbox note consists of a large number of attributes. Every note possesses every attribute—both built-in (system) and user-created (user) attributes—even if not using it.

Every system attribute starts with the built-in default which becomes the default for the document for that attribute. Unless pre-configured otherwise, each attribute data has its own default value (0 for Number, 'never' for Date, ' `false`' for Boolean, etc.).

For user-created attributes, the default is set at creation time and will start with the data-type default unless changed by the user.

The easiest place to *review* all attribute values for a note is the attributes tab of the Get Info pop-over. This allows the user to select an attribute group and then scroll through a table of attribute names and values.

To edit attribute values, for the currently selected note, i.e. only one note, use any of:

- the Get Info/attributes tab. The edit mechanism is as described for the Displayed Attributes table (next bullet below).
- the note's Displayed Attributes table.
- the Quickstamp Inspector.
- any Action code via rules, edicts, stamps, agents, etc.

To edit attribute values, for selections of multiple items use any of:

- the Quickstamp Inspector.
- any Action code via rules, edicts, stamps, agents, etc.

Get Info & Displayed Attributes vs. Quickstamp vs. Actions

The first two (functionally the same table) make it easier to see/edit multiple attributes. Quickstamp show one attribute ate a time, though can still set any attribute but it has the upside of being able to act on the selection, thus one, or more notes.

Action code generally addresses a different context but can set as many attributes as required and any scope, depending on the code the user writes.

Stamps and Quickstamp have a link beyond the name as a stamp can be thought of as a pre-configured Quickstamp. Like a Quickstamp it acts on the current selection and is run once only per use (unlike a rule or edict).

Setting or resetting an attribute's default

What if the default value assigned to an attribute, system or user, is not what is desired? Can it be changed globally across the current TBX document so that all notes inheriting that attribute use a different value. Yes, it is possible by using view.

Set/add new value. Select any note, it does not matter which, and open the Document Inspector's System tab. Locate the attribute and type in the new value. If you have already altered the default and need to check the listing, either use aTbRef's listings or open a new document and via the same location in the Inspector to see he as-new default value. This can then be noted and used in the original document

CAUTION: take special care when editing system attribute defaults that you set a valid and sensible value, lest you cause the application and/or document to become unstable. The same holds true for altering the defaults for user attributes after first use; ensure no existing code depends on the original default value for correct function.

It is not possible to set attribute defaults via action code. This task can only be one manually via the above method or by direct editing of the XML code outside Tinderbox using a text editor.

Attributes - $ prefix notation

$Attribute notation

Although legacy syntax may still be supported, all calls to attributes—i.e. to their values—in Action code should use a '$' prefix to the attribute name.

```
$AttributeName=value
$AttributeName=$AttributeName
$AttributeName=$AttributeName(where)
$AttributeName(where)=$AttributeName(where)
```

Long term users must note that either/both the left or right-hand expressions must begin with a '$'.

Legacy issues

Since v4.6.0, Tinderbox has attempted to support legacy code usage alongside the new forms, but do not expect such support to continue. It is thus worth coming to an understanding of the different implications of $Attribute(path) and AttributeName(regex). Indeed, AttributeName(regex) is deprecated: use the current String/List/Set.contains("regexStr").

Examples of $Attribute(path) usage:

```
$Name(parent)="archives"
if($ChildCount(parent)>6)
```

Use of Agents

An agent is a persistent search query; it can also be thought of as a more powerful version of the Find view. The maximum scope of a query is every note (and agent, etc.) in the current TBX document. This holds true even if more than one TI is open in Tinderbox. The query's arguments then restrict the scope of matches within the overall document scope. Tinderbox has no mechanism for searching across multiple TBX documents.

An agent is a special note: it can have text, links, attributes, etc., per a normal note but its children can only ever be aliases. These aliases are those of anything meeting the criteria of a query stored in the special $AgentQuery attribute. Thus an agent normally acts as a container for aliases. Agent queries can match notes, (including containers), separator notes, aliases, other agents but not map adornments. Adornments are the one exception to an agent's ability to query any object in the document.

As well as gathering aliases, an Agent can act upon the aliased notes via the code stored in its $AgentAction attribute. As the query is run continuously, agents are also dynamic in their filtering. As notes alter they may change their match to the query and be added to or removed from the agent's child aliases. Whilst the agent's action code may cause a matched note's original to move within the underlying outline (e.g. to a new container), an agent can neither delete notes entirel nor create new. The latter is a deliberate safeguard against unintentional data loss.

Normally, attribute changes made to an alias changes the alias' original note. But, be aware that some attributes are ' intrinsic', meaning the original note may hold a different value for the attribute than that in any of its alias(es). With such attributes it may be necessary to use the 'original' designator to ensure the original note's value rather than the value of alias inside the agent is altered.

The 'agent' designator is also useful as it can be used in both the agent query and agent action to use a value stored in an attribute of agent. The allows either/both the query or action to be altered simply by editing attribute(s) in the agent itself. A useful manner of doing this is to make those attributes 'Displayed Attributes' in the agent.

Agents may have regularity of their update cycle varied, including the ability to turn an agent off entirely. In the latter case, this means that when turned off the agent retains whatever children (aliases) it had at the last update cycle before it was turned off. The agent's action is not run when an agent is switched off.

Agents may not be turned into notes, nor notes into agents: both must initially be created as the type of object desired. Agents may not have children other than aliases created by the agent query. An exception is that adornments may be added to agent maps.

An agent may be a separator ($IsSeparator). This option is set via the Properties Inspector's Prototype sub-tab. As a separator, an agent is not visible in Map view but is visible in all other major views (Chart, etc.).

Meanwhile, an agent can have note text, Displayed Attributes and other note features although an agent may be used simply as a filter to gather—and act upon—aliases.

As an agent's output is primarily the effect of its action on aliases, it does not matter where in a document the agent is placed. This is something new users often misunderstand. The choice of whether an agent is better placed amongst the 'content' of the document or hidden away in a separate branch of the outline will depend on the agent's role in the document. Viewing the source of this TBX, it will be seen some agents are within the main content—e.g. those re-stating the attribute listings via data type, etc.—and others which are hidden away in a separate branch of the outline. Indeed, when structuring a new TBX, it is often useful to place all 'content'—the data being worked on—as descendant from a single root level container, making it easy to hide away 'back-of-house' aspects like utility agents, export templates, prototypes, etc.

On export, agent aliases are exported as discrete files, albeit sharing their original's text and most attributes (except intrinsic ones).

Like a note, an agent can also use the $Rule and $Edict attributes allowing it to apply changes to itself as well as its contained aliases.

Queries match both actual notes as well as their aliases. A note satisfies a query if:

- the note itself satisfies the query

or

- any alias of the note satisfies the query

Regardless of whether the note or one of its aliases satisfies the query, the agent makes a new alias of the original note.

The agent's child map (i.e. the aliases) is auto-updated and laid out. The default use of a grids can be altered or turned off - see re-arrangeable agent maps.

Aspects of agent and query use

See:

- Agent & Queries
- Self-referring agents
- Aliases, children and descendants
- Controlling Agent Update Cycle Time
- Sorting Agent Results
- Manually triggering agent updates

Agent & Queries

Queries are action code expressions in a conditional form that must resolving to `true/false` statements. Agent queries result in the agent creating a list of aliases that will include each matching item once. Thus even if the original and/or or or more of its aliases are all within the scope of the query, that item will create *a single alias* in the agent. An exception is allowed, where it is needed to locate an original *and all its aliases*, by using a contains() query based on the item's name as this will match all containers of both that note and all of its alias(es).

Agents, including smart adornments, can match notes (including container notes), separator notes, aliases, other agents but can *not* match map adornments. Adornments are the one exception to an agent's ability to query any object in the document.

References in this section to "BooleanAttribute", "DateAttribute", etc., refer to any system or user attribute of the data type specified. Thus "BooleanAttribute" implies any attribute of the data type "Boolean".

Note that an agent's query only controls which notes are matched and whose aliases are added as children of the agent. Anything done to the (aliases of) matching notes is controlled via the the code stored in the AgentAction attribute. Action code can also be viewed and edited via the Action Inspector.

Queries are written in action code. In the very early days of Tinderbox there was a discrete syntax for queries, using operators with a # prefix. If such code is seen in old files or code examples, it should be re-written in existing action code before use.

Unlike action code expressions, *queries do not and should not use a terminating semi-colon* . As this may easily be done by mistake, Tinderbox will ignore a terminating semi-colon if found in a query.

Query Syntax

Queries are used by:

- agents
- action code conditional statements
- action find() operators (most normally for scoping arguments)

Queries are written in action code syntax but with some differences:

- Queries never use semi-colon terminators unlike in rules, etc.
- Equality tests (are two things the same?) always use a == operator (**two** equals signs).
  - Note: very old code examples may use a single '=' and if using copy paste do not forget to double the = to ==.
- If a query has multiple parts, these are joined using & (and) or | (or) join characters.

If it is desired to control the order in which different parts of a multi-term query are evaluated, each query term—or sub-groups of query—can be enclosed in parentheses, i.e. round brackets '(' and ')'. As with parentheses in a spreadsheet formula, Tinderbox will evaluate the inner-most (most deeply nested) parentheses before working outwards until the whole query is evaluated.

Action code offset addresses can be used, i.e. referencing values in another note `$Color` versus `$Color("Some other note")`. In the special case of wanting to reference attribute values in the agent itself, use the `agent` designator to refer to the attributes of the current agent, whilst smart adornments can similarly use the 'adornment' designator. For example:

Query: `$Status=="Important"`

Action: `$Color=$Color(agent)`

This agent gathers all "Important" notes and sets their colour to match the colour of the agent. Note that this is different from

Action: `$Color=$Color(this)`

since "this" refers to the note being gathered, and is also different from

Action: `$Color=$Color(parent)`

since "parent" is the parent of the original note, not the parent of the newly-made alias. The latter is a case where, for intrinsic attributes, it may be necessary to use the 'original' designator to avoid using the aliases value for the attribute. Thus:

Action: `$Color=$Color(parent(original))`

More on Queries

See:

- Simple Queries - equality and inequality
- String Attributes - comparison operators & case sensitivity
- Querying Lists and Sets
- Regular Expressions in queries
- Legacy Query Code for agents

Simple Queries - equality and inequality

A very common desire when starting out with agent queries test is a given attribute does—or does not—match a specific value. Here, 'match' implies equality, i.e. the two are exactly the same. The exactitude relates to subtleties like the fact that 'Ant' and 'ant' are not the same as one starts with an uppercase character. If new to using code and queries, take a moment to familiarise yourself with Tinderbox's comparison operators.

Thus to match all notes that have a colour of 'red', the query is:

`$Color=="red"`

Do **not** use this form:

~~`$Color="red"`~~

IMPORTANT: Tinderbox will accept a single '=' but only to stop code breaking in very old TBX documents some point this support will cease, so do not use the '=' form in queries and update old code.

Inequality is tested using '!=" (exclamation mark + equals sign):

`$Color!="red"`

Note the inequality test only used a *single* equals sign, i.e. do not use '!=='.

If woking with Date-type attributes and testing a specific date, it is best to use the date() operator to enclose the actual date string. This ensures Tinderbox turns the string into a Date-type object before doing the equality test:

- GOOD: `$StartDate==date("today")`
- GOOD: `$StartDate==date("24/11/1987")`
- BAD: ~~`$StartDate=="today"`~~
- BAD: ~~`$StartDate=="24/11/1987"`~~

String Attributes - comparison operators & case sensitivity

Comparison Operators

The comparison operators available are <, <=, ==, !=, >=, and >. The == and != operators always work as a case-sensitive comparison. Examples:

```
$Name > $AttribName
$AttribName > $AttribName(path)
$Created > $Created(parent)
$Name < $Text
$BorderColor == $Color
```

For case-sensitive matches, use String.contains("regex"), e.g. `$MyString.contains("dog")`, where the regex is a string literal or regular expression. Using String.icontains("regex") forces a case-*insensitive* match. The same applies to Lis and Sets, see also Querying Lists and Sets.

Regex matches (via both old and new style contains) are case-sensitive by default, but when used in the context of an agent this can be modified. This is done by setting the agent's $AgentCaseSensistive attribute to `false`.

Thus, a case-insensitive 'dog' regex could match all case variants of 'dog' (dog, Dog, DOG, etc.). However, as this would also match a value of 'big dog', the degree of match of the regex can be restricted further by regular expression string delimiters, e.g. `$MyString.contains("^dog$")`. The caret (^) forces matching to start at the beginning of the string; note that it has to be doubled so as to escape it for Tinderbox, i.e. tell the program the symbol is not, in this context, export mark-up code. Similarly, the $ forces the match to run all the way to the end of a value string. This therefore excludes longer strings that might contain the word in question. The use of the $ in a regular expression should not be confused with the $AttributeName usage, which simply references an attribute's value.

Querying Lists and Sets

Testing (querying) lists and sets

Testing lists/sets, the contains operator, syntax AttributeName.contains("regex"), returns *true* if a list/set contains an exact match for the the designated regex; the test is case sensitive regardless of agent query case sensitivity settings; regex matches are not supported. If user attribute `PetTypes` has a value of `[dogs;cats]` then either

`$PetTypes.contains("dogs")` is `true`,

but

`$PetTypes.contains("dog")` is `false`

as 'dog' is an incomplete value match.

It can be useful to use a stored value as the search regex, for instance using the name of an agent as the search term. However, the following will not work or an agent named 'dogs':

`$PetTypes.contains($Name)` WRONG!

Instead, use the *evaluated* result of the above via a rule or some other action code. Thus, in an agent's Rule box, enter:

`$AgentQuery='$PetTypes.contains("'+$Name+'")'`

…which results in a query of:

`$PetTypes.contains("dogs")`

See Set Attributes for more on Sets.

Regular Expressions in queries

Tinderbox regular expressions ('regex') recognise a number of patterns containing the backslash character. Some common examples are:

\s matches any whitespace character

\S matches anything EXCEPT a whitespace character

\w matches a word character, such as a letter

\W matches any non-word character, such as punctuation

Otherwise Tinderbox uses normal regular expression escaping (i.e. to double-escaped backslashes). For example:

```
"this\sthat"
```

recognises

```
"this&[tab char]&that"
```

as well as

```
"this   that"
```

Legacy Query Code for agents

This note is for legacy reference only as Query Code for writing agent queries was replaced from v4.6.0. Essentially, in early Tinderbox versions some operators were written in a different code style, essentially these uses a #-prefix. All such codes now have action code replacements (`#inside()` → `inside()`) so is unlikely to be found except in very olds code examples or TBX documents.

Query back-references in agents

Not all queries use regex, therefore not all queries can generate back-references. The only difference, relating to an agent, is the context of where the regular expression (regex) is input.

The more general issue of back-references is discussed under using regular expression back-references.

Querying for aliases - agents

Although the parent of an alias is its (outline) parent, as the image shows, it is not always interpreted as such. In addition, queries like inside() and descendedFrom() find aliases in 'other' containers besides their original's because the *original note* lies within scope of the query.

Note that use of the 'parent' designator can be used to filter queries rendering apparent false positives. The `$Name(parent)` for alias and original will differ *unless* both are in the same container, as is often the case when an alias is freshly created and not yet moved to its intended location. Be aware though that alias and original share the same $DisplayExpression so this is not a good a good way to investigate the difference.

All children of an agent are aliases regardless of whether they point to an original or an alias.

$IsAlias cannot be used on its own, can help find only original notes:

```
inside(Some Container) & !$IsAlias
```

To find aliases as opposed to originals, it may be insufficient to use the reverse:

```
inside(Some Container) & $IsAlias
```

Why? Because an agent's children are also aliases, although not user-created ones. In order to filter out aliases within agents an extra query term is required:

```
inside(someContainer) & $IsAlias & !$AgentQuery(parent)
```

As agents de-dupe listings, the above methods cannot be used to locate multiple aliases of the same note. However, it is possible to locate their containers as long as the latter are themselves not all named the same. Use:

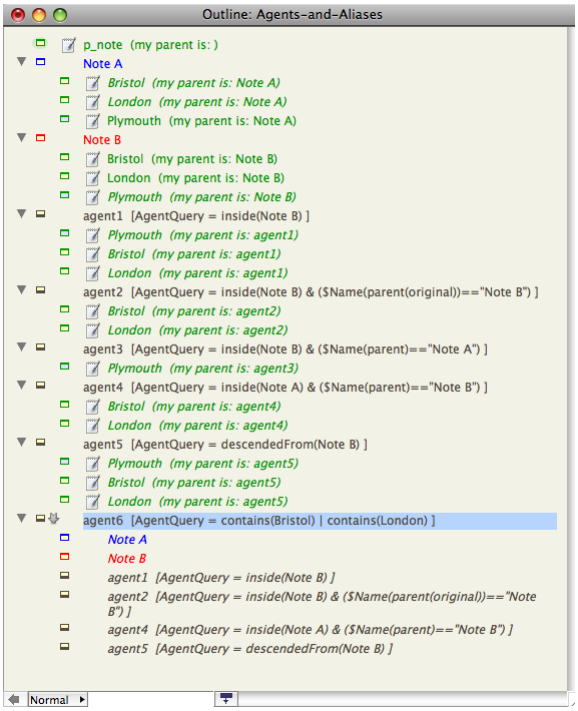
```
!$AgentQuery & any(child,$IsAlias)
```

Clause one discards any containers that are agents (because they will have a query value). The second clause matches any container where one or more children are aliases. To filter this for just aliases of a known note title, try:

```
!$AgentQuery & any(children,$IsAlias) & any(children,$Name=="Other Attribute types")
```

There is no 'IsAgent' test but only agent will have a value for $AgentQuery so it works in this context. Whilst it is possible to make an agent and not give it a query, such an agent cannot have any children (no query to match against) and so would not affect the above query.

It may be easier to use find() for tracking down aliases.



Querying for aliases - find()

The find() operator can be used to run a query, and produces a *list* of Paths ($Path data). Being a list it is thus not de-duped like an agent's results.

It is important to note this method does not use an agent query but instead its output must be passed to a list attribute or formatted into a string. Thus matches cannot be seen in a Map or Outline view as with an agent. The most likely form of use is thus as a Rule in a note (though an agent Rule will work just as well).

Thus for note 'Test', this will return a list of the paths to each alias of Test:

```
find($IsAlias & $Name=="Test")
```

Want the aliases of all notes whose name contains the string "Test"?

```
find($IsAlias & $Name.contains("Test"))
```

The above will match names like "Test", "My Test" as well a "Tester" but not "Intestate". By using .icontains() for case insensitive matching, "intestate" would be found.

```
find($IsAlias & $Name.icontains("Test"))
```

Want to exclude aliases that have 'Exam' in their path?

```
find(!$Path.contains("Exam") & $IsAlias & $Name.contains("Test"))
```

Want the original find to exclude aliases that are agents' contents?

```
find($IsAlias & $Name.contains("Test") & !$AgentQuery(parent))
```

Want to display the resulting list as line-by-line entries in $Text?

```
$Text = find($IsAlias & $Name.contains("Test")).format("\n")
```

Fine-tuning query search results using $Searchable

It is possible to exclude individual notes (and agents) by setting an item-scope Boolean $Searchable. By setting a `false` value, that item is excluded from searches even if otherwise matching search criteria. This makes it easier to exclude behind-the-scenes notes like templates, prototypes, and general framework containers for normal searches whist allowing content to be intermingled. Previously, such items were excluded by storing them away from primary content in a different outline branch or map.

The attribute affects notes, containers, agents and aliases (see below). Adornments are *always* excluded, regardless of the $Searchable setting.

The attribute is intrinsic so may be set separately for aliases, for instance ensuring a particular alias is matched rather than its original.

Unparse-able queries

If the query string does not parse the agent's query will be disabled and an item added to the error list. Once the query if fixed (or deleted) the item will clear from the list.

Date Comparisons - Date vs. Date-time

All Tinderbox Date-type data includes a time component in hours and minutes (no seconds or smaller). The time can be a confusing factor when comparing dates as often in the mind of the user whole days are being compared. Also, when using date designators, like "today" or "tomorrow + 1 month", the Date object created takes the OS' *current* clock time.

Thus at local time 13:00 on 28 November 2012, 'today' will return a value of 28 November 2012 13:00. A minute later it will return a value of 28 November 2012 13:01.

The time element is often overlooked when doing date-based arithmetic, especially for jobs like agents designed to find something like "all unfinished tasks in the next week". The natural tendency is to use a date (part of the) query like so:

```
$DueDate > date("today + 7 days") & $DueDate < date("today + 14 days")
```

However, for > and < the time element is counted. Thus, if 'today' was 28 November 2012 12:00, any items with a $DueDate before 12:00 on that day would not match the query. This is likely not what a new user would intend. To avoid the time issue with > and < comparisons where the time element is to be ignored there are two strategies:

- adjust the number of offset days and use >= or <= instead.
- ensure the reference date ('today' or whatever) has its time element correctly set.

Alternatively, to find past (or future dates) that only coincide (or not!) with the current calendar day, a two-part query is needed. Equality test operators (== and !=) and those including an equality test component (>= and <=) ignore the time element and test only for the whole calendar day. The latter effectively combine the less/greater test *and* an equality test, e.g. ((A > B) | (A ==)), the first of which is time sensitive and latter not. So, if the tested date time is 12:00 on 15 Jan 2013 >= will find all date-times before 00:00 on 16 Jan 2013; <= will find all dates after 00: on 15 Jan 2013. Neither of these is likely what the user intended. To find overdue $DueDate that are only on the current calendar day:

```
$DueDate < date("today") & $DueDate == date("today ")
```

Conversely to get the same but where the overdue notes are not on the current calendar day:

```
$DueDate < date("today") & $DueDate != date("today ")
```

Setting the time element of a date is discussed in more detail here.

Agents and intrinsic attributes

Some attributes are intrinsic to aliases. For instance, this allows an alias to have a different $Xpos and $Ypos from its original and thus a different location on the agent's map to that of the aliases' original.

In an agent action, it may sometimes be necessary to use the ' original' designator to set the original note's attribute and not the value of the same which is intrinsic to the alias. In an agent action:

```
$MyString = "hello"
```

…sets $MyString both original and alias as the user attribute is not intrinsic. But:

```
$Xpos = 4;
```

…would set only the $Xpos of the alias, whilst this would set only the original's value:

```
$Xpos(original) = 4;
```

Either of the following would set both:

```
$Xpos = 4;$Xpos(original) = 4;
$Xpos = 4;$Xpos(original) = $Xpos;
```

See a list of intrinsic attributes.

Self-referring agents

It can be useful for an agent's query or action to refer to the agent itself. This is especially helpful if creating generic agents where the same query/action needs to be applied many times in different contexts.

The key to reining this process is the ' agent1' designator can be used within the agent to refer to its own attributes. The 'agent' designator is understood only by agents. Note: smart adornments must use the ' adornment' designator instead 'agent' but otherwise in the same fashion.

The designator allows things like:

Query: `$Tags.contains($Name(agent))`

This will match all notes where the $Tags attribute matches the $name of the agent. For Lists/Sets the match is to a whole exact list value. For a String, the match could be to a sub-string. For an exact match in either case, but especially String attributes:

Query: `$Tag == $Name(agent)`

Now, if the agent's title is changed the query is automatically adjusted. The same trick can be used in the agent's action or rule. Here the $LessonNumber in a matched note is set to the value of the same attribute in the agent:

Action: `$LessonNumber = $LessonNumber(agent)`

The designator can be used on the left side of an action. Here the matched notes $Author2 value is added to the agent's $Authors Set attribute:

Action: `$Authors(agent) = $Authors(agent) + $Author2`

In actions and rule, as with the query, the 'agent' designator passes the current value of the cited attribute so changing the source attribute value will automatically affect the result of the agent. Often the need is to run the agent action only once, e.g. once per newly added note. In that case, consider altering action and/or query such that the action causes the acted on notes to no longer match the query. There are some other techniques.

If the agent queries are going to use inside(), descendedFrom() or test note $Name, then it may be better not using use $Name(agent) to see the process and instead add a user attribute to hold the customisable value. This avoids the scenario for value A seeing the agent for value B and so on, with unexpected consequences.

Indeed, the more clever the automation is made here, the more it makes sense to test the process on a copy of your date before committing it to the master unless you overlooked a flaw in your logic. Writing a badly thought-out agent can do a lot of edits very fast that can take much longer to unwind as there is no simple 'undo' for agent actions (unless the action's code logic allows for this)

Using a prototype

As well as a re-configurable single agent, the 'agent' designator also offers a useful way to make a prototype agent that can be set up quickly. Creating and using prototypes with agents is the same as with notes, though care needs to be taken not to run code in the prototype or to start altering a new agent's setting before the prototype is applied. Side note: Tinderbox actions cannot create new agents (nor notes) and there is no shortcut to create an agent: use the menus or context menus.

Thus:

Query: `descendedFrom($Name(agent))`

To avoid the agent miscuing, it is better to avoid the apparent convenience of $Name and to use a user attribute, which is then accessed and edited via a Displayed Attribute. Thus:

Query: `descendedFrom($TargetValue(agent))`

In the latter, the prototype agent's $TargetValue is deliberately left blank, to ensure noting is matched and no actions occur by mistake. The prototype has $TargetValue set as a Displayed Attribute. To use the system make a new agent, apply the prototype, open the agent text window and set its $TargetValue to the name of the container you wish to analyse. This way a generic agent with many attributes/actions can be pre-made and re-used many times.

For prototype smart adornments, do the same except use the ' adornment' designator in the same way as agents use the 'agent' designator.

Another approach to avoid premature execution in the prototype is first set $RuleDisabled to `true`. Leave the query blank. Then set this rule in the prototype:

Rule: `$AgentQuery|="descendedFrom("+$Name+")";$Rule="";`

This time we do not need the 'agent' designator the current note context is already the agent. It is important that $RuleDisabled is set before adding the prototype's rule as the latter must not fire except in agents using the prototype. In this example the $Rule also sets itself to nothing—if it was fully reset—it would, of course, re-inherit the prototype's code each cycle. The point of the rule re-writing itself is so the query is set just the once, just as a simple exercise in avoiding running code more than needed, if the query was set anew each cycle it would not really matter. This technique no use if the intent is to use a Displayed Attribute rather than $Name for the source value as by the time the attribute is set the query is already set. The above rule also uses a "|=" assignment so the rule runs just once to set the query. Otherwise every agent cycle the query would get re-set and never get to run as the query gets used the next cycle after being set, before which the rule would reset it.

However, the latter approach does allow for a more complex setting where the query regex must be derived from an expression using several attributes:

Rule: `if(!$IsPrototype){$AgentQuery|="descendedFrom("+$TargetValue+" - " $TargetValue(Other note)+")"}`

If wanting to set OnAdd code that uses a literal value of an agent's attribute, then the prototype method is needed.

If wanting to set the query/action just the once, besides using |=, you may consider using a self-cancelling code. However, you are better using a "" (empty string) reset than resetting inheritance. Otherwise, the note re-inherits from its prototype in a never-ending loop. In the latter case the process will not run away but code will re-run each agent cycle.

As noted above, the more inventive the customisation used the more important it becomes, especially for less experienced, to test before letting the process on a real data file. Better to discover any mistakes in test!

Aliases, children and descendants

Although aliases do not have children or descendants of their own, an alias in action code (such as queries) can act as a proxy reference to such descendants. Previously, is was necessary to write `every(descendant(original),$Checked` whereas now the same can be queried, via an alias in scope by using `every(descendant(,$Checked).`

In an agent query, we may want to search for notes whose children or descendants share a property:

Query: $ChildCount>0 & every(descendant,$Checked)

This will find all notes that have a child, for which every descendant is checked.

In previous versions, it was necessary to reference the original of the alias, since the query applies to an alias, and aliases have no children. An alias' children are, for the purposes of actions, the children of its original note, and its descendants are the descendants of the original note. By contrast, this does **not** mean that aliases can have their own discrete children/descendants.

Controlling Agent Update Cycle Time

To check your current agent cycle time and number of agents use the Tinderbox Inspector, Agents & Rules tab.

Controlling Agent Priority

The $AgentPriority attribute, allows the user to set the periodicity with which agents are run. This attribute can be set via;

- Get Info, agent tab.
- Get Info, attributes tab, General group.
- Displayed Attributes, if $AgentPriority is added.
- Properties Inspector, Quickstamp tab.
- Action Inspector, Query tab.
- Action code.

This attribute allows agents with queries tracking low rate of change to be run less often, reducing the overall agent cycle time.

Although $AgentPriority can be set (via code) to a chosen number, a subset of values map to labels used in pop-up menus in the UI:

- Highest = `0` . Runs every few seconds.
- Normal = `1` (default). Runs roughly every ten seconds.
- Low = `4`. Runs roughly every minute.
- Lowest = `10`. Runs roughly every five minutes'
- Occasional = `20`. Runs roughly every thirty minutes.
- Off = `-1`. Does not run (retains any existing aliases)

When set to 'Off' (`-1`), Tinderbox 'freezes' the contents of an agent and it does not run again; this reduces the overall cycle time. It means that when turned off, the agent retains whatever children (i.e. child aliases) it had at the last update cycle before it was turned off. Agents switched 'off' do not get updated via the file menu's 'Update now' command. It is thus not possible to have an agent that is only 'manually' updated. When an agent's query is 'off' (`-1`):

- Query does not run
- Agent add/remove actions ($AgentAction, $OnRemove) do not run.
- Agent rules and edicts still run, i.e. in the agent itself
- Child aliases' rules and edicts still run.

This means that agents can be used to harvest sets of aliases and then turned off to 'fix' the content. Alternatively, such agents' priority can be turned on/up to update content before an export and then turned off/downTbRef uses this methoc for agents used to do tasks such as representing the main attribute list in alternate forms such as by data type or TBX system attribute group.

All agents that are not 'off' run once on document load or a force refresh of agents.

To avoid having to change each agent by hand, they are located by an 'utility' agent '@attributes_agents_frequency' with this query:

```
inside(Attribute Data Types) | inside(Attribute Groups)
```

An action sets priority for the matches found (which here will be agents):

```
$AgentPriority=10
```

Note the $AgentPriority **must** be set using a numerical value. Using 'Low' instead of '10' results in—due to type coercion—a value of 'Highest' and not quite what's wanted. In turn, this utility agent is turned off when not in use. When enabled, it allows 18 agents to be controlled via one setting.

Although the Agents & Rules Inspector tab will tell you how many agents you have, you may be sure *where* they are in your TBX. An agent can help using this query:

```
$AgentQuery!=""
```

Notes, adornments, etc., will not have an $AgentPriority value. More precisely, adornments *may* have an $AgentPriority but only if they are 'smart' adornments.

Now you have all the agents you can use use the Show Original command to locate the agent. To aid more granular control of separate groups of agents, consider their location in the TBX. For example, in aTbRef the exported content, with the exception of the RSS feeds is all in one note hierarchy 9outline branch). A separate root-level note 'UTILS' contains, sets of prototypes and utility agents. Meanwhile both the Attribute and Export Code section use agents to present their lists in different splits; because the agents are within a containing (note) the #inside() query syntax can be used to easily locate them.

If your agents are more scattered in the TBX consider using a user-set attribute to flag agents so an agent may easily find them. However location based queries (as above) are more efficient, for reason discussed further below.

Altering Content of Disabled Agents

When switched off, an agent retains the aliases it had at that point. You can also do several things you cannot do when the agent is active:

- Re-order the aliases, e.g. alter the outline order or map location
- Delete aliases without them being recreated (N.B. you cannot add new notes aliases)

Note that re-enabling the agent may undo some of the above order/location changes and may add back deleted aliases.

If you need to alter the layout of an *active* agent, make sure to set $CleanupAction to any non-default value.

Making Agents More Efficient

In small TBXs you will not notice your agent settings affecting update time but as a TBX grows in size and complexity, ideally your agents should match as few notes as possible, or at least where that is possible to do so, in order to keep performance snappy. In this context, think about using inside() and descendedFrom() as part of your queries. The former allows you specify only the immediate children of a given note whilst the latter matches any descendant of a given note. For example, if searching for text in children of a note called 'The Library', it is *much* faster to run:

```
inside("The Library)" & $Text.contains("When.+April")
```

…than…

```
$Text.contains("When.+April") & inside("The Library")
```

…because the inside() part of the query cuts the number of notes searched. The second example does a regular expression search against *all* notes in the TBX before the results are then filtered to only children of 'The Library).

It is much faster, in performance terms, to factor out very complex queries into separate agents, and then use inside() to leverage those agents, than it is to repeat the queries. Complex regular expressions are also comparatively heavy users c TB thinking time so consider if there is not an easier way to find the same match. For example, if using such an expression to find misspellings, you might delete the agent when done or at least turn it off between major editing sessions.

As a general point, regex (regular expression) based queries like .contains(0 and .icontains() are far more CPU-intensive than many other actions.

Check That Results Make Sense

Finally, beware of broken agents. Sometimes, a syntax error can create an agent that finds nothing (or the wrong things), but takes an age to do so.

Disabling Automatic Updates

Trying to do text editing and getting disturbed by the hourglass? In a big TBX with many agents this can happen. In such circumstances it can help to disable the " Update Agents Automatically" option (via the File menu), just do not forget t enable it again if your document relies on ongoing use of live agents.

Flushing All Agents

File menu ▸ Update Agents Now runs *all* 'on' agents once. It also forces the process in front of other task, like running rules. This can be useful if a document has many agents and rules and you wish to update agents more quickly, e.g. to reflect a change you know you have made. But…

Cascading Agents

If you either do not use (or have temporarily turned off) automatic agent updates, you can use the same (File) menu to force a one-off update via "Update now" (cmd+ctrl+=). However, if you use a lot of agents and have different priorities set can be a wise idea to use this update call *several* times in succession to ensure all agents are fired, as those agents with lower priority settings are omitted from some firing cycles (i.e. update); the lower the priority setting, the more often an agent is omitted from the cycle. For instance, for the aTbRef TBX it can take 3 or 4 successive "Update now" calls to ensure all the low priority agents are run. Failure to do so means that sections like the listings of attributes by data type do not get populated, given that the HTML output expects each alias to be exported as a separate HTML file. If the TBX is not properly updated after being opened, some sections (agent containers) my be empty until enough update cycles has completed to populate even the lowest priority ones.

However, Tinderbox is aware of the latter issue and when doing an HTML export, Tinderbox should update *all* agents *regardless of priority* before export (including agents set to 'Off') but it is still worth checking output to ensure that output depending on several agent's actions has actually been generated. Once happy the process is working as expected, you can ease back on such checks.

Put another way, when you get creative with agents and start to create more complex TBXs, do not just check for the obvious output but look for unintended consequences: either of your query/action/rule coding or simply the inheritances an agent activities you have put in place.

Sorting Agent Results

By default, agents have no specific sort order for their matches.

The user may impose a deliberate single or two-level sort using criteria set via the Sorting group of system attributes, most easily accessed via the Action Inspector's Sort tab. Other options include the agent's Get Info or by setting the attributes via action code.

There is no document-level setting for agent sort order. If it is desired to use a sort order across the TBX, there are several possible solutions:

- Make an agent:
  - Query: `$AgentQuery & !$Sort`
  - Action: `$Sort="OutlineOrder";` or "Name", etc.
- Make an agent, set just the sort order and make it a prototype.
- Use the action from above as a stamp, and use the stamp on any agent as required.

When sorting by $OutlineOrder, the original note's value is used, regardless of whether the match is to an original or an alias. This gives speedier response whilst generally giving an acceptable sort order.

Sorting the contents of agents poses an extra problem to that of normal containers. This is because agents collect aliases of notes and in some cases the alias does not share attribute values with its original note, e.g. for $Width and $Height, the intrinsic attributes of an alias.

Thus setting $Sort (and $SortAlso) for a given attribute may not always have the intended consequences. The sort transform pop-up menu has the option 'original note'. Setting this transform tells Tinderbox to use the original note's value instead of the aliases, should the two differ. The 'original note' transform is useful if you want the aliases to sort in the outline order of their original notes.

Manually triggering agent updates

By default, new Tinderbox documents are set to update all agents automatically. In the vast majority of cases this is what the user wants. All agents run all the time, unless they have had their $AgentPriority altered via the agent's Action Inspector Query sub-tab or directly in action code.

In addition, you can trigger an immediate 'Update now' from the File menu. This runs **all** agents once, regardless of $AgentPriority **except** for any agents expressly turned 'off'' ($AgentPriority set to -1).

In addition, when Exporting to HTML or Exporting to Text (File menu), this invokes a once-only update of all running agents.

Detecting when manual 'Update now' is complete. In a very large or complex document with a lot of data and agents, where agent cycle time is a bit longer, it can be difficult to know when the full cycle is complete. A giveaway is that wher the File menu's 'Update now' option is clicked, the menu clears but the 'File' caption remains highlighted in the main menu bar. Once the process is complete, the highlight disappears.

Considerations when using agents that query other agents. If you have a complex set-up with agents looking at the contents of another agents, it may take several manual 'Update now' calls to flush a change through all the agents involved. How many calls is down to the user knowing how they constructed their code. For the same reason, when exporting, it may be expedient to either leave the document for a while to update fully or make several manual updates before starting as the full export process only updates agents once.

Action Code

An action is an automatic way of setting a certain attribute to a certain value (with the exception of system read-only attributes which cannot be altered).

Here 'note' may be read to mean any note-like object (note, agent, adornment, etc.). The term 'rule' is just one form of applying an action and relates to the scope of its use. Actions can be invoked a number of ways:

- An OnAdd action is performed by a container upon a note added to it, or by an adornment to a note placed upon it. A note's OnAdd action code is stored in its $OnAdd attribute. It can also be easily be viewed or edited via the Actio tab of the Action Inspector. An OnAdd applies only to direct new children and thus not their children (i.e. other descendants).
- An Agent Action is performed by an agent1 upon any alias of a note it creates as a result of a match to its query. An agent's AgentAction action code is stored in its $AgentAction attribute. It can also be easily be viewed or edited via t Action tab of the Action Inspector. Note: agents have no OnAdd and conversely notes have no Agent Action, thus the Action Inspector can show both in the same UI.
- An OnRemove action is performed by a container upon a note removed from it, or by an adornment on a note moved off it. This can be seen as the opposite of an OnAdd action. A note's OnRemove action code is stored in its $OnRemove attribute. It can also easily be viewed or edited via the Remove tab of the Action Inspector.
- An OnJoin action is performed by a member of a composite, once, upon any item joining the composite (or being moved within it) so as to touch the item with the OnJoin action. A note's OnJoin action code is stored in its $OnJoin attribute.
- A Rule is performed by a note or agent upon itself its Rule is run; this happens from time to time, typically every few seconds. A note's Rule action code is stored in its $Rule attribute. It can also easily be viewed or edited via the Rule tab of the Action Inspector.
- An Edict is performed by a note or agent upon itself its Edict is run; this happens from time to time, typically about once an hour. A note's Edict action code is stored in its $Edict attribute. It can also easily be viewed or edited via the Edict tab of the Action Inspector.
- A Stamp is a pre-saved action that is run once on each selected note or agent when selected from the Stamps menu or applied via the Stamps Inspector. Stamps are saved in a Tinderbox document and edited via the Stamps Inspector.
- A function is a pre-saved set of actions that can be called from any action, allowing a complex process to be reused without having to duplicate a lot of code. Unlike a stamp, a function can take input values that are used by the function to generate its output.
- A Quickstamp is a method for applying an unsaved stamp action, and is run once on each selected note or agent applied via the Quickstamp Inspector. Quickstamps are ad hoc use of code and not saved after use. For re-usable cod use a Stamp (above) instead.

*An ^action(some_code)^ export code allows inclusion of action code in an export template. The action code input some_code is executed when the template is evaluated. Attributes storing action events. View a full listing of attributes stori actions. Case sensitivity: Action code operators are always case-sensitive (unlike export codes). Agents can have actions. An agent performs its action on aliases of all the notes it finds that match its agent query. Making/deleting notes: See create(), createAgent() and delete(). Syntax: discussed under basic action code syntax. Note the point above re case-sensitivity. Originally, Tinderbox had ' actions' that notes applied to them elves or their children. These were written in 'Actic Code' though at the time what is now 'Export Code' was generally used for internal macros (actions). In v4 and v5 this mix was clarified: Export Code moved to being for export only whilst all internal 'coding' was done using 'Action Code', i.e. the style of code used for writing actions. Action code is now the most likely, indeed only, form of automation users are likely to meet unless they use export features to pass information out of Tinderbox. Action Code is not a formal coding language (programmers, take note!), and being artisanal software has grown as needed: it has what it needs or its users has asked for. It may lack some features that those with a programming back ground might assume are 'standard'. Further discussion of use of actions:

- Actions
- Coloured syntax highlighting in Action code
- Basic action code syntax
- Expressions in Action code
- Basic Comparison Operators
- Compound Actions
- Conditional statements using multiple arguments
- Defining an 'item' object—a single item—in action code
- Defining 'group' list objects—one or more items—in action code
- Why is a text 'line' in action code actually a paragraph?
- Explicit declaration of lists using square brackets
- Explicit declaration of dictionaries using curly braces
- Punctuation and special characters in definitions, actions and expressions
- Designators in actions
- Designators
- Comments in Action code
- Action code operator terminology explained
- Variables
- Functions
- Operators
- Problematic Characters for Action code in $Name and $Path
- Conditional Actions (if clauses)
- Counting characters in strings
- Getting and setting attribute values and inheritance
- Delaying code execution in prototypes and notes using prototypes
- Stream Processing and parsing
- Using attributes as global variables or constants
- Chaining 'dot' operators
- Look-up tables
- Actions, Stamps and Quickstamps
- Self-Cancelling Rules & Actions
- Using regular expression back-references
- Using long sections of code—code notes
- Using .each() for loops
- Constructing $Attribute references in loops
- Concatenation versus addition
- Parentheses as a guide to code execution
- Updates and cascading actions
- Optimising code for performance
- Checking and setting Time correctly in Date data
- Debugging user-written Action code
- Getting section numbering via Action code
- Moving notes by setting $Container
- Strings vs. StyledStrings in operator documentation
- Type coercion, strings and numbers

Actions

Both notes and agents can have actions. Actions can be applied to self via a rule or edict, or be applied to new/newly add:

- note(s), including container(s)
- map adornments
- agents and their child aliases
  - map smart adornments (action is applied to items 'on' the adornment)

In their original design use, the parent (with the action) acts on its children. In this regard a note/container $OnAdd is the functional equivalent of $AgentAction. Indeed, the Action Inspector's 'Action' tab shows the $AgentAction code for a selected agent or the $OnAdd code for a note.

A special form of Action called a Rule can be used to act just on the note/agent itself. Values of attributes are changed and information is gathered or changed as a result; the value of (system) read-only attributes may not be altered by action code. Any resulting effects are applied only to the children. Rules are actions that only affect the note itself.

Agents may have actions.

All code-editing text fields use the default value of $CodeFont for the current document.

Actions and rules may use simple arithmetic and logical expressions, written in action code as expressions. A broad range of functions can be run at action/rule scope often removing the need to use export codes.

But there are a number of action code contexts: see a listing of Action-type system attributes.

A further, easily overlooked context for action use is link actions.

The core build-in types of action are:

- OnAdd & Agent actions
- Rules
- Edicts
- Stamps

OnAdd & Agent actions

All containers and adornments apply an 'OnAdd' action (stored in $OnAdd) to items added to them. The action fires once only, on addition. If a note is removed from a container and then added back again, the OnAdd would fire again.

Agents have the same mechanism for their child aliases, except is it is terms the agent action and stored in $AgentAction. Thus agents do not use $OnAdd, any code stored there is never used by the agent.

The action can be any valid action code expression(s). Multiple expressions need a semi-colon delimiter between them.

The Action Inspector's Action tab's code is automatically stored in $AgentAction for agents and $OnAdd for all other types of note object.

Rules

A Rule is an action, much like an agent action, but while agents act on other notes a Rule acts on it own note (and only that note). If a rule is desired that only runs occasionally, consider using an edict instead

Rules are actions attached to notes and containers. Agents may have rules.

A note or agent's rule is stored in $Rule.

Rules are run periodically, and change the note's attributes. For example, suppose we have a Boolean attribute called $Urgent. A note might have a rule

```
$Urgent=any(children,$Urgent)
```

…meaning, "this note is urgent if any of its immediate children are urgent."

Another rule might be

```
$Urgent=$Urgent(parent)
```

…meaning, "this note is urgent if its parent is urgent."

Given that rules run periodically, Tinderbox will eventually enforce each rule, although there may be a delay between a change made to one note and its impact throughout the document depending on the agent update (cycle) time, i.e. in complex TBXs the update time may be a little slower.

Notice, too, that contradictory rules can be written; this may lead to constant changes in the document and should be avoided.

Agents can have rules and this is especially useful for highlighting agents under specified circumstances (e.g. if $UrgentTasks is not empty, or if $ThingsToDo find more than seven tasks in your inbox).

Like actions, rules may use simple arithmetic and logical expressions as well as action code functions.

If there are a lot of rules running, especially if complex ones, it is a good idea to use a logical OR (|=) rather than a normal assignment (=). The |= method ensures the rule is only run if the left side equates as empty. An alternate solution is to use a Boolean 'guard' attribute and an if() test to ensure the running of the rule is controlled. This is a specimen such Rule:

```
if($ShouldRun){$SomeAttribute=eval([some expression]);$ShouldRun=false}
```

Note how the rule only runs if $ShouldRun is `true` and that once the run is run, $ShouldRun is set to `false`. This allows for some other action or rule to reset $ShouldRun and allow the rule to run one more time.

Rules can be disabled locally in a note using $RuleDisabled. The most common application of this is with prototypes where it is desired to run a rule in inheriting notes but not in the prototype itself.

Editing $Text or Displayed Attribute values will cause the note's rule to be run.

Edicts

An edict is essentially a rule—in terms of scope—but one which is run very occasionally. It is akin to an agent set to a low priority. There is no priority control for edicts. They are designed for background work, i.e. housekeeping tasks that nee to run occasionally but not often enough to have impact in the more pressing rules.

A note or agent's rule is stored in $Edict.

Edicts may be overridden locally in a note by using $EdictDisabled.

A note's edict runs once when it is:

- first created.
- edited.

All edicts run once when:

- a Tinderbox document is first opened.
- *roughly* once an hour for the rest of the session (i.e. while the document remains open).
- when File menu, Update Agents Now is clicked (i.e. both agents and edicts are refreshed once).
- when $Text or Displayed Attribute values are changed.

Stamps

Stamps are a method for manually applying action code to the currently selected item(s) in the view pane of the front document. If multiple items are selected, every item in the selection discretely has the same action code run upon it.

If the built-in Hints container is present, existing stamps are added as stamp notes and may be managed/edited either there or in the Stamps Inspector.

Stamps are action code 'expressions', i.e. one or more complete action(s) to be effected by the code. The concept of a stamp allows easy (manual) application and re-use of such code. They enable the user to:

- set many attributes at once. For example, a stamp with two discrete expressions to set a colour ($Color) and, separately, a badge ($Badge): `$Color="blue"; $Badge="ok";`.
- use conditions, shell scripts and macros for more complex tasks. For example, conditionally (an if() operation) set or clear a warning badge ($Badge) based on evaluating if a task's due date has passed ($DueDate): `if($DueDate<date("today")){$Badge="warning";}else{$Badge=;}`.

Stamps are defined/edited via the Stamps tab of the Document Inspector. Stamps can be called by:

- The Stamps menu.
- The Stamps tab of the Document Inspector (use the 'Apply' button).

Stamps are essentially an entirely manual way to apply actions that might otherwise as easily be effected automatically via rules, edicts, OnAdd events, or agent actions. Stamps are thus useful for when a known change needs to be made to some attributes but the scope or timing of such a change cannot easily be predicted or automated and so manual application is the best method.

Stamps requiring long/complex code can store the action in a code note and call it via the action() operator.

If stamps are just being used to easily apply a consistent look and layout to notes, consider using prototypes instead for the greater flexibility they offer when you subsequently need to make changes.

A document's stamps can be used in any stored action, e.g. any Action-type (system) attribute.

The stamp is called as an expression containing *only* the name of the stamp. The name is case-sensitive. Do not enclose it in quotes or Tinderbox will read it as a literal string (i.e. actual text). It must be terminated with a semicolon is followed by another expression. However, it is a good idea in this particular case of always using s semi-colon to help indicate your intent to the app.

Defined stamps with no code (a rare edge case when setting up a large document) will show in the Stamps Inspector but not in the Stamps menu.

The stamp name may be enclosed in quotes; unquoted and quoted forms both work. For a stamp called 'A test', the action would be be:

```
A test;
```

or

```
"A test";
```

The same stamp called, but here as one of 3 discrete expressions within an action:

```
$Color="blue";A test;$Width=4;
```

Unlike editing attributes via a note's Displayed Attributes table or Get Info's attributes tab, altering an attribute via a stamp (or Quickstamp) does not alter $Modified.

If a stamp changes the text, Tinderbox will immediately expand any ziplinks mark-up found in the revised text.

When applying a stamp to a note that is on exactly one adornment, the designator 'adornment' implies that adornment. If the note is on several adornments, the designator 'adornment' reflects only one of those adornments (the front-most?).

Stamp Groups

The Stamps menu supports hierarchical organisation of stamps. To group stamps, include a colon in the stamps title:

```
Colour:red border
```

This will cause the Stamps menu to add a group named 'Colour' and with submenu for the groups members, e.g. 'red border'. This method allows organising groupings of related stamps without creating unwieldy menus. Stamps cannot be nested more deeply than in groups, i.e. 1 level of nesting. Use of more than one colon in the name causes the stamp to not list as part of a Groupames are case-sensitive, so 'Colour Map:blue' and 'Colour map:green' would list in separate groups.

Note that pre-existing stamp names containing colons will form their own group, with the second part of the original name as a sub-menu, but they otherwise work the same way.

Stamps menu order

The Stamps menu lists stamps in the order shown in the Stamps Inspector. Where a group occurs, it it is listed where the first group member occurs in the Inspector List and within the sub-menu, the items are listed as in Inspector listing order. Grouped items do no need to be contiguous in the Inspector listing.

Currently, although Inspector-created stamps are replicated in the /Hints/Stamps container, the ordering of both is independent; re-ordering in one location does not update it in the other. If differing, the order in the Stamps menu is that of the Inspector and not the Stamps container.

Coloured syntax highlighting in Action code

UI input boxes for action code fields can use syntax highlighting. This includes link actions (Links Inspector) and notes using the built-in ' Action' prototype.

Coloured highlighting assist in spotting simple coding mistakes:

- Double-quoted strings are red.
- Single-quoted strings are orange.
- Attribute references are purple.
- Operators appear in blue. The function operator is coloured blue but function names are not.
- Comments appear in grey.
- The variable 'var' command in green. Note that variable names, including function inputs and loop variables are *not* coloured.
- Unmatched parentheses, brackets and braces are highlighted.

Notes:

- Nesting unmatched quotes, i.e. 2 doubles around a single or vice versa, will (may) cause that expression within the overall action to be wrongly coloured. If correctly coded, this glitch (due to regex limitations) will not affect the functioning of the code.This does not affect
- Code syntax highlighting of Action Code cannot be set by the user (i.e. by using custom highlighters). However, from v9.1.0, applying the Action prototype will enable code colouring in any note using that prototype.
- Syntax highlighting will remove any pre-existing text colouring/highlighting. Font/font size are not affected.
- Syntax highlighting should not be used at the same time as highlighters. If both are used, the outcome will be unpredictable. As the highlighter feature is intended for marking up general text, its use is inappropriate for action code syntax marking.

Code notes and use of the action prototype

For large sections of code, it has long been practice to store such code in a note using the built-in ' Code' prototype. However, from v9.6.0, if the code being stored is *action* code, e.g. a long/complex Rule, then using the Action prototype m be a better choice.

If applying the 'Action' prototype to a pre-existing note and the $Text is not in already in a Monospaced font, it is a good idea to re-set the note font. Select the note and use menu Format ▸ Style ▸ Standard Font.

Basic action code syntax

Case sensitivity: Action code operators are always case-sensitive (unlike export codes).

Delimiters. Each discrete code expression (explained further below), must be terminated with a semi-colon. This acts like a full stop (or period) in normal text writing. Tinderbox, will run the first expression up to the first-semi-colon, before moving on the the next and so on. Queries, as used by agents, do use action code operators but use slightly different syntax and **never** use a delimiter.

In action code, the delimiter can be omitted in two cases.

Firstly, the action only contains one expression. Both these are valid code for the same action:

```
$Color="red"
$Color="red";
```

Secondly, it is the last expression in a multi-expression action.Both these are valid code for the same action:

```
$Color="red";$AccentColor="blue";$StartDate=date("now")
$Color="red";$AccentColor="blue";$StartDate=date("now");
```

Note, in this second context, how all expression except the last one must have a semi-colon delimiter.

Where a multi-expression code includes if() conditional sections, a delimiter must be placed after the closing '}' if another expression follows after. For instance:

```
if($Prototype=="Event"){$StartDate=date("today")}
```

But:

```
if($Prototype=="Event"){$StartDate=date("today")};$Color="red"
```

Note that the code inside the curly brackets does not need a delimiter as there is only one action. But, a further action follows the if() test so a delimiter is needed after the '}' that closes the if() action. Where there is an else branch, e.g. if() {}else{}, the delimiter goes after the '}' at the end of the else branchode inside the {} sections obeys the rules above and only needs delimiters between multiple actions.

However, if in doubt, use a delimiter after each action!

Expression: this is a term for a section of action code involving some discrete evaluation, as described further here.

Coding conventions in Tinderbox. See more.

Review old code: Longtime users of Tinderbox should revisit code to ensure all string literals are quoted as current support for non-quoted string usage is for legacy purposes and will be withdrawn; by convention paths and designators are not quoted but there is no harm if they are and quoting all strings may be an easier approach for new users to take. Likewise, new users trying old code samples should be aware that the code may use legacy syntax.

Expressions in Action code

'Expression' is a term for action code involving some evaluation, and by extension any discrete complete section of action code. Thus even the simplest code, such as the assignment of a literal value, may still be regarded as an expression:

```
$Text="A Life on the Ocean Wave.";
```

Normally, actions and stamps require an action: an assignment, or an if(), each(), or var() statement.

However, actions may be expressions, which are evaluated for their side-effects. For example, `"frogs".speak()` is technically an expression, not an action, but if the action is run, the macOS will say the word "frogs". More often, actions and rules may use simple arithmetic and logical expressions.

```
$Width=5+2;
$Width=$Height(parent)*1.5;
$Urgent= ($Overdue|$Essential);
$Width =($Width(/$Configuration)+1)*$Scale;
```

Note the latter examples above the right-side involves more complex evaluation where extra parentheses are added to make explicit the evaluation order, as discussed further here.

One action, e.g. rule, can have multiple expressions:

```
$Color= "blue";$Badge="ok";
```

Multiple expressions must be delimited by semi-colons. Line breaks between expressions are optional. They may aid legibility of code but have no effect on how the code is parsed.

Because Tinderbox recognises operators such as + and -, notes whose names begin with characters other than letters and numbers may sometimes be interpreted in unexpected ways. For example, if a note is named "6*7", rules like

```
$Prototype=6*7
```

…might be parsed as a multiplication with the result of 42.

```
$Prototype="6*7"
```

…should have the expected effect. Quoting characters will always cause Tinderbox to treat such content as literal text.

Actions and rules may use computation with data attributes, including attribute values from other notes. For example:

```
$DueDate=$DueDate(parent)+"7 days";
$DueDate=$ExpiryDate(parent)-"10 days";
```

Date attribute values can also be computed with literal dates and date designators:

```
$DueDate="November 15, 2004"+"7 weeks"
```

In the previous examples note that the actual and partial date strings use quotes.

See further discussion of expressions:

- Expressions in attribute offset addresses
- Stand-alone expressions
- Left side expressions

Expressions in attribute offset addresses

Many action expressions designate a note, either by name, by path, or by object designators such as *parent* or *prevSibling*.

An extension to the path mechanism allows you to replace the normal string literal used for note designation with an expression:

```
$Color(7*6) <-- the colour of the note named "42"
$Color($Path) <-- the colour of the note whose name is stored in the attribute "Path"
```

In the case of ambiguity, expression evaluation has the lowest precedence. For example, if a note happens to be named "7*6", the reference will be to that note, not to the note named "42".

From v9.6.0, in actions, offset addresses in which the inner values is an expression are now supported. For example:

```
$Name(nextSibling($MyDictionary[vMagicKey]))
```

The way nested designator expressions are handled might conceivably change the behaviour of previously-working code. A good refactoring is to unpack the nesting. Thus instead of:

```
$Result= $Name(nextSibling(complexFunction($MyString)))
```

consider:

```
var:string vPath=complexFunction($MyString);
$Result=$Name(vPath);
```

Stand-alone expressions

Actions may include stand-alone expressions, that is expressions without an assignment or if() clause. A stand-alone expression can not change any attributes but a few expressions have useful side-effects, including use of the runCommar linkTo, linkFrom, unlinkTo, and unlinkFrom operators.

Often, the output from these operators is of no practical use and this saves the need for adding a user attribute simply to run this type of operator.

The action() function makes it easier to execute standalone code expressions.

Left side expressions

Group designators may be used to assign a value to the attributes of several notes at once. For example, the $Rule:

```
$Color(children)="red"
```

turns all the current note's children to red.

Designators may also be lists of notes, either explicit literal string list:

```
$Color(/config/one;/config/two)="red"
```

or implicitly computed via a find():

```
$Color(find($Price>5))="red"
$Color(find($Color=="red"))="red"
```

or implicitly computed by an expression enclosed in quotation marks:

```
$Color("collect_if(all,$Price>5,$Path)")="red"
$Color("collect_if(all,Color=='red',$Path)")="blue"
```

In the last example note the need for nested single and double quotes.

Basic Comparison Operators

Queries & conditional expressions support 6 basic comparison operators:

- ==. Equals. (See below re == and comparing dates. A single '=' usage is supported for legacy code purposes *but is deprecated in v4.6+*.)
- !=. Does not equal. The macOS-specific ≠ symbol may be used interchangeably with !=.
- >. Greater than
- <. Less than.
- <=. Less than or equal to. The macOS-specific ≤ symbol may be used interchangeably with <=.
- >=. Greater than or equal to. The macOS-specific ≥ symbol may be used interchangeably with >=.

Comparing date (data type values) with the == and != operators. Agents regard two dates as being equal if they share the *same year, month, and day* , regardless of the time of day. This makes useful agent expressions that use the == equalit operator. In early releases, equality looked only for identical time stamps, which was rarely what users expected.

For all other date comparisons, the time element of date-time attribute value is taken into account. Where no explicit time is set, the time defaults to '00:00' (hh:mm). Seconds are not displayed in Displayed Attributes or Get Info. Seconds are '00' in most cases unless specifically set otherwise (via action code or use of designators link 'now').

To test the exact equivalence of date attribute values use interval() where an exact match returns the string "00:00".

```
if(interval($DateA,$DateB)=="00:00"){…
```

Note that the comparison operators >, >=, < and <= continue to compare both date and time. It is possible, for example, for the same note to be discovered by two agents with different query operators, one that locates $TheDate < today and another that searches for $TheDate == today. If the $TheDate has a time of 14:30 and today has a time of 11:00 then both queries will match; the first matches simple on a time agnostic whole-day basis, the second matches because 14:40 is later (greater) than 11:00.

The date parser accepts the following expressions as placeholders for calculated dates (within quoted string literals):

- **yesterday**
- **today**
- **now**
- **tomorrow**
- **day**
- **week**
- **month**
- **year**

There is a further special case:

- **never**. No date, and treated as earlier or later than any real date.

Day placeholders Sunday, Monday, etc., may be used. Tinderbox recognises the day of the week (in English, currently) and interprets it as the day after today with that week day. Thus on Sunday June 1, the date "Sunday" refers to Sunday, June 8.

In other respects, days of the week act like today and can be modified similarly, e.g. "Monday - 1 week" evaluates as the previous Monday.

Parentheses (rounded brackets) can be used to indicate a hierarchy of resolution:

```
($Cost > (450 * $Margin)) & ($Priority == "High")
```

The code in parentheses is resolved before the '&' join is made.

Agent query comparison operations will accept expressions on their right-hand side. This is legal syntax: `$Width > ($Height * 2)`

- Equals (is the EXACT equivalent of)
- Does Not Equal
- Greater Than
- Greater Than Or Equal To
- Less Than
- Less Than Or Equal To

Equals (is the EXACT equivalent of)

$Attribute == $OtherAttribute

The double-equals (==) usage for tests of equality replace older use of a single equals sign.

Legacy use of single '=' for comparison

Previously, a single equals sign acted as either an assignment of the right side to the left or as an equality operator, Tinderbox deciding which form to use depending on context. A single equals sign is used for assigning values only and not fo testing equality.

$BooleanAttribute == *true*

Matches notes whose **$BooleanAttribute** value is ***true***.

For other data types, testing for the presence of a value can be considered a boolean test. This is useful as queries and if() clauses use a 'condition' which must resolve as `true` *or* `false`.

$ColorAttribute == color("data")

The **data** needs to be a string that matches $ColorAttribute *case-sensitively*.

Named colours matching hex values will be match and the case of hex values is ignored. Thus color("#0000ff") matches color("#0000FF") and color("bright blue"). But none of those would match color("Bright Blue"), unless of course it too had been defined with the same hex equivalent of #0000ff.

Note that Tinderbox's pre-defined named colours are all-lowercase, and that hexadecimal colour codes created via the colour picker use lower case for any letter characters.

$DateAttribute == date("aDate")

$DateAttribute == "aDate" <— Deprecated, do not use

In tests of date equality, the *time* element of a date/time is ignored. In other words the granularity of test is for the dates being on the same calendar day.

$DateAttribute == date("aDate")

$DateAttributeA == $DateAttributeB

true where **$DateAttribute** date matches the given **aDate**'s calendar day

$DateAttribute == date("today"). `true` where **$DateAttribute** date is today's date (same calendar day).

$DateAttribute == date("yesterday"). `true` where **$DateAttribute** date is **yesterday's** calendar day.

$DateAttribute == date("tomorrow"). `true` where **$DateAttribute** date is **tomorrow's** calendar day.

$DateAttribute == date("never"). `true` if there is no date. Note, 'never' is not < any date, nor is it > than any date, but it is equal to 'never'. Atypically, this null value is both older than the oldest set date and newer than the most recent set date; thus 'never' is earlier or later than any real date. When agents search for a specified time range, the comparison operators < and > treat the value `'never'` specially.

To search for a note with a valid date/time, it might seem logical to use `$DateAttributeA > "never"`. No! The correct test is `$DateAttributeA != "never"`.

The operators '<=' and '>=':

$DateAttributeA >= $DateAttributeB

$DateAttributeA <= $DateAttributeB

also return *true* if the dates are on the same calendar day (because the '=' part of these operators are a '==' equality test).

$NumberAttribute == number

This is `true` where $NumberAttribute value equals the given number. Note the number value is not quoted.

$ListAttribute == "data"

Note: Tinderbox does not perform meaningful equality comparisons on List-type attributes.

A match only occurs if data matches *all values* in the list. Thus if the list value is `[dog;cat]` and the **data** string is "dog", the result is `false`. If the string is "dog:cat" the result is `true`; a successful match also occurs for "cat;dog".

It can be seen that **data** is parsed for all values but that a true result is only achieved if *all values* match, whereas the likely intention of the user is to see if **data** matches one, or more, list values.

Lists are best compared for a pattern—either a literal string or regular expression—using:

- for matching only complete individual list values: the List.contains("regex") action operator.
- for regex partial matches: coerce the Set values to a String, and use String.contains("regex").

$SetAttribute == "data"

Note: Tinderbox does not perform meaningful equality comparisons on Set-type attributes.

A match only occurs if data matches *all values* in the set. Thus if the set value is `[dog;cat]` and the **data** string is "dog", the result is `false`. If the string is "dog:cat" the result is `true`; a successful match also occurs for "cat;dog".

It can be seen that **data** is parsed for all values but that a true result is only achieved if *all values* match, whereas the likely intention of the user is to see if **data** matches one, or more, set values.

Sets are best compared for a pattern—either a literal string or regular expression—using:

- for matching only complete individual list values: the Set.contains("regex") action operator.
- for regex partial matches: coerce the Set values to a String, and use String.contains("regex").

$StringAttribute == "data"

True where **$StringAttribute** value is identical to the **data** string. For instance:

```
$Text == "Hello"
```

…would be `true` if the note's text consisted only of the word "Hello".

Note the **data** string is must be quoted (it may work without quotes but do not rely on that!).

Does Not Equal

$Attribute != $OtherAttribute

$Attribute ≠ $OtherAttribute

This uses the negative equivalent test to the double-equals (==) usage for tests of equality.

Note that, when parsing *queries*, the template is evaluated in the agent's context. In particular, attribute references on the right side of the test are made to the agent and this refers to the agent. Conversely, in *actions*, on the other hand, the action is evaluated in the context of the note to which it is applied.

Using the '!=' syntax rather than '≠' is recommended if content is to exported to the web as not all fonts include the latter special character; the Unicode address simply says where a glyph is stored in the font, it does not dictate whether it exists in a given font.

$BooleanAttribute != *false*

!($BooleanAttribute == *false*)

~~**$BooleanAttribute == *false***~~ (do not use this form)

Matches notes whose **BooleanAttribute** value is `true`.

For other data types, testing for the presence of a value can be considered a boolean test. This is useful as queries and if() clauses use a 'condition' which must resolve as `true` or `false`.

Note the alternative **!(expression)** syntax in the second example above and its use of parentheses. Omitting these and simply pre-pending an exclamation mark to the attribute being tested may work but is deprecated in favour of use of parentheses as this helps Tinderbox to parse correctly the user's intent.

$ColorAttribute != color("data")

The **data** needs to be a string that matches $ColorAttribute *case-sensitively*.

Named colours matching hex values will be match and the case of hex values is ignored. Thus color("#0000ff") matches color("#0000FF") and color("bright blue"). But none of those would match color("Bright Blue"), unless of course it too had been defined with the same hex equivalent of #0000ff.

Note that Tinderbox's pre-defined named colours are all-lowercase, and that hexadecimal colour codes created via the colour picker use lower case for any letter characters.

$DateAttribute != "aDate"

In tests of date inequality, the *time* element of a date/time is ignored. In other words the granularity of test is for the dates not being on the same calendar day.

$DateAttribute != date("aDate")

True where **$DateAttribute** date does not match the given **aDate**'s calendar day.

$DateAttribute != date("today"). `true` where $DateAttribute date is not **today**'s date (same calendar day).

$DateAttribute != date("yesterday"). `true` where $DateAttribute date is not **yesterday**'s calendar day.

$DateAttribute != date("tomorrow"). `true` where $DateAttribute date is not **tomorrow**'s calendar day.

$DateAttribute != "never". As **never** is 'no date', this is effectively a test that a value exists for the attribute. Note, 'never' is not < any date, nor is it > than any date, but it is equal to 'never'.

NumberAttribute != number

or

Gathers all notes whose **NumberAttribute** value do not equal the given **number**.

Using the '!=' syntax rather than '≠' is recommended if content is to exported to the web as not all fonts include the latter special character; the Unicode address simply says where a glyph is stored in the font, it does not dictate whether it exists in a given font.

Note that the usage '<>' is not a valid method in Tinderbox for asserting inequality.

$ListAttribute != "data"

Note: Tinderbox does not perform meaningful equality comparisons on List-type attributes.

A match only occurs if data matches *all values* in the list. Thus if the list value is `[dog;cat]` and the **data** string is "dog", the result is `false`. If the string is "dog:cat" the result is `true`; a successful match also occurs for "cat;dog".

It can be seen that **data** is parsed for all values but that a true result is only achieved if *all values* match, whereas the likely intention of the user is to see if **data** matches one, or more, list values.

Lists are best compared for a pattern—either a literal string or regular expression—using:

- for matching only complete individual list values: the List.contains("regex") action operator.
- for regex partial matches: coerce the Set values to a String, and use String.contains("regex").

$SetAttribute != "data"

Note: Tinderbox does not perform meaningful equality comparisons on Set-type attributes.

A match only occurs if data matches *all values* in the set. Thus if the set value is `[dog;cat]` and the **data** string is "dog", the result is `false`. If the string is "dog:cat" the result is `true`; a successful match also occurs for "cat;dog".

It can be seen that **data** is parsed for all values but that a true result is only achieved if *all values* match, whereas the likely intention of the user is to see if **data** matches one, or more, set values.

Sets are best compared for a pattern—either a literal string or regular expression—using:

- for matching only complete individual list values: the Set.contains("regex") action operator.
- for regex partial matches: coerce the Set values to a String, and use String.contains("regex").

$StringAttribute != "data"

This is `true` where StringAttribute value is identical to the data string. For instance:

```
$Text != "Hello"
```

…would be true if the note's text consisted of something other than just the word "Hello".

Note the data string is must be quoted (it may work without quotes but do not rely on that!).

Greater Than

$DateAttribute > "date"

Gathers all notes whose **$DateAttribute** value is after the given **date**. The left-side date cannot be more than 'never'.

WARNING: unlike an equals comparison the *time* element of the date/time is used. If $DateA is 28 November 2012 21:00:00 and $DateB is 28 November 2012 12:00:00 then:

```
$DateA < $DateB is false
```

whilst

```
$DateA == $DateB is true
```

This is because the former takes time into account and the latter does not. Also:

```
$DateA <= $DateB is true
```

Here the implicit equality test (ignoring time) will trump the time-sensitive < test. In <=, it is effectively doing an OR join, i.e. the result is true if:

```
$DateA < $DateB is false
```

$NumberAttribute > number

Gathers all notes whose **$NumberAttribute** value is greater than the given **number**.

Other data types…

Boolean. A `true` is always greater than `true` (reflecting the fact the coerce to the values '1' and '0' respectively).

Color. Tinderbox does not perform greater/less than comparisons on Color-type attributes.

Sets/Lists. Tinderbox does not perform greater/less than comparisons on Set-type and List-type attributes.

String. A lexical comparison is done of each string character by character, i.e. the ASCII value of each first character, then each second character, etc. Uppercase characters < lowercase < numbers. For other characters, accents, etc., comparisons are likely to not meet linguistic expectation as the values will be based on Unicode sort order. Thus:

"dog" > "cat"
"dog" > "Dog"
"dogs" > "dog"
"dogs" > "dogma"
"dogs" < "døg" <-- NOTE!

Greater Than Or Equal To

$DateAttribute >= "date"

$DateAttribute ≥ "date"

Gathers all notes whose **$DateAttribute** value is on or after the given **date**. The left-side date cannot be more than 'never'.

Because this test is effectively ((A > B) | (A ==)), the equality test means the time part of the date time is ignored, unlike with a plain 'greater than' test.

Using the '>=' syntax rather than '≥' is recommended if content is to exported to the web as not all fonts include the latter special character; the Unicode address simply says where a glyph is stored in the font, it does not dictate whether it exists in a given font.

Depending on the granularity required, it may be necessary to use a two-term query for date comparisons.

$NumberAttribute >= number

$NumberAttribute ≥ number

Gathers all notes whose **$NumberAttribute** value is greater than or equal the given **number**.

Using the '>=' syntax rather than '≥' is recommended if content is to exported to the web as not all fonts include the latter special character; the Unicode address simply says where a glyph is stored in the font, it does not dictate whether it exists in a given font.

Other data types…

Boolean. A `true` is always greater than `true` (reflecting the fact the coerce to the values '1' and '0' respectively).

Color. Tinderbox does not perform greater/less than comparisons on Color-type attributes.

Sets/Lists. Tinderbox does not perform greater/less than comparisons on Set-type and List-type attributes.

String. A lexical comparison is done of each string character by character, i.e. the ASCII value of each first character, then each second character, etc. Uppercase characters < lowercase < numbers. For other characters, accents, etc., comparisons are likely to not meet linguistic expectation as the values will be based on Unicode sort order. Thus:

"dog" > "cat"
"dog" > "Dog"
"dogs" > "dog"
"dogs" > "dogma"
"dogs" < "døg" <-- NOTE!

Less Than

$DateAttribute < "date"

Gathers all notes whose **$DateAttribute** value is before the given **date**. The left-side date cannot be less than 'never'.

WARNING: unlike an equals comparison the *time* element of the date/time is used. If $DateA is 28 November 2012 09:00:00 and $DateB is 28 November 2012 12:00:00 then:

`$DateA < $DateB` is `true`

whilst also

`$DateA == $DateB` is `true`

This is because the former takes time into account and the latter does not.

$NumberAttribute < number

Gathers all notes whose **$NumberAttribute** value is smaller than the given **number**.

Other data types…

Boolean. A `true` is always greater than `true` (reflecting the fact the coerce to the values '1' and '0' respectively).

Color. Tinderbox does not perform greater/less than comparisons on Color-type attributes.

Sets/Lists. Tinderbox does not perform greater/less than comparisons on Set-type and List-type attributes.

String. A lexical comparison is done of each string character by character, i.e. the ASCII value of each first character, then each second character, etc. Uppercase characters < lowercase < numbers. For other characters, accents, etc., comparisons are likely to not meet linguistic expectation as the values will be based on Unicode sort order. Thus:

"dog" > "cat"
"dog" > "Dog"
"dogs" > "dog"
"dogs" > "dogma"
"dogs" < "døg" <-- NOTE!

Less Than Or Equal To

$DateAttribute <= "date"

$DateAttribute ≤ "date"

Gathers all notes whose **$DateAttribute** value is before or on the given **date**. The left-side date cannot be less than 'never'.

Because this test is effectively ((A < B) | (A ==)), the equality test means the time part of the date time is ignored, unlike with a plain 'less than' test.

Using the '<=' syntax rather than '≤' is recommended if content is to exported to the web as not all fonts include the latter special character; the Unicode address simply says where a glyph is stored in the font, it does not dictate whether it exists in a given font.

Depending on the granularity required, it may be necessary to use a two-term query for date comparisons.

$NumberAttribute <= number

$NumberAttribute ≤ number

Gathers all notes whose **$NumberAttribute** value is smaller than or equal to the given **number**.

Using the '<=' syntax rather than '≤' is recommended if content is to exported to the web as not all fonts include the latter special character; the Unicode address simply says where a glyph is stored in the font, it does not dictate whether it exists in a given font.

Other data types…

Boolean. A `true` is always greater than `true` (reflecting the fact the coerce to the values '1' and '0' respectively).

Color. Tinderbox does not perform greater/less than comparisons on Color-type attributes.

Sets/Lists. Tinderbox does not perform greater/less than comparisons on Set-type and List-type attributes.

String. A lexical comparison is done of each string character by character, i.e. the ASCII value of each first character, then each second character, etc. Uppercase characters < lowercase < numbers. For other characters, accents, etc., comparisons are likely to not meet linguistic expectation as the values will be based on Unicode sort order. Thus:

"dog" > "cat"
"dog" > "Dog"
"dogs" > "dog"
"dogs" > "dogma"
"dogs" < "døg" <-- NOTE!

Compound Actions

Compound actions are just like simple actions but each action is separated by a semicolon:

Action: `$Color="red";$Priority="high"`

Sometimes, this leads to complications. Suppose, for example, you want an action to set the $OnAdd action. (Perhaps you have an agent that's looking for containers of $UrgentTasks, and wants to set the $OnAdd action for those containe
The simplest way of writing this is:

Action: `$OnAdd=$Color="red";$Priority="high"`

…which will do two things: set the $OnAdd action to "$Color=Red" and set the $Priority to high.

If you want the $OnAdd action to be set to the entire expression, enclose it in quotes:

Action: `$OnAdd="$Color=red;$Priority=high"`

The square bracketed style is also required if setting multiple $DisplayedAttributes values (or any List or Set-type attribute).

Action: `$DisplayedAttributes=[Color;Name;Rule]`

Action: `$DisplayedAttributes= $DisplayedAttributes(parent) + [Border;AccentColor]`

Previously, the latter used quoted strings but this is now deprecated.

Conditional statements using multiple arguments

If() conditional statements and queries may use more than one argument using logical joins. See AND and OR join syntax.

(A & B)

Reads: if **condition A AND condition B** are `true`, then this compound query is `true`.

Contrast this with the &= operator (below).

(A &= B)

Reads: result is `true` if **condition A** is currently `true` and, if so, if **condition B** is also `true`.

Thus, if **condition A** is `false`, **condition B** is not checked and the overall result is `false`. Thus in some cases only **condition A** is evaluated.

By comparison, with a **(A & B)** query both **condition A** and **condition B** are are *always* evaluated.

(A | B)

Reads: if **condition A OR condition B** is `true`, then this compound query is `true`.

Contrast this with the |= operator (below).

(A |= B)

Reads: result is true if **condition A** is currently `true` or, if so if **condition B** is `true`.

Thus, if **condition A** is `false`, **condition B** is not checked and the overall result is `false`. Thus in some cases only **condition A** is evaluated.

By comparison, with a **(A | B)** query both **condition A** and **condition B** are are *always* evaluated.

(A | B) & (C | (D | E))

This is a more complex example of logical joins. Reads: if (**condition A OR condition B** is `true`) **AND** (**condition C OR** (either **condition D OR condition E** are `true`)), then this compound query is `true`.

Defining an 'item' object—a single item—in action code

The term 'item' is not a Tinderbox definition but is useful in the context of discussing action code and documenting action code usage.

Where an argument is defined as item scope this means it returns *one and only one object* (as compared to a group scope). If more than one item is returned where only one is desired, e.g. from a find() query) Tinderbox will use *only the firs item* of the list, by ascending $OutlineOrder, as the specified item; the same holds for an expression that evaluates with an ambiguous result. It thus makes sense to avoid such ambiguity and ensure the query is sufficiently scoped to return only one match. The overall point is that whether using a string literal value (i.e. just stating a name) or using a code expression, the outcome must evaluate to provide Tinderbox with a reference to a single note.

A single note 'item' can be referenced as:

- A note's title (its value for $Name).
- A notes's path (its value for $Path). This is useful if the note's name is not unique.
- (less commonly) A note's ID ($ID). This is less usual but can be useful if there are problematic characters in the title or path, especially if using (un)linking actions.
- An item-scope note designator, e.g. 'parent' or 'firstSibling'. Designators return, $Path data for the matching object.
- A query function, e.g. find(), collect(), collect_if(), links(). The query must resolve to a single note item. Query functions return $Path data for the matching object.
- An expression, i.e. action code that evaluates as any of the above. Complex expressions may require use of parentheses and/or use of eval() to help Tinderbox understand the order of evaluation and to deal with encapsulated evaluations.
- A string attribute value that is an expression or a literal title/path.
- A discrete value in a list attribute that is an expression or a literal title/path.
- A literal string, e.g. `"This note"`.

Defining 'group' list objects—one or more items—in action code

The term 'group' is not a Tinderbox definition but is useful in the context of discussing action code and documenting action code usage.

A group is a list of *one or more* Tinderbox note items (q.v.). A group can be referenced as:

- A list of note titles ($Name).
- A list of note paths ($Path). This is useful if any note's $Name value is is not unique.
- (less commonly) A list of note IDs ($ID). This is less usual but can be useful if there are problematic characters in the title or path, especially if using (un)linking actions.
- A group-scope note designator, e.g. 'all' or 'children'. Designators return $Path data for the matching objects.
- A query function, e.g. find(), collect(), collect_if(), links(). The query must resolve to a list of one or more note items. Query functions return $Path data for the matching objects.
- An expression, i.e. action code that evaluates as any of the above.
- A string attribute value that is an expression or a literal title/path.
- A discrete value in a list attribute that is an expression or a literal title/path.
- A literal list, e.g. `[This note;That note]`. (Previously, and now deprecated, was to use a quote-enclosed list)

See also the article on items.

Note: In the linkedTo() and linkedFrom() query terms the `" *"` designator replaces the `" all"`, presumably for legacy purposes.

Why is a text 'line' in action code actually a paragraph?

To the lay person, a 'line of text' is a line as seen on the page or screen, i.e. from the left margin to where the text stops at the right margin. Thus a long sentence may 'wrap' onto a new, different, line so as to remain visible to the reader.

For instance, as currently drafted in the author's TBX file, the text above comprises *two sentences* wrapped onto *three lines* (for display purposes). It is also only *one paragraph* of text.

Importantly, it is the latter definition of a 'line'—i.e what we more normally think of as a discrete 'paragraph'—that must be understood as 'line' when using action code text operations. To clarify, from a code perspective the discrete lines within a String of textual data are most easily understood as:

- the run of characters from the text's start to the first line break character,
- then each run of characters between a pair of successive line break characters within the text,
- plus the run of characters after the last line break character up to the text's end.

Therefore for action code use, this article is *seven* 'lines' of text (each bullet is effectively a paragraph), even though it may *display* as many more lines when read in the TBX file or ints output web pages.

Explicit declaration of lists using square brackets

NOTE: in this article 'list' refers to use in both List and Set type data.

The List (or Set) operator: declaring lists using literal values

This explicitly converts a String to a List. That used to be important, but the introduction of typed local variables makes these unnecessary in many cases.

The preferred usage is for declaring lists (List or Set) is `[… ]`

From v9.6.0 a List or Set is declared using enclosing square brackets and without enclosing quotes:

```
var:list vList = [peach;orange;apple];
```

Deprecated: in the past, defining literal list values involved using a quote-enclosed semicolon delimited list:

```
var:list vList = "peach;orange;apple";
```

N.B. *Both* forms work equally well—at least for now.

However, the bracket-enclosed method is now the preferred syntax for writing list in new code. In due course aTbRef will update most examples to the newer format.

Declaring Nested lists

The new form of list definition also allows declaration of *nested* lists:

```
var:list vMatrix = [[0;1;2];[3;4;5];[6;7;8];[9;10;11]];
$Text = vMatrix[1];
```

This returns the result, [3;4;5], i.e. a List-type value containing a 3-item list.

Nested lists and Zip-method linking

Typing the syntax for nested links can involve character sequences like `[[` and `]]` which are other wise use with the 'zip' method of making Tinderbox links. the latter can be suppressed for the current note by altering the $Ziplinks system attribute.

Strings In Lists

Take caution mixing quotes and square brackets:

```
var:list vList = ["peach;orange;apple"];
```
→ single item list

This creates a list with one element: the string "peach;orange;apple". This is needed very occasionally, e.g. for a string value legitimately needing to hold a semi-colon, such as might occur when doing manipulation of $Text content. If inspecting the list value in Displayed Attributes or Get Info/attributes, such a string list value is easily spotted as it is enclosed in the 'raw' list value using double quotes. This is a four-item list with a single 'string' item value as the third list item.

```
[rock;eel;"peach;orange;apple";monkey]
```

By comparison:

```
var:list vList = "[peach;orange;apple]";
```
→ three item list, but see next example...

In fact, the latter is a redundant form of the new bracketed declaration:

```
var:list vList = [peach;orange;apple];
```

but which might be typed in error by someone more used to the original from of defining a list.

The list() operator

Occasionally, it may be necessary to declare a list form evaluated expressions. The list() operator is used for that task.

Evaluation of [] list terms

From v9.6.0, implicit evaluation is no longer performed in bracketed lists. For example,

```
$DisplayedAttributes=[MyList;MyString];
```

is now equivalent to

```
$DisplayedAttributes="MyList;MyString";
```

If list terms need evaluation, use list() instead.

Explicit declaration of dictionaries using curly braces

The Dictionary operator

This explicitly converts a String to a Dictionary. That used to be important, but the introduction of typed local variables makes these unnecessary in many cases.

The preferred usage is for declaring Dictionaries is using `{… }` rather than the previous dictionary() operator. Thus:

```
var:dictionary vDict = {English:I; Latin: ego};
```

replaces both these existing methods—though they still work (the quoted string version already bing deprecated):

```
var:dictionary vDict="English:I; Latin: ego";
var:dictionary vDict=dictionary("English:I; Latin: ego");
```

Nested Lists in Dictionaries

The square-bracket List declaration can be used to define multi-item key values

```
var:dictionary vDict = {Apple:[red;green];Lemon:yellow};
```

Calling Dictionary keys

By preference, dictionary keys should not be quoted though for legacy reasons either form will still work:

```
$MyDictionary = {Latin: "ego"; English: "I"};
// Preferred form
$MyValue = $MyDictionary[Latin];
// and not
$MyValue = $MyDictionary["Latin"];
// though *both* forms work is only for legacy support
```

The Order Of [Dictionary].keys

Tinderbox dictionaries currently depend on AppKit's NSDictionary, which appears to use a hash to order its keys. So, the (number) order is arbitrary, i.e. calling a value using `$MyDictionary[N]` is not recommended.

Punctuation and special characters in definitions, actions and expressions

In short, what is the meaning of brackets (parentheses, square or curly)? What to they do and what are they called. These are explained in the articles below.

This is a topic that is difficult to describe easily for all readers as the perspective and vocabulary used by different users will vary. Notably, those who have formal programming training may differ from most others. This note is written with the latter in mind, as they do not have any formal grounding upon which to fall back. Thus for some readers, terminology may vary from that they might otherwise use, but here it is consistent within aTBRef and in Tinderbox use.

A number of non-letter characters described here can have varying names depending on locale (even amongst English speakers) and field of study or experience. There are 4 types of brackets and in the listing below the name used in this section of the documentation is the one in italics:

- **( )**. Variously: brackets, round brackets or *parentheses*.
- **[ ]**. Variously: brackets, *square brackets*, crochets.
- **{ }**. Variously: *curly brackets*, braces.
- **< >.** Angle brackets. These are not used in action code. However, the same characters are used individually as mathematical operators for 'less-than' and 'more-than'.

Also be aware that the meaning of a 'line' of text is different in action code to the convention of the printed page.

More on specific usages of differing types of non-letter characters in code:

- Semicolon: expression delimiter, code line end
- Semicolon: list and dictionary item delimiter
- Colon: dictionary key-value pair delimiter
- Colon: ad hoc delimiter in some action operators
- Dollar-sign prefix: attribute references
- Dollar-sign prefixed numbers: query back-references
- Dollar-sign prefixed numbers: macro arguments
- Parentheses: attribute 'offset' references (offset addressing)
- Parentheses: arguments for action code operators and user functions
- Parentheses: controlling parsing of code
- Square brackets: lists and nested lists
- Square brackets: dictionary data keys
- Square brackets: list indexes
- Square brackets: dictionary keys with multiple values
- Square brackets: in documentation, optional arguments
- Curly brackets: dictionaries and nested dictionaries
- Curly brackets: defining code blocks
- Caret delimiters: export code operators
- Forward slash: folder delimiter in paths
- Double forward slash: action code comments
- Backslash: escape character
- Full stop: dot-operators
- Comma: function argument delimiter
- Symbols used in Mathematical and Logical operations
- Symbols used in Regular Expressions

Semicolon: expression delimiter, code line end

Semicolons are used in actions to separate discrete code expressions, i.e. discrete complete tasks. a desired outcome may use one such expression or several.

```
$Color = "blue'; $Badge = "ok";
```

General practice is to place each expression on a new line within an action, so semicolons are most often seen as line-ends in code.

```
$Color = "blue';
$Badge = "ok";
```

The last expression in an action of multiple expression doesn't require a semicolon end but no harm comes from doing so.

```
$Color = "blue'; $Badge = "ok";
```

Thus for the novice a good rule of them is to always end lines of code with a semicolon.

An exception is code lines ending with an opening curly bracket— see why.

Failure to use a semicolon where needed will result in a silent failure of all or part of the action.

Semicolon: list and dictionary item delimiter

Where attribute values can have multiple values, a semi-colon is used as a 'delimiter' between each item and optionally after the last item. Multi-value data types include:

List:

```
$MyList = [car;train;boat;plane;boat];
```

Set:

```
$MySet = [car;train;boat;plane;boat];
```

(here value `boat` will be de-duplicated and the list sorted in the resulting Set), i.e. `[boat;car;plane;train]`.

Dictionary:

```
$MyDictionary = {car:weels;truck:wheels;train:rails};
```

Note in the case of a dictionary, the list is one of `key:value` pairs.

Colon: dictionary key-value pair delimiter

In a Dictionary type, data, consists of a semicolon delimited list of keys and their values. For each such list item, the key and its value are separated by a colon. Generically:

```
dictionary("key:value; key:value");
```

Thus, in current form:

```
$MyDictionary = {apple:fruit;pear:fruit;pea:vegetable};
```

and old form (both forms work):

```
$MyDictionary = dictionary("apple:fruit;pear:fruit;pea:vegetable");
```

Noting that whitespace in the dictionary data *either side of* colons and semicolons is ignored, but not within a key name or value. Keys ideally should be a single word:

Colon: ad hoc delimiter in some action operators

In a small number of operators, a colon is used instead of other more normal delimiters and joins.

In some of the operators relating to composites, a colon is used where a dot-operator type of dot-join might be expected:

- compositeFor(nameStr):count
- compositeFor(nameStr):kind
- compositeFor(nameStr):name
- compositeFor(nameStr):role(roleStr)
- compositeFor(nameStr):roles

In the stream processing group of operators, Stream.eachLine(loopVar[:condition]){actions} uses a colon where a comma argument delimiter might normally be expected.

Dollar-sign prefix: attribute references

When reading from/writing to attributes in action code the desired attribute is 'referenced' by using a $-prefix to the attribute name thus:

```
$MyString
```

and not

```
MyString
```

Why so? Adding the $-prefix helps signal to Tinderbox that the user is referring to a specific attribute and not literal text, or an action code operator, or a user function, or a user variable of the same name.

Note on legacy use

The above holds true except for in the very early Tinderbox days, when the application would figure it out. But as action code has grown in complexity over the years it makes sense to signal user intent to the action code parser and so whilst omitting the $-prefix *may* still work it is formally deprecated since version 4.5.0.

Dollar-sign prefixed numbers: query back-references

Regular expressions used in queries can create up to 10 back-references expressed as `$9` through `$9`. Such codes are discussed in detail here.

Dollar-sign prefixed numbers: macro arguments

Macros can be used in both action and export code. Within the macro's stored code, calling argument values are inserted using a $-number that reflects the sequence number of the argument supplied in calling code.

Consider a macro named "Hello" that has this stored code:

```
"Hello $1!"
```

When using the 'Hello' macro in code, it is called like this:

```
$MyString = do("Hello","Cubert"));
```

which inserts the argument value 'Cubert' into the macro code at the position `$1`. The result is the output of the macro passed to $MyString is "Hello Cubert!".

See more on macros and do().

Parentheses: attribute 'offset' references (offset addressing)

Referring to attributes of the current note

In action code, when calling the value of an attribute the value returned is for the current note. In most normal cases this is the note currently selected in the view (or the first item in the selection for a selection of notes). Getting the colour of the current note is coded like so:

```
$Color
```

So it could also be written as:

```
$Color(this)
```

as in specifying 'this' note. But, here the explicit use of the this designator is redundant, as is coding the action as:

```
$Color()
```

All the above 'mean the same thing to the Tinderbox action code parser, but the first example is the normal usage. The value returned is the value set for that note, or an inherited value. Inheritance of attribute is discussed separately, here. The syntax above is used for both reading and setting values:

```
$MyColor = $Color;
$Color = "blue";
```

Addressing (referring to) attributes of the current note

But what if there is a need to fetch a value of the same attribute, *but from a different note*? That is where those trailing parentheses come into play. To fetch the value of $Path for a note called "Prospects", the note title of target source note the value—its $Name—is given as an argument inside the parentheses:

```
$Color("Prospects")
```

Note it is the same as for the current note but extra information is now supplied in parentheses directly after the attribute name:

```
$Color("Prospects")
```

Correct

Do not use white space between the name and opening parenthesis:

~~`$Color ("Prospects")`~~ WRONG

The same method applies whether or getting a value in another note:

```
$MyColor = $Color("Prospects");
```

or setting a value in another note:

```
$Color("Prospects") = "blue";
```

For offset addressing there is only ever one argument inside the parentheses.

The basic syntax for an offset addressing is as basic as that.

Other ways to specify the argument for offset addressing (i.e. to a different note)

The two methods above are the most common form of specifying the source/target. Occasionally, such as when two notes in the same container (or map) have the same $Name, this means their path is the same. In this case, the note's ID can be used via $IDString (which supplants older use of $ID for this purpose). In this example note 'Prospects' has an $IDString value of "tbx:BktGvS":

```
$MyColor = $Color("tbx:BktGvS");
$Color("tbx:BktGvS") = "blue";
```

Also see the operator listing for $AttributeName[(scope)], which covers this topic from the perspective of action code operators.

What if the offset value needs to be variable value?

For instance, finding the $Color of the first child of the current note.

Designators. In this case a designator is used (for more about designator use in actions: see here). Here, the correct choice would be the 'child' designator:

```
$MyColor = $Color(child);
$Color(child) = "blue";
```

Note that designators are not enclosed in quotes. This helps signal to Tinderbox that the argument is not a note title or path.

Attribute values. Another way to use a variable argument is to use a value stored in another attribute. If a Color-type attribute 'SomeColor' holds a colour value, then:

```
$Color = $MyColor;
```

or putting together some lessons above

```
$Color("tbx:BktGvS") = $MyColor("Prospects");
```

Note that when reading or setting an attribute values, the attribute name must be prefixed with a '$' character.

Action code variables. Action code also allow creation of variable and these to can be used in an offset. Assume a variable 'vColor' has been defined and stored a colour value, then:

```
$Color = vColor;
```

Evaluating arguments

In more complex situations it might appear there is a need to use an expression as an argument. Better is to evaluate that expression and store the result in an attribute or a variable and use the latter as the argument describing the offset.

Offset address arguments: when to quote or not?

Standard practice is to enclose literal text and titles ($Name) used as arguments within (straight) quotes but not quote note paths ($Path). The norm is to use double straight quotes but single straight quotes may be used instead. Doing the opposite *generally* gives the same result. But, users new to action code and/or without programming experience are advised to use the standard practice described. In addition, variables or attribute $-references are never enclosed in quotes. So:

```
$Color = "blue";
$Color = $Color("Prospects");
$Color = $Color(/the/path/to/Prospects);
$Color = $Color($MyString); where $MyString holds a path or title value
$Color = $Color(vString); where vString holds a path or title value
```

Parentheses: arguments for action code operators and user functions

Although historically user functions came the Tinderbox much later than action code operators, both use the same syntax:

- Operator or function name. *Names are case-sensitive*.
  - User function names are set by the user by defining the function.
- Trailing parentheses containing argument(s)
  - arguments are always comma-delimited. Whitespace before after the commas is ignored by the code parser.
  - where there are no arguments or all arguments are optional and not being used, the parentheses may be omitted, e.g. `.sort()` vs. `.sort`. If in doubt, e.g if new to action code, use the empty parentheses. If unsure if parts of th syntax may safely be omitted, do not guess—use the full syntax.
  - operators may have optional arguments. Optional arguments are indicated in the main operator listing.
  - non-optional arguments *must* have a value, even if only an empty/default value, e.g. `""`.
    - *all* user function arguments are *always* mandatory.
- some operators, and user functions also use a curly-bracket enclosed code block.

Examples

An operator, here the '[ ]' indicate that one of the arguments is optional:

```
stamp([scope, ]stampName)
if(condition){actions}[else{actions}]
```

A user function (following aTbRef general naming conventions):

```
myFunction(iArg1,iArg2){ }
```

In the last example, as a user-defined panel, all arguments must be populated even if only with an empty value, thus:

```
$SomeResult = myFunction("hello",24){ } Correct
$SomeResult = myFunction("",24){ } Correct
$SomeResult = myFunction(,24){ } WRONG
```

Parentheses: controlling parsing of code

By adding parentheses into action code, including queries, the oder of evaluation can be altered. When evaluating the most deeply nested terms are evaluated first.

Order of evaluating mathematical operators

Just as is the case in formal coding or formulas in an ordinary spread sheet, when it comes to carrying out mathematical operations there is an 'Order of Operations' (see more on Wikipedia).

Thus '1 + 2 × 3' can be considered ambiguous depending on whether the addition or the multiplication is done first. Thus the convention is

```
1 + (2 × 3) = 7
```

but alternatively:

```
(1 + 2) × 3 = 9
```

As is evident from the use of parentheses in these examples, by adding parentheses the user can indicate to Tinderbox their desired order of evaluation. Generally, additional parentheses are needed if the desired outcome is know to be the normal order of evaluation.

Order of evaluation in queries

Queries are used in agents, if(condition) tests and in find(query) operations. Queries are evaluated as read, i.e. left to right (start to finish), but in some cases additional hinting via parentheses can help both the parser and the user to understand their intent (especially when re-reading complex queries weeks or months later). Consider the outcome of this 5-term query, with terms A to E:

```
A | B & C | D | E
```

compared to:

```
(A | B) & ( C | ( D | E ) )
```

Regardless of whether both evaluate the same way, the latter is much more easily understood in terms of the order things are done.

Query syntax is discussed in more detail here, and here.

Possible issues when using parentheses in a note title ($Name)

Because parentheses are used to indicate how to parse data, this is why it is a good idea—if possible—to avoid using a forward slash character within a note name as it can prove problematic for some Tinderbox automation operations.

Square brackets: lists and nested lists

Square brackets in action code were originally added to deal with the problem of nested lists, i.e. where a list item is itself a list. This first became an issue when the Dictionary data type was added and users wished to have keys with multiple values. As a dictionary is a semicolon delimited list of key;value paired terms, if a value were also a list where does the key:value pair end.

To overcome this a list item value in a List or Set that is a list, or a Dictionary key value that is itself a list, i.e. a nested list, can be marked with [ ] brackets:

```
$MyList = [places;[apples;oranges];plants];
```

Furthermore, the [ ] is used to define a complete list:

```
$MyList = [places;plants];
$MyList = [places;[apples;oranges];plants];
```

this replaces the older, and still working—but *deprecated*— quote enclosed form:

```
$MyList = "places;plants";
```

Importantly, note how in the first examples above the brackets *replace* the use of quotes as seen in the last example.

Note that, being new, at present very few code examples use this notation although it is now to be considered best practice new code when defining lists. Especially for those used to using quotes the old form still works but avoid these mixed forms, which may not work or have unexpected outcomes:

```
$MyList = "[places;plants]"; WRONG
$MyList = ["places;plants"]; WRONG
```

If it is necessary to define a list where the list item(s) need to first be evaluated, use the list() operator.

Here is a further example using Dictionary-type data where a key's value needs to be a list First in the new { } dictionary notation:

```
$MyDictionary = {dog:terrier;boat:yawl;fruit:[apple;pear]};
```

and now in the older (still working) format:

```
$MyDictionary = dictionary("dog:terrier;boat:yawl;fruit:[apple;pair]");
```

Square brackets: dictionary data keys

A different use of square brackets from list identification is that of addressing a key in a Dictionary. Doing so causes the value of the address key to be returned. So:

```
$MyDictionary = {car:hatchback;ship:ketch;plane:cargo};
$MyValue = $MyDictionary[ship]; gives 'ketch'
```

In the past, string quoting rules are interpreted as meaning key terms with quoted but this is no longer best practice—see the article on {} Dictionary definition.

Square brackets: list indexes

A list comprises a a number of different items. Often it is useful to address (call for) a list item by its number within the list. For example, historically this was done using .at():

```
$MyItem = $MyList.at(2);
```

This returns item number *three* in the list, as the count used is 'zero-based', i.e. it starts at zero rather than one.

Another, and more current way is to use [N] directly after the object addressed. Thus, and functionally the same as above:

```
$MyItem = $MyList[2];
```

This also makes it easier to address values in nested lists. Thus in this list:

```
$MyList = [4;5;[6;2;4];9;7];
```

the third list item is itself a list. To address a value within that nested lest, append another set of [] like so:

```
$MyItem = $MyList[2][1];
```

returns '2', the first [2] is item #3 in the main list (i.e. `6;2;4`) and then [1] calls the second item in that nested list, returning `2`.

Square brackets: dictionary keys with multiple values

Another case where a nested list might be used is where a dictionary key's value is a list:

```
$MyDictionary = {dog:terrier;boat:yawl;fruit:[apple;pear]};
$MyFruit = $MyDictionary[fruit][1];
```

returns 'pear', the second item in the (zero-indexed) list that is the value of key 'fruit'.

Square brackets: in documentation, optional arguments

In *documentation* of code, as opposed to code itself, the common technical writing convention is followed whereby something used optionally, i.e. that which may be omitted, is placed inside [ ] brackets. In this case the [ ] are not part of the syntax used in code, but they simply indicate the optional omissions

An obvious place to see examples of this is in aTbRef's full list of operators.

As an example, the if() conditional test:

```
if(condition){...actions...}[else{...actions...}]
```

Note the second part starting `else` is inside `[ ]` square brackets. This implies that both these are valid syntax:

```
if(condition){...actions...}else{...actions...};
```

or

```
if(condition){...actions...};
```

Curly brackets: dictionaries and nested dictionaries

The newer { } notation allow Dictionary data type objects to be defined, or to define dictionary-type data as an item in a list. This is similar to the newer to the [ ] method for defining List or Set types, or nested list values,Thus:

```
$MyDictionary = {dog:terrier;boat:yawl;fruit:apple};
```

which replaces the older (still working) syntax:

```
$MyDictionary = dictionary("dog:terrier;boat:yawl;fruit:apple");
```

Just as a list can be used as a dictionary key value (see), a { } object can be used as a list item or a key value:

```
MyDictionary = {boat:yawl;plant:{fruit:apple;veg:pea};dog:terrier};
$MyValue = $MyDictionary[plant][veg]; gives 'pear'
```

Here a list is nested in a dictionary:

```
MyDictionary = {boat:yawl;fruit:[apple;pear];dog:terrier};
$MyValue = $MyDictionary[fruit][1]; gives 'pear'
```

Caution is advised in nesting deeply. So far only a single level of nesting has been well tested. Deeper nesting might result in unexpected outcomes.

Note that Dictionary keys have no notion of a sort order so a [N] numbered address must not be used in place of a [keyname] named address.

Curly brackets: defining code blocks

Action code operators that generates loops and branches enclose such section of code in { } brackets. Examples are below:

Conditional

```
if($Text.contains("Project"){
 // code if query evaluates true
}else{
 // code if query evaluates false
};
```

Loops

```
$MyList.each(aPath){
 create(aPath);
};
```

Note that inside a loop, the loop variable—here `aPath`—only holds a value within the loop code enclosed by the { }. For each iteration of the loop, the value of `aPath` is the value of the item in the source list. So, for the first loop `aPath` is th
value of first item of $MyList.

Caret delimiters: export code operators

The caret ^ character is used with export code operators *only*. A caret is used to start the operator and (optionally) a closing caret. In the latter is omitted Tinderbox will output export correctly. Thus:

```
The value is: ^value($MyString).
```

or

```
The value is: ^value($MyString)^.
```

For the novice, it is safer to always use a caret as start *and* end of the the operator. aTbRef uses the latter convention for code examples. If reading this note in the aTbRef, all carets used in examples will be doubled, so that there don't get evaluated as actual export code when the note is exported for the website.

Note that use of export code in action code is strongly deprecated. It was allowed in very early Tinderbox so may still work. But such mixing should not be used for new code.

Export code does allow use of action code in export templates via the ^action()^export operator.

Forward slash: folder delimiter in paths

When describing the path of a note in Tinderbox, each container (folder) within which the note is nested is delimited (separated) in the path text by a forward slash.

So if 'Project A' is a child of 'Projects' which in turn is a child of root-level container 'Work', the path (as also found via $Path) is:

```
/Work/Projects/Project A
```

Note: this is why it is a good idea—if possible—to avoid using a forward slash character within a note name as it can prove problematic for some Tinderbox automation operations. Thus titles like "Project A/B" or "Project A (Part 1)" are not generally recommended if the title can be written a different way to avoid these characters: e.g. "Project A-B" or "Project A, B" and "Project A [Part 1]".

Double forward slash: action code comments

Action code allows commenting in code, by using a // double forward slash. Here the second line of code is a comment and ignored when the action is run. Only lines #1 and #3 are evaluated as expressions.

```
$MyList = [ant;bee;cow];
// now set MyString to the second item in MyList
$MyString = $MyList[1];
```

Backslash: escape character

The backslash character `\` is used in literal strings in action code to 'escape' the character. This is necessary where some characters have a special meaning. So a '(' can open a parenthesis in a text string, i.e. indicate to a human reader a side-note in brackets. But if that string is evaluate for regular expressions, both '(' and ')' have a special meaning and so must be escaped). Thus the literal text string:

```
"Do this (only) before that"
```

if it will be evaluated for regular expressions, e.g. when used with String.contains(), might be seen written as:

```
$Text.contains("Do this \(only\) before that")
```

These are also a few cases where (borrowing from regular expressions methods) escaped letters are used in general action code. The two most common are to indicate a line break or a tab character:

- Line break: `\n` creates a single non-printing (i.e. invisible) line break character.
- Tab: `\t` creates a single non-printing (i.e. invisible) tab character.

Full stop: dot-operators

The full stop, or period, or point, or dot is normally used to mark a sentence end in text.

A full stop is also seen used in one special context in action code: chaining operators together. Operators that can do this are referred to in Tinderbox as 'dot operators. For instance in this:

```
$MyList = $MyList.unique().sort();
```

The list stored in attribute called 'MyList' (`$MyList`) has been de-duplicated (`.unique()`) and A–Z sorted (`.sort()`) and written back over the original value.

A listing of dot-operators is available.

Comma: function argument delimiter

For arguments (i.e. the input parameters) of action code operators and user functions, if more than one argument, each is separated by a comma.

Some operators take no arguments.:

```
version();
```

Some take a single argument:

```
tan(radiansNum);
```

But many take two or more arguments:

```
time(aDate, hoursNum, minutesNum, secondsNum)
```

and here each argument is separated by a comma. White space either side of the comma is ignored.

Note that this differs from other lists in action code where semicolon is used.

If no arguments exist or are all optional, they may be omitted. If in doubt, supply the empty parentheses. These examples both work and are functional similar:

```
$MyList = $MyList.unique().sort();
$MyList = $MyList.unique.sort;
```

The .unique() operator never takes an argument, but .sort can, so another pair of alternates are:

```
$MyList = $MyList.unique().sort($Year);
$MyList = $MyList.unique.sort($Year);
```

Symbols used in Mathematical and Logical operations

A number of non-letter characters are used in mathematical operations (note this isn't exhaustive, and doesn't list uncommon mathematical operators):

- Addition: + 'plus', 'plus sign'. See here. Note the different use from action code concatenation.
- Subtraction: - 'minus', 'minus sign' or 'hyphen'. Note this is discrete from the typographic en-dash '–' (⌥+-) and em-dash '—' (⌥+⇧+-). See here.
- Multiplication: * 'asterisk'. See here. Note that in note $Text the asterisk is also used for generating (export) ' quick lists'.
- Division: / 'forward slash' or 'solidus'. See here. Note the separate use in Tinderbox path values.
- Equals: = 'equals sign'. Note this is also used for assigning values in action code expressions.
- Less than: < 'less than', 'left angle bracket'. See here.
- Less than or equal to: <=. See here.
- Greater than: > 'greater than', 'more than', 'right angle bracket'. See here.
- Greater than or equal to: >=. See here.
- Increment: +=. See here.
- Decrement: -=. See here.

In addition there are logical operators using single or double characters:

- Equality: ==. See here.
- Inequality: !=. See here.
- AND: &. See here.
- OR: |. See here.
- NOT: !. See here.
- Logical AND: &=. See here.
- Logical OR: |=. See here.

Symbols used in Regular Expressions

A number of non-letter characters are used in regular expressions where they have special meanings: `^$\()[]{}+*=?`. If using regular expressions do take some time to read about them: the subject is too large to address here.

Designators in actions

Both item and group scoped designators are allowed in attribute references. For example:

```
$MyList=$Color(children);
```

finds a list of the colours ($Color) of each child of this note, and

```
$MyList=$Color(children).unique;
```

finds the same list but without duplicates. N.B. lists can hold duplicate values.

Working with numerical values:

```
$MyNumber=$Width(children).max;
```

will find the maximum width of the container's children.

When the same designator is applied to the attribute $Text

```
$Text=$Text(children);
```

the texts of each child are placed in this note's $Text, separated by paragraph breaks. To append the same child text to existing text:

```
$Text=$Text+"\n"+$Text(children);
```

Designators

Tinderbox supplies a number of 'designator' terms, use of which instead of a note's $Name or path which may infer an individual note, or in a few cases a group of notes. Item designators are generally hierarchical in their references. Designators ignore map adornments, so cannot be used to match adornments.

Designators are generally used with attribute value references (`$Name(parent)`), as action or export code arguments (`linkTo(grandparent)`, `linkFrom(grandparent(original))`). In some cases cases they are specific to certain operators, as with links() and the link direction designators.

Most item and group designators can use paths (or note name, if unique) to refer to data from a different note from the current context, e.g. `descendants("Projects")`, or use other designators as path/group proxies, e.g. `grandparent(original)`. This makes them even more flexible in use: see the section on paths for more detail.

However, designators may not nest the find(query) designator inside another designator.

Both item and group scoped designators are allowed in attribute references. For example:

```
$MyList=$Color(children);
```

finds a list of the colours ($Color) of each child of this note, and

```
$MyList=$Color(children).unique;
```

finds the same list but without duplicates. N.B. lists can hold duplicate values.

Working with numerical values:

```
$MyNumber=$Width(children).max;
```

will find the maximum width of the container's children.

When the same designator is applied to the attribute $Text

```
$Text=$Text(children);
```

the texts of each child are placed in this note's $Text, separated by paragraph breaks. To append the same child text to existing text:

```
$Text=$Text+"\n"+$Text(children);
```

There are also separate series of designators designed to help date creation/calculation and the polarity of links.

Designators are generally used with attribute value references (`$Name(parent)`), as action or export code arguments (`linkTo(grandparent)`, `linkFrom(grandparent(original))`). In some cases cases they are specific to certain operators, as with links() and the link direction designators.

Most item and group designators can use paths (or note name, if unique) to refer to data from a different note from the current context, e.g. `descendants("Projects")`, or use other designators as path/group proxies, e.g. `grandparent(original)`. This makes them even more flexible in use: see the section on paths for more detail. However, designators may **not** nest the find(query) designator inside another designator.

Designators and root-level properties

From v9.6.0, Tinderbox no longer forbids expressions that interrogate properties of the root note—the parent of top-level notes. For example, a top-level note can now get the value of `$MapBackgroundColor(parent)` if it wants to know t background colour of the top-level map.

Using designator values as note titles

Tinderbox does not regard designators as 'reserved' values. It is perfectly possible to have a note called "children". However, in action or export code, if referring to such a note via code always use the full path ($Path) of the note rather tha title ($Name) to avoid ambiguity as to the intended object of the reference.

Quoting designators vs. paths or /titles

By convention, designators are normally not quote-enclosed in action code (i.e. queries and actions (and most export code)). Although designators are strings, not using quotes helps Tinderbox's parser to identify them as designators as opposed to notes of the same name. Designators used as path proxies, e.g. $Name(parent), may safely be quoted. Historically, path arguments were not quoted but since v5 there has been a move to explicitly quoting all string values. Ideally do not use notes with a $Name matching a designator also using designators in that TBX's actions (using explicit paths can route around such name collisions).

Designators for Date data construction

Designators, or rather 'keywords', for use defining date/time elements in Date-type data are described separately.

A listing of designators

Designators are listed via their scope:

- Item Note Designators
- Group Note Designators
- Link Designators
- Miscellaneous Designators
- root

Item Note Designators

You sometimes need to specify a reference to a note relative to something else, like the current note. Item Objects are a set of generic relative references to, or placeholders for, another single [sic] note whose actual identity is context sensitive. Item designators are written with lower case starting letters and inter-capitalised if multiword; e.g. child, prevSibling. Usage is case sensitive.

The 'order' of notes, where referred to, relates to their sequence within the overall Tinderbox file. To see this order, expand all levels of an Outline, Explorer or Chart view. The order is the top-to-bottom ordering of the notes on a line by line basis; so the first child's children are ordered before the second child, etc. (Outline order is also exposed via the read-only System attribute $OutlineOrder).

For convenience in export, several designators skip notes that are not exported, or usually considered outside the document outline. These include:

- adornments
- separators
- other notes which do not export

Such designators include next, previous, nextSibling, previousSibling. From v9.6.0, to designate a note that might be an adornment, separator, or unexported item, use the corresponding item designators: nextItem, previousItem, nextSiblingItem, previousSiblingItem.

Group-scope designators may return only a single item if the evaluated group only contains a single item. This is not the same as the expectation that an item-scope designator must return a reference to only a single item.

- $ID value
- adornment
- agent
- child[N]
- cover
- current
- find(condition)
- firstSibling
- grandparent
- lastChild
- lastSibling
- next
- nextItem
- nextSibling
- nextSiblingItem
- original
- parent
- previous
- previousItem
- previousSiblingItem
- prevSibling
- randomChild
- that
- this

$ID value

The $ID value of an item may be used as a special designator for single items or groups.

For example:

`$Name(1524673590)` locates the $Name of the note whose $ID is 1524673590.

`$Name(1524673590;1524673572)` locates the $Name of the two notes whose $IDs are listed.

Note that this particular designator usage should be avoided whenever possible, as it is hard to read and thus confusing for the occasional user. However it is made available because in some circumstances it may be the only practical way to reference item(s) where relying on unique paths are impractical.

adornment

The designator **adornment** allows notes on an adornment to refer to the adornment's attributes. Only available in an adornment's $OnAdd action and in no other contexts. This is similar to the ' Link type honouring operators' designator.

In this context, using parent would result in the value being taken from the parent of the current map view.

In all other contexts than the above, the **adornment** refers to the map view adornment(s) on which the current note rests. If the note overlaps two or more adornments, **adornment** designates the *uppermost* (front-most) adornment.

If the note overlaps no adornment, then **adornment** is bound to this. For example, the rule

```
$Color=$Color(adornment);
```

will change the note's colour to match the colour of an adornment, but leaves the colour unchanged if the note does not overlap an adornment. This latter usage, with an inherent 'if on adornment' switch makes for more flexible coding in map

agent

The designator **agent** allows the children of an agent (always aliases) to refer to the agent's attributes. N.B. this is *only available in agent queries and actions* (i.e. $AgentQuery and $AgentAction) and no other contexts. This is similar to the 'adornment' designator.

In this context, using parent would result in the value being taken from the parent of the original note to which the alias refers.

child[N]

child

The designator **child** represents the oldest child (note) of the note currently in focus, thus the first as listed in Outline view. Mention in various references of 'oldest' deriving from creation/modification date are erroneous. 'Oldest' equates to th note whose $OutlineOrder attribute value is 1 greater than the current note's OutlineOrder value.

The logical opposite of this (first) **child** designator is lastChild; there is no firstChild designator.

child[N]

The designator can take an optional numbered offset **N**, in square brackets. N numbers up from zero and down from -1. Thus:

`child[0]` is the same as **child**

whilst:

`child[1]` is the second child

`child[-1]` is the last child (i.e. **lastChild**)

`child[-2]` is the last but one child

Legacy code issue

To avoid ambiguity, the old *group*-scope '**child**' designator is deprecated in favour of ' children'.

cover

The designator **cover** represents the top level note in the hierarchy, or the first listed if there is more than one at this level. It will be the first listed in an outline window that shows the whole document. It has an $OutlineOrder value of '1'. Th cover note will be the first listed root note if there is more than one note at the highest Outline level.

current

The designator **current** refers to the note which Tinderbox is currently exporting. Where a note is exported to its own page, **current** is the same as **this**.

However, if a note includes children or other notes, **current** differs from **this** in the included notes reflecting the note actually being processed (and not the note that initiated the export). It is extremely useful for boilerplate-style includes that need to use attribute values from the calling note.

For instance, if 'Note X' is calling an ^include^ of the text of 'Note Y' and the latter makes a code reference to the note title then, *in the include's code*:

`^value($Name)^` gives `Note Y`.

`^value($Name(current))^` gives `Note X`.

Note that, when it appears outside an export template (e.g. in an agent or container action), **current** is synonymous with **this**.

find(condition)

Using a suitable query, find() can be used as a special item or group designator. The function returns a list of the $Path for every matching note.

In an item-scope context, care should be taken in constructing the query so that only one note is matched.

firstSibling

The designator **firstSibling** refers to the first sibling of this note, in outline order. Note that, if a note has no siblings, **firstSibling**, this, and lastSibling refer to the same note.

grandparent

The designator **grandparent** describes the parent note of the current note's parent, i.e. its grandparent.

It is especially useful for representing the relationship of an exploded note to its source note as the former do not automatically inherit the latter's attributes.

lastChild

The designator **lastChild** refers to the last immediate child of the current note. For example:

I. A note
    Ia. Another Note
    Ib. Yet Another Note
II. Some Note
    IIc. Subsidiary stuff
III. Different stuff
IV. More Stuff

The lastChild of I is Ib. The lastChild of II is IIc. III has no lastChild.

The logical opposite of **lastChild**, i.e. the first child, is child [sic].

Using the optional **child[N]** notation, the equivalent of **lastChild** is **child[-1]**

lastSibling

The designator **lastSibling** refers to the last sibling of this note, in outline order. Note that, if a note has no siblings, firstSibling, this, and **lastSibling** refer to the same note.

next

The designator **next** describes the next note in $OutlineOrder following current note.

next is a less closely designed test than nextSibling as as the former may be at a different outline depth.

next will *not* match an adornment, separator, or item set to not export, but see nextItem.

nextItem

From v9.6.0, the designator **nextItem** describes the next note in $OutlineOrder following current note.

nextItem differs from next in that it will match an adornment, separator, or item set to not export.

nextSibling

The designator **nextSibling** describes the sibling note in $OutlineOrder after the note in current focus; the next note on the same level, i.e. excluding children.

nextSibling is a more closely designed test than next as as the latter may be at a different outline depth.

nextSibling will *not* match an adornment, separator, or item set to not export, but see nextSiblingItem.

nextSiblingItem

From v9.6.0, the designator **nextSiblingItem** describes the sibling note in $OutlineOrder after the note in current focus; the next note on the same level, i.e. excluding children.

nextSiblingItem differs from nextSibling in that it will match an adornment, separator, or item set to not export.

original

The designator **original** is synonymous with **this** for all notes except alias. In the context of an alias, **original** is the alias's *original note*.

This designator is most useful in the context of an agent's action. For instance:

```
$Container="/Bin";
```

will move any matching aliases to the root-level note 'Bin' but the query will spawn a new alias next cycle; unless stopped, 'Bin' will fill with aliases as new ones are added each agent cycle. However:

```
$Container(original)="/Bin";
```

will move the original not of the alias to 'Bin' causing the original to be out of scope of the agent query and thus having the expected effect.

parent

The designator **parent** describes the note containing the note currently in focus. In Tinderbox there is only one parent note per child or set of sibling children.

Testing the 'parent' of an alias

The **parent** of an alias is the alias' own container (note or agent), not that of its original note. However, in more complex scenarios care needs to be taken to establish which alias/original note the user intends to reference. Where multiple aliases exist whether inside agents or elsewhere, there is scope for ambiguity if the user is not careful with their code. When using parent with aliases there are several useful considerations

- The original designator, as in parent(original), will unambiguously indicate that it is the parent of the alias' original note that is being referenced
- Only in agents, the Link type honouring operators designator can be used to allow agent aliases to refer to their 'parent' rather than the parent of the original note on which the alias is based.
- The $Container attribute is intrinsic to an alias and potentially less ambiguous than $Name(parent) if trying to establish an aliases parentage within code.

previous

The designator **previous** describes the note in $OutlineOrder preceding current note.

previous is a less closely designed test than prevSibling as as the former may be at a different outline depth.

previous will *not* match an adornment, separator, or item set to not export, but see previousItem.

previousItem

From v9.6.0, the designator **previousItem** describes the note in $OutlineOrder preceding current note.

previousItem differs from previous in that it will match an adornment, separator, or item set to not export.

previousSiblingItem

From v9.6.0, the designator **prevSiblingItem** describes the sibling note in $OutlineOrder before the note in current focus; the previous note on the same outline level with the same parent, i.e. excluding children.

prevSiblingItem differs from prevSibling in that it will match an adornment, separator, or item set to not export.

prevSibling

The designator **prevSibling** describes the sibling note in $OutlineOrder before the note in current focus; the previous note on the same outline level with the same parent, i.e. excluding children.

Note that whilst **prevSibling** is the normal assumed spelling, the alternate of previousSibling is supported, though deprecated.

prevSibling is a more closely designed test than previous as as the latter may be at a different outline depth.

prevSibling will *not* match an adornment, separator, or item set to not export but see previousSiblingItem.

randomChild

The designator **randomChild** returns a randomly-chosen child of the current note (i.e. this note). If the current note has no children, **randomChild** returns the name of the current note. For example:

```
$MyString = $Name(randomChild)
```

that

The designator '**that**' refers to note running a query within find() code. As expressions like find() change the meaning of 'this', **that** provides access to the original value of this. **that** is evaluated in stamps.

For instance, it makes it possible to to use an attribute value from the calling notes within a find() query in an expression. In such contexts, 'this' would apply to the note being queried by find() rather than the note calling find().

For example if Note A, has 'Note B' stored in $MyString and runs a rule (or some other action code) the following would fail to test each note in find() for a $MyString value of Note B:

```
find($MyString(this)=="Note B")
```
Fail!

But:

```
find($MyString(that)=="Note B")
```

will test each note in scope of the find() query for the $MyString value stored in the calling note, i.e. Note A.

this

The designator **this** describes the current note (i.e.the note *currently in focus*). The focus of this can vary slightly by context:

- in $Rule & $Edict: the note whose rule is running
- in $OnAdd/OnRemove: the note that is being added/removed. In context, an attribute *with no designator* thus refers to the note running the action
- in $AgentQuery: the note that is being examined
- in $AgentAction: the alias that is being examined
- in find(), collect(), etc.: the note that is being examined

Beware of nesting changing the context. Consider the following rule:

```
$Text=find($MyString==$Name(this))
```

Although the overall context is a rule, the rule contains a find, which in turn holds the 'this' designator. Therefore in the above example 'this' is evaluated in the context of the nested find() and not as in a simple $Rule. The problem is resolved by using a different designator: see 'that'.

Group Note Designators

You sometimes need to specify a reference to a note relative to something else, like the current note. Group designators are a set of generic relative references to, or placeholders for, a group [sic] of notes whose actual identity is context sensitive. Group designators are written with lower case starting letters and inter-capitalised if multiword; e.g. *sibling*. Usage is case sensitive.

Unlike item designators, group designators may **not** be use for specifying attribute path references.

Thus, for a word count of the current note's children you use:

```
$MyCount=sum(descendants,WordCount)
```

In older versions this would have required using an export code:

```
$MyCount=^sum(descendants,WordCount)^
```

Some action operators and export codes work on groups of notes. These can use group designators in place of literal lists (or attribute-stored lists).

If a group is empty, e.g. the current note has no children, the group operator returns the equivalent value/boolean for the current note instead.

It is perfectly valid for a group designator to return a single item, if only one item meets the criteria of the designator. A container with only one child note with return list containing only item reference for its 'children'. Conversely, (single) iter scope designators must return a single item reference.

- [literal group assignment lists]
- $ID value
- adorments
- all
- ancestors
- children
- descendants
- find(condition)
- siblings

[literal group assignment lists]

Group designators may also be lists of notes, either explicitly

```
$Color(/config/one;/config/two)="red"
```

or implicitly computed by an expression equating to a list enclosed in double-quotes:

```
$Color("collect_if(all,$Path,$Price>5)")="red"
$Color("collect_if(all,Color='red',$Path)")="blue"
```

Note in the latter example how string literals within the expression use single-quotes to avoid premature closures.

Another new option is a list implicitly computed by a find() query expression: `$Color(find($Price>5))="red"`

```
$Color(find($Color="red"))="red"
```

Note that find() queries do *not* need to be double-quoted, unlike the earlier examples above.

$ID value

The $ID value of an item may be used as a special designator for single items or groups.

For example:

`$Name(1524673590)` locates the $Name of the note whose $ID is 1524673590.

`$Name(1524673590;1524673572)` locates the $Name of the two notes whose $IDs are listed.

Note that this particular designator usage should be avoided whenever possible, as it is hard to read and thus confusing for the occasional user. However it is made available because in some circumstances it may be the only practical way to reference item(s) where relying on unique paths are impractical.

adorments

The designator **adornments** returns all the adornments that are contained inside the current note, i.e. this container's child map. The data is returns as a List of the adornments' $Path values, in $OutlineOrder.

Siblings vs. children

For any given map, in terms of $OutlineDepth/$OutlineOrder, notes on the map are *siblings* of the maps' adornment(s). However, for **adornments** any adornments are for the addressed item's *child* map. If the logic seems odd, bear in mind that a map (timeline, etc.) is normally addressed in terms of its enclosing—and thus parent–container.

Thus, for an item on a map to get its *sibling* adornments use either of these relative references, use either:

```
$MyList = $Name(adornments(parent))
$MyList = $Name(adornments(..))
```

noting that `..` is the unix term for the parent folder. For most Tinderbox users, who aren't familiar with Unix structures, the `parent` designator as the offset address argument is the most sensible to use.

Title vs. ID and action code

When using a large number of adornments, e.g. to set out a time or date grid, many of the individual adornments may not have a unique $Name. For instance if there is an adornment '10' mapping the first 10 minutes of an hour, there may be multiple adornments on the same map with the same $Name and $Path. Recall that when accessing a note via a not unique title/path, Tinderbox *always* matches the first match by $OutlineOrder, even when processing a list. In other words,

processing a list of $Name values with multiple '10' values, the action will act on the same, first match target, every time.

$ID offers a sensible workaround, not least because within the action code context the use of item titles ($Name) is a convenience. If you want to process a list of adornments for whatever ever reason it may therefore make sense to use their $ID instead as this will always be unique. For example:

`$MyList = $ID(adornments)` → a list of child adornment IDs

`$MyList = $Name(adornments(parent))` → a list of sibling adornment IDs

all

The designator **all** describes all notes in the current TBX. 'all' is assumed as a value by some group-based operators if no explicit group name is supplied.

It appears **all** is explicitly supported for group-based work in actions & rules, but not for group-based export codes.

ancestors

The designator **ancestors** describes all notes in a direct line of descent from the cover to the current note. Also, if there are siblings at the top level, it may be to the root note for the section of the hierarchy containing the current note.

children

The designator **children** describes any direct child of the current note.

This designator removes the contextual ambiguity of older usage of 'child' for both group and item scope usage. By comparison the child designator is always assumed to apply to item scope (i.e. one referenced container only).

As designators ignore map adornments, a container note with just adornments on it but child notes with evaluate as having a $ChildCount (the number of adornments) but no **children** (the number of notes).

descendants

The designator **descendants** describes all direct and indirect children of the current note, i.e. all those for whom the current note is an ancestor.

find(condition)

Using a suitable query, find() can be used as a special item or group designator. The function returns a list of the $Path for every matching note.

In an item-scope context, care should be taken in constructing the query so that only one note is matched.

siblings

The designator **siblings** describes all notes at the same outline level as the current note that share a common parent.

Link Designators

When referring to notes in the context of links links, two designators are offered to help differentiate link direction and thus at which of the two connected notes a reference should be pointed. Treat usage as case sensitive.

These designators can be used within expressions (e.g. as attribute offset addresses) within expressions used in link-creation export codes.

- destination
- source

destination

Used (only) by export codes that create link lists and action code links(), and by OnLink actions.

The designator **destination** refers to the note *to which* the exported links will point. This object allows users to access data generically from the destination of the link.

See also: source.

For example,

```
^basicLinks(<ul>,<li>," : (modified ^value($Modified(destination))^)</li>",</ul>)^
```

will export a list of links, each destination note's Name followed by followed by the modification time for the link's destination. Note this designator cannot be used with ^value^ and attribute references.

Note that the designator can be used in the link export code's item prefix and/or suffix arguments but this designator cannot be used with ^value^ and attribute references. Meanwhile, the link's anchor text always remains the $Name (i.e. title) of the link's destination note, linked to that note's page's URL.

source

Used (only) by export codes that create link lists and action code links(), and by OnLink actions.

The designator **source** refers to the note *from which* the exported links will arise. In most (all?) cases this equates to both this and current, so the object 'source' makes it easier for the user to specify generically from where data is drawn.

See also: destination.

For example,

```
^basicLinks("<ul>","<li>^value($Name(source))^ => ","</li>","</ul>")^
```

will export a list of links, each destination note's Name preceded by the source note's name and a => pointing to the destination note's name.

Note that the designator can be used in the link export code's item prefix and/or suffix arguments but this designator cannot be used with ^value^ and attribute references. Meanwhile, the link's anchor text always remains the Name (i.e. title) of the link's destination note, linked to that note's page's URL.

Miscellaneous Designators

The are a few designators that are used in particular contexts:

- asterisk ("*")
- document
- my

asterisk ("*")

The linkedTo() and linkedFrom() action code query operators can use a wildcard designator which has whole-doc scope, akin the the 'all' group designator. This designator must always be quote-enclosed. Note that these operators are otherwise item-scope.

document

The **document** designator refers to the overall TBX document. Used with word cloud exports.

my

The **my** designator is used in action code in the context of composites, specifically when referring to the composite containing the note using the code, i.e. this note.

my can be used as a short form of `compositeFor(this)` when using with colon-operators used (only) with composites. For example, the pair of examples below have the same functional effect:

```
compositeFor(this):role("someRole")
my:role("someRole")
```

Thus the use of 'my' makes for more compact code; it has no effect on performance.

If using compositeFor() in an any other context, e.g. a composite of which the current note is not a member then the 'my' designator can not be used.

root

N.B. THIS IS NOT A VALD DESIGNATOR TERM!

The **root** describes a note that has no parent, i.e. at the highest level in outline view. If a root note has no siblings then it is also the cover note. Otherwise the first listed top level note is the cover note.

Action code cannot use 'root' but the concept is relevant in the use of path references to notes.

In export codes, there is a ^root^ code.

Comments in Action code

Action code may contain comments. A comment is part of the overall code but which is not executed when the code is run. This allows annotation within code. Such comments should be brief, and usually indicate something about the intent of the code before or after it (depending on comment placement).

Comments are useful. The working of some complex code may be obvious now, but will it be when re-visited several years later?

A comment begins with two slashes, and ends either with a second pair of slashes or at the end of the line. To write a multi-line comment simple start each line of the comment with a double forward slash. For example:

```
// this is a comment that
// continues onto
// several lines
$Color = "blue";
```

In spaces using action code in the Inspector and in notes using the build-in Action prototype (e.g. Stamp and Library (function) notes in /Hints) commented lines are rendered in grey. this can be a useful tell-back

The above shows comments on a discrete line or continued on successive lines by simply starting the next line of comment with the same marker. The next example shows the two other forms. The first line of code shows a to-end-of current-line comment and the second line a fulling inline comment with valid code before/after the comment. Example, of line end and inline comments

```
if($Prototype=="Task") { // special handling for tasks
 $Width= // 5 // 7;
}
```

To recap, the three forms of comment are, in order of reading:

- a multi-line comment. Essentially, from a code perspective, this is a series of successive whole line comments.
- a line-end comment. Here, the comment continues to the end of the line, but does not affect the running of the code on the beginning of that line.
- an inline comment, i.e. within the content of a single line. The comment on the last line skips over the "5", starting before that figure and closing after it. This means that when run, $Width is set to 7 rather than 5.

Some considerations for effective use of comments

If you add comments to working code and it stops working, it may be gremlins in the parsing of the code causing the issue. What's clear to the human eye, may be less so to code trying to parse the code text. If adding comments causes a problem, don't overlook making a copy or your current code elsewhere and then deleting all the comments; doing so can save hours of frustration chasing wrong causes. If this type of problem occurs it may be one of the issues below. Meanwhile work is ongoing to resolve these edge-cases.

Avoid: soft line-wrap

Action code has no explicit multi-line comment (although colouring my imply this. Avoid guesswork, and so **do not** rely on the text area's RTF ruler's soft-line-wrap. Best practice is to put explicit line breaks after a suitable width of code in th $Text and then start a new comment line (as above) with the continuing comment text.

Avoid: double forward slashes inside a string (false code colouring results)

This can occur if something like a web url is contained in a string. consider this:

```
$Text = "https://example.com";
```

In a code input box, everything from the '//' will colour as a comment, but the action actually works—setting a $Text value. This is an edge case for the code-colouring syntax as the differing intent of '//' is not obvious to the code-parsing regular expression.

If the problem faced is trying to see the syntax (colouring) is the now-comment-coloured part of the code, temporarily delete one of the two slashes and de-dub the test of the action before re-inserting the slash.

Another workaround is to move the slashes into a variable:

```
var:string vSlashes = "//";
$Text = "https:"+vSlashes+"example.com";
```

Although needing an extra code expression/line, this has the same outcome as the previous example but should make the main section of code colour correctly.

Avoid: unmatched quotes

Within a single comment (or single comment line in multi-line) **always pair quotes**. Assume an odd number of single or double straight quotes may and likely will cause a silent failure. The code parser and the code colouring processes work slightly differently. So either pair quotes or use 'curly' quotes . Thus, Opt+Shift+] will give a single right curly quote for an apostrophe (or, avoid speech-like contractions).

Examples of what to avoid

1. Unpaired single quotes.

BAD. Unpaired straight quote:

```
// don't do this.
```

GOOD. Uses curly quote for apostrophe:

```
// don’t do this.
```

GOOD. Avoid speech-style contraction:

```
// do not do this.
```

2. Unpaired double quotes

BAD. Unpaired double quote in wrapped comment text.

```
// Here is "Some
//  wrapped quoted text". Etc...
```

GOOD. Wrapped partial quote is fully quote-enclosed:

```
// Here is "Some…"
//  "...wrapped quoted text". Etc...
```

GOOD. Uses curly double quotes for quoted text:

```
// Here is “Some
//   wrapped quoted text”. Etc...
```

Avoiding contractions is also likely kinder to translation used by those whose first language differs from our own.

Action code operator terminology explained

NOTE: This article (and its children) discusses a several potentially confusing or ambiguous aspects of action code operators and aTbRef's own conventions adopted to try and reduce that effect.

Tinderbox includes a numerous action code features called a 'operators' or simply 'codes'. 'operator' is the most used term and the one used here.

In aTbRef, operators are classified into 3 main functional groups:

- mathematical and logic operators, e.g. '**+**' (addition) '**==**' (equality test), '**&**' ('AND' query join), etc. See a listing of operator-type operators.
- property operators. These generally allow the reading and/or writing of an aspect of something, for instance separately addressing a colour's hue, saturation and brilliance. See a listing of property-type operators.
- function-type operators. These allow the manipulation of information in the TBX file, similar to—*but not the same as*—action code functions. See a listing of function-type operators.

- Action Operator Arguments
- Export Operator Arguments

Action Operator Arguments

What is the difference in meaning between a function's 'argument', 'parameter' or 'input'?

In this context 'parameter' and 'input' are just alternative terms for 'argument'. The latter is the terminology upon which aTbRef is now standardising in order to try and make the documentation as a whole more consistent. An argument will always have a value, e.g. `'12'` or `"Green eggs and ham"`, and it is the value supplied by the argument upon which

What is an 'argument'?

It can be said that an operator 'takes' one or more 'arguments '(inputs), does something (i.e. 'operates') with the *value* of the arguments, and 'returns' a result. In most cases, operators have at least once argument but a small number of Function-type operators might confuse they may have no arguments or all their arguments are optional (their being optional will be covered further below).

So then, what is an 'argument'? When an action code operator such as a function-type operator needs information to work correctly, that information is passed to the function in an 'argument'. For example, the operator **abs()** is described as abs(number), indicating the **abs()** operator takes a single argument that is a number. Other operators can take multiple arguments. For instance, any(scope,testValue) takes two arguments: a scoped argument for the group of notes the function examines, and the type of test the operator should perform.

Generally, if any of the specified arguments are not supplied, the operator will either fail to give an outcome; or if arguments are supplied but in an incorrect form, the operator will either fail or give an unexpected wrong outcome. The documentation of the operator will explain the arguments.

What do arguments do?

Above, the description has centred on how arguments are written, literally, in action code. However, is it possible to ask important questions about an argument, especially when encountering a new operator for the first time or one whose purpose is not self-evident from its name. These aspects are:

- What is the scope of the argument—e.g. is a single note affected or are multiple notes affected?
- What data type does Tinderbox expect—i.e. what information must Tinderbox receive to operate the function?
- What is the purpose of the argument—i.e. what does the operator do with the information in the argument?
- Is the argument evaluated—i.e. how does Tinderbox interpret the information in the argument?

Argument order

In action code arguments must always be supplied in the order they are documented. This also applies to arguments for user-defined functions. If an (optional) argument is omitted, and other arguments follow it, a comma+space placeholde

should be inserted for example, in pseudocode:

```
someOperator(arg1, optArg2, optArg3)
someOperator(arg1, , optArg3)
```

Argument delimiters and whitespace

A comma **must be** placed between each argument in function operator. Without the commas, Tinderbox is unable to correctly process the instructions. It is acceptable to type a comma followed by a space, for readability purposes, but that space is not required by Tinderbox and is ignored by the parser. Thus Tinderbox parses both the following examples to the same outcome:

```
any(children,$Checked)
any(children, $Checked)
```

Quoting argument content

The literal string parts of expressions are quoted as in normal action code, in straight quotes, e.g. `"Name of some note"`. Single or double quotes may be used as long as correctly paired. If the literal value of a complete path (i.e. starting with a '/') is used in code the quotes may be omitted. If in doubt, use quotes!

By contrast, attribute references, e.g. `$MyString` or action code variables, e.g. `vSomeVar`, are *not* quoted.

Further Aspects of Argument Use

Programming and action coding can at times follow 'rules' that are not obvious in 'the real world'. This can be daunting at first, but careful reading of the documentation and personal experimentation with Tinderbox may quickly lead to mastery of the 'rules'. Meanwhile, those with prior coding/programming expertise will see where these 'rules' can be altered or ignored.

In the notes below, additional aspects of argument use are described:

- Optional Arguments
- Argument scope
- Argument expected data type
- Argument purpose
- Argument evaluation
- Loop variable arguments
- The implication of square brackets in operator descriptions, regex, and literal strings
- Are operator parentheses needed if there are no arguments?

Optional Arguments

What is an optional argument?

For some operators, there may be optional arguments which, if supplied, allow the operator's output to be given differently. In the main documentation listing of operators, square brackets indicate optional argument(s). For instance, with the indented(N[,"item"]) operator it takes two arguments the second of which is optional. Thus, using **intended(N)** works with only a single argument (**N**) supplied but **indented(N[,item])** can be coded with two arguments (**N** and **item**)to give a more nuanced output.

In *almost* all cases, the optional arguments are supplied last, after any mandatory arguments, but be aware there are a few where the optional argument comes first, e.g. values([group,]attribute).

Arguments may be optional because:

- A default value is assumed unless a (non-default) value is passed, thus saving having to 'state the obvious' by passing in a value that is already presumed.
- The optional argument signals the function to alter its default behaviour, e.g. giving a different form of returned output.

Argument scope

Scope: how widely in a Tinderbox document is the target operator applied?

Some arguments are designed to pass the identity of a single note or a group of notes upon which the operator then acts, thus affecting the scope of the work carried out.

The identifying data collected is usually the $Path of the note(s) but can be the title ($Name) of the note(s). In this latter case, Tinderbox will assume the $Name value is unique in the document, silently matching the first such $Name as detected in ascending $OutlineOrder and acting **only** on the first match. A more recent option is to use note $ID values, the only downside being that, read by eye, 10-digit numbers mean little although they are unambiguous to Tinderbox.

Not all arguments are scoping. Scoping arguments are most common in operators that collect or work on lists, including making or removing links.

$ID for notes with problematic characters in $Path or Name

As documented, some characters—such as parentheses '(' or forward slashes '/'—can in some circumstances confuse Tinderbox's code parser. The most likely context is when using (un)linkTo/From operators. Such usage is a good candidate for using $ID data instead of paths is the current document is known to contain 'difficult' paths or titles (bearing in mind that a path is effectively a list of ancestor note's titles).

Argument expected data type

Data type: what Tinderbox 'data type' is needed by an argument or its target operator?

Whether explicit in the argument name or not, most argument will have a documented or implicit data type. This worth bearing in mind especially when passing attribute values or variables as the argument value.

If the source data type is incorrect, it *might* be necessary for Tinderbox to 'coerce' the data from one form to another at which result in an unexpected outcome. In programming, when software makes one kind of data into another, e.g. makes a string of digits into a number, that precess is called 'coercion'.

Experienced users will likely know where this might occur and act accordingly. New users are advised to avoid coercion, which might have unexpected results, by first formatting their data to the desired type and only then pass the result as the argument. For example, if a function is looking to receive a numerical argument do not give it a quoted string of digits such as `"12345"` but the numerical value `12345`. Practice and experience will soon help this become clear.

Another aspect to this is knowing the data type upon which the operators work (without implied coercion). For instance, there are a suite of dot-operators that work only on **Color** data-type, or only on **Date** data-type, etc. Thus, even if the arguments do not stipulate a data type, be aware of the data type being passed into, or literally written as, the argument' value.

Another aspect of the data type is the likely range of acceptable vales, which may not be explicitly stated. For example, a number of opacity-related system attributes use a number representing a percentage between 0% (value: `0`) and 100 (value: `1`). Using `10` (100%) instead of `0.1` (10%) when intending to indicate '10%' may confuse Tinderbox. Even if it treats a value of greater than 1 as 1, that will result in an opacity 10 times that expected. Where value range limits are known they will be documented.

Failure to take account of data type when passing argument values is likely to result in unexpected results or silent failure—no results and no explanation or indication as to why. Should he latter occur, checking the argument values is a good place to start looking for an explanation.

Argument purpose

Purpose: what does this argument do, with relation to its target operator?

It may not always be obvious from the syntax example name of the argument as to its purpose when used. For a simple task like getting a square root of a number it should be obvious that the operator will need to know the number from which to obtain a square root. Sure enough, the sqrt() operator takes one argument 'number'. By comparison, the pow() operator for raising a number to a power;; it takes two number type arguments. Although the purpose of the two can be guessed, it is still necessary to know which argument is the number acted upon and which argument is the desired order to which it is to be raised.

Arguments are documented as part of articles describing discrete operators. Where possible, aTbRef will attempt to indicate an argument's purpose within a given operator, though the name used in short syntax examples may not give all detail about that argument. If unsure, the user may need to consult the full documentation article on the relevant operator.

At present there is no clear, unambiguous method for naming arguments that works across the whole set of operators, so if new to use of an operator, do read the note (and where necessary its linked notes) before first use.

Argument evaluation

Evaluation: does this argument allow evaluation of expressions?

An argument's value can be defined in a few ways:

- A literal value: `2`, "bar", "2 February 2022, 22:22:22", etc.
- A designator—only for some argument types—such as scoping arguments, dates, etc.
- An action code expression
  - A simple expression, e.g. simple string concatenation like `"Hello"+" world"`.
  - A complex expression where some calculation is required to generate the equivalent of a literal value.
- An attribute or variable value holding either a literal value or an expression.

Note that few operators mandate only use of literal argument values. However, Tinderbox's own documentation does not generally address (degree of) argument evaluation. This can mean that at times a use *may* need to experiment before applying complex argument values to operators in a working document.

Simple evaluations are generally not problematic and, in most cases, no extra preparations are needed.

Complex expressions can arise, often unintentionally, because operator X's argument needs a value being calculated by process Y elsewhere in the document. Thus, an attribute's value might be a call to a user function. To get the literal value of the argument, Tinderbox must read the argument, finding it is an attribute reference, fetch the attribute's value to find that is a function call, call the function, receive the function output … and hope that result is a suitable literal value.

Another aspect is that the scope and capability of Tinderbox action code has changed significantly over the application's 20+ year lifespan. The range of possible interactions between operators can create scenarios where deeply nested evaluations (i.e., evaluation within evaluation within … etc.) arise, and importantly occurring without the user being aware of that. Insufficiently deep evaluation can occur. If it does this should be reported directly to Tinderbox tech support as it will be unintentional unless the limitation is formally documented. Generally, such limitations are resolved, but if encountering them be aware that any fix will require a new app built/release, so in the short term another approach to the problem is required.

aTbRef will document evaluation limits where known, but as stated this aspect of the action code toolbox's interactions is least understood or documented. Be prepared to do small/limited tests if in doubt.

Loop variable arguments

Loop variables

A small number of operators (see a listing) employ a *user-named* 'loop variable'.

Loop variables, or 'loopVar', are a special form of input. Though not strictly an argument, a loop variable is most easily understood in the usage for list.each(loopVar){} operator.

The important point to understand, and which confuses new users, is that the 'name' if the loop variable is whatever *the user chooses*. So, if coding an **.each()** loop like this:

```
$MyList.each(anItem){...}
```

the variable's name inside the `{ }` code block is 'anItem' or if coded:

```
$MyList.each(x){...}
```

it is 'x', or:

```
$MyList.each(foobar){...}
```

is it 'foobar', etc.

A key point is using the loop variable's name *anywhere inside the loop* (i.e. within the `{ }` curly braces) the current value of the loop variable is the value of the source list item currently being processed. In the first example above, on the first iteration of the loop 'anItem' has the value of the first item in $MyList, in the second loop the second $MyList item, etc.

The implication of square brackets in operator descriptions, regex, and literal strings

Potential confusion over the implication of [ ] characters in operator descriptions vs. other uses e.g. regex and literal strings

Operator Syntax. Square brackets `[ ]` are used when documenting the optional parts of operator syntax, but are not (normally) used when employing the operator in actual action code. Otherwise, except for special cases listed below, do no type square brackets when creating code. Thus the example of the indented(N[,"item"]) operator, if using both operators the code entered would be like `indented(4,"Project X")`, noting that absence of square brackets.

For special cases where square brackets are used in literal operator syntax see:

- the attribute Dictionary data type.
- the List/Set[itemNumber] list item address operator. This is essentially an array accessor and is also used in some JSON and XML contexts. It is also an alternate syntax to List/set.at(ItemNumber).

Therefore if square brackets appear in code syntax examples—as opposed to general syntax code example descriptions—it will imply one of the above uses.

Square brackets can of course occur in *values* within the code, include code examples for using operators:

- as literal strings: `"This string contains [square brackets] because it can"`.
- in regular expressions ('regex') as part of the matching pattern definition: `$Text.contains("[^ b|d]")`.

Are operator parentheses needed if there are no arguments?

Are operator parentheses, i.e. trailing ( ), needed if there are no arguments?

Some operators have no arguments and others have only optional arguments. When there are no arguments in use, it is permitted to omit the parentheses. Both these are acceptable:

```
$MyList = $MyList.sort;
$MyList = $MyList.sort();
```

Exceptions

Most strict transactional operators (see listing) such as '`+`' or '`==`' *never* use trailing parentheses, as in—hypothetically—coding '`+()`' would be **wrong**.

Statement-type operators never take parentheses, but are used as a keyword that precedes some code defining the operator's task:

- function.
- return.
- var.

Conversely, a user-defined function defined via function must always have trailing parentheses or the parser will not detect it. Thus a user defined function 'fSomeTask' that has no defined arguments is nonetheless always coded as `fSomeTask()` with tailing parentheses.

Summary

Though optional in some cases, it is considered 'good practice' with Tinderbox action code and other programming languages to use the parentheses when using operators even then they are not strictly required. Not least, it helps remind the user that the code object is a an operator.

So, if in doubt, use empty trailing parentheses.

Export Operator Arguments

Export code operators also use arguments. Action code and export code are discrete code systems in Tinderbox with disparate tasks: one internal to to the app and the other related to exporting Tinderbox data. However, export code uses the same broad conventions for arguments as action code, so the nature and usage of arguments is covered under description of action code arguments.

Variables

Action code allow for the use of user-defined variable within action code. A primary use of these is to avoid the need for a lot of user attributes just to hold interim values during more complex actions or for re-use of a value within loops and th like.

Variables are declared in an action and cease to exist at the end of the action. To hold the value further, the existing value would need to be stored in an attribute.

Variables are created using the var() operator.

A declared variable can be passed into action code operators as arguments and as arguments to user-defined functions. In export template, a variable declared in an ^action()^ code block can be addressed anywhere in the rest of the template (following/below the point of declaration) and inserted into the exported code using ^value()^ just as if inserting an attribute value. Thus `^value(vSomeValue)^` inserts the value of variable 'vSomeValue' into the exported code stream (note: unlike an attribute there is no $-prefix used). Also see:

- Choosing to use variables or attributes in code
- Simulating global variables

Choosing to use variables or attributes in code

This is not a binary, either/or question, not least because using (user) attributes to hold values during the course of an action long pre-dates the arrival of the `var` operator for variables (which was only added in v5.10.0). Data-typed variables (e.g. `var:string`) arrived in v9.1.0, as did the `function` operator. Data-typed function arguments, a type of variable, arrived in v9.5.0. The later version also added the ability in templates for ^value()^ to inset a variable set in an ^action()^ section further up in the template. Thus a lot of longstanding code examples in the Tinderbox community were written before the current range of variable uses were added. **When must an attribute be used?** If the document needs to retain a value form session to session, i.e. persisting when the document is closed, that an attribute is required. This is because variables (and any value) are lost/deleted once an action is run. But, if such a value can be calculated from available information at document start, then storing a value in an attribute may be an advantage rather than a necessity. **When must a variable be used?** Never, but in all cases except that above, using a variable can make for shorter, clearer, action code. Additionally, as variable (values) are lost when an action completes, there is no need to clean up after. By comparison, passing an attribute a list of the paths of 1,000 items, so the list can then be iterated means that value remains forever unless the attribute is reset to an empty value.

Simulating global variables

When programming, it is sometime useful to have an arbitrary value for a something that is available anywhere in the code, regardless of the current scope; essentially a default value. Here, setting an attribute value, does the same. The exact attribute name or the location if the note storing the value isn't important, as long as having made a choice the user refers to the name/location consistently.

Since the advent of the attribute() operator in v9.5.0+, another method is simply to make a user attribute per desired global variable and set its default, altering if needs be via `attribute("AttributeName")["default"]`.

More common practice is designate a single note, often at the outline root as a configuration or 'config' note and store/alter global values there. By the same token, it is possible to use one attribute for several globals using a different note, bu the same attribute, to hold each value.

There is no correct way. With the advent of attribute() adding a suitably named/typed attribute is the cleanest method as there are not locations to remember for getting/setting the attribute. In contrast, if globals might need to be copied from one TBX document to another, e.g. to set up a particular workflow, them a config note(s) is possibly an easier approach. This aspect of Tinderbox is still evolving so what is best/safest/least effort may change over time.

Functions

Tinderbox allows user-defined operators, also called 'functions'. Such operators are typically defined in notes inside the Library container in the document's built-in Hints container (if not present this should be added to the document) though they may be defined in other contexts too.

Notes in that container which have names enclosed in parentheses, such as `/Hints/Library/(What's This)` are treated as documentation; all other notes are executed at document startup and after they are edited.

Why use functions?

Functions provide a 'define once / use multiple times' feature in documents, and allow:

- storing a process so that it can be used repeatedly throughout the document;
- defining that process once and using it from many different locations, i.e. not needing a copy of the same code in each, rule/agent/etc.;
- allowing a process that takes inputs, which is a limitation of stamps;
- allowing a process that can return results, another limitation of stamps;
- transportable function or libraries of functions that can be copied to and re-used in other document and by other users. Note: any (user) attributes called in a function need to pre-exist before use— Tinderbox will not auto-create them, e.g. when copying a function from one document to another.

What sort of things can functions do ?

Functions comprise a set of action code instructions that perform a specific set of operations to achieve a specific outcome. In most cases, this will include acting upon optional input arguments provided to the function at the point where it is called (a.k.a., invoked) and optionally return results to the code that called (called) the function. A simple case is as shown in the example below (explained in more detail here):

```
function fAddTax(iPrice){return(1.18*iPrice);};
```

It is still possible to define a useful function that takes no input and returns no value but acts on the document based on the functions code, e.g. setting a selection of attribute values. In this mode the function is akin to calling a stamp, though calling a function may seem a more intuitive usage.

Functions fall into broad categories of use:

- Mathematical/logical. Probably the simplest use. A function can easily implement mathematical and statistical calculation tasks. Generally a number in, a number out.
- String manipulation. Slightly more complex due to the ad hoc nature of the inputs. Whilst new stream parsing methods improve basic action code capability a function allows for larger processes.
- Encapsulate complex processes. In more complex documents, workflow may involve a complex series of transformations to support ingests of data or for formatted export using structure beyond normal export code offerings.

More about functions

The following sections should be read through completely on first encounter but are structured as small articles to assist cross-referencing and later reference for refreshing understanding or syntax.

Important note: all examples follow aTbRef naming conventions. These may seem verbose or over-descriptive to experienced programmers but are provided to help the *learner*, not the expert.

Aspects of using functions:

Storing function code

Important note: all examples follow aTbRef naming conventions.

Functions can be defined in *any* action code space:

- Library notes.
- Action attributes: Rule, Edict, AgentAction, OnAdd, OnRemove, etc. (see all).
- Stamps (note or Inspector).
- A note with the 'Action' prototype set (edge case, not generally recommended unless the above don't suffice)

Given their re-useable nature, Library notes are probably the most logical place for storage, but is not a requirement. Regardless of where stored, *every* function in the current document can be called from *any* action.

So, for novice users, the last option—a Library note, is the best place to start. Also, new users might choose to add only only function per note until comfortable with the process.

A library note can contain:

- one, or more, discrete functions.
- other arbitrary action code expressions, e.g. rules, agent actions.

In combination, the above two an allow Library notes to act as a way to store and share 'libraries' of code between different Tinderbox documents and/or users.

If storing code such as rules in a Library note, it may be useful to enclose such code in an inert function, thus:

```
function fDoNotUse(){..ode ...}
```

i.e. one with no inputs and no return and with a name unlikely to be called in code.

Next: Naming functions

Naming functions

Important note: all examples follow aTbRef naming conventions.

Naming Functions

Functions should not use names reserved for built-in operators. Names are case-sensitive. Function names should begin with a letter, and may contain letters, digits, or the underscore character, i.e. A-Z, a-z, 0-9 or _ (but not a hyphen!).

Functions must not share names with attributes or erroneous outcomes will occur.

The convention here in aTbRef is to use the CamelCase naming as for attributes with a lowercase 'f' prefix.

Note that parentheses are **always** used regardless of whether the function has input arguments—that being a difference from action code operators in general where trailing parentheses may often be omitted if that are no input arguments Th naming of arguments is discussed separately, here.

Function arguments are user-named an have no specified limitations but are most sensibly named using letters only (in Latin or other supported scripts).

Next: Defining a function

Defining a function

Important note: all examples follow aTbRef naming conventions.

Defining functions

Functions are defined by the function statement:

function name(arguments){actions}

The basic syntax is comprises four parts:

- the **function** action code. This tells Tinderbox that a function is about to be defined.
- the function's **name**. For function naming rules/constraints, see naming functions. The function name *must* be separated from the **function** code by a single space.
- parentheses **()** containing the function's input **arguments**, i.e. **(arguments)**. The parentheses *must* be used even if there are no arguments—that is, use **()**. See more on setting arguments. Also:
  - There must be no white space before the parentheses. White space between the parentheses and the opening { of the action code is allowed, but is deprecated so as to give an easier rule to follow: no white space before or afte the obligatory parentheses.
  - A value *must* be passed for every **argument** defined in the function. To simulate an optional **argument**, pass a null value (empty string `""`, zero `0`, `false`, etc.) and test that input within the function's code.
  - The need for parentheses when there are no arguments is a slight exception to action code operator syntax where Tinderbox is generally happy for parentheses to be omitted if there are no arguments being required/used. Thus, functions are an exception to that principle.
- curly braces **{actions}** enclosing the function's complete action code, i.e. one or more discrete code expressions. These curly braces must be used. There is no space between the preceding parentheses and the curly braces. Though Tinderbox appears to cope with white space in these positions, such use is *not recommended*: assume no white space is allowed here. Unless other code follows the function code, there should be no need to place a semi-colon after the closing curly brace.

A function is essentially an encapsulated set of action(s) which may take inputs— **arguments**—similar to the way some action code operators take input arguments. The terms 'argument', 'input' and 'argument' may be used interchangeably, as the implied meaning is the same.

Storage of function code is discussed here.

A function can be called from anywhere, as explained here.

Examples of function syntax are given here.

Can a function code contain another complete function?

Next: Function arguments

Function arguments

Important note: all examples follow aTbRef naming conventions.

Function arguments

A function's arguments are the terms—zero or more) that are defined within the parentheses following a function's name. Thus `function fSomeTask(iNumber, iString){...` defines *two* discrete arguments: 'iNumber' and 'iString'. A function with no arguments `function fReset(){...` is perfectly allowable, noting that parentheses *must* be included even if empty. This latter form, though valid, is less commonly used because in general the aim of the function's code is to manipulate one or more values passed to that function.

A function may, or may not, have one or more arguments. Multiple arguments are entered separated by commas (with optional whitespace after).

A value *must* be passed for every argument defined in the function, i.e. the action code calling the function must be written so as to supply a value for each argument defined by the function.

Naming arguments

Each input argument must have a discrete name. The argument's name *must not* be an attribute reference, e.g. "$MyString", or an attribute's literal name, e.g. "MyString". Within this restriction, the exact naming of the argument is the user's choice.

The argument name is essentially declaring/naming an internal variable for use (only) within the function, as explained further below: see 'Argument names as internal variables'.

In aTbRef documentation any function arguments in the parentheses use the same CamelCase convention as for attribute names but with an 'i' prefix. For example, this function definition code:

```
function fMakeTable(iSomeList){ ..code ...}
```

'iSomeList' will be a variable available within the scope of the function, i.e. in the function but not outside it. Calling the above function like so:

```
fMakeTable($CategoryList);
```

will result in iSomeList within the function holding a list containing the value of the calling note's CategoryList attribute.

An argument's name vs. the value name used in action code calling a function

It is important to understand that a function argument's defined name and the string defining the input for that argument as used in the code calling the function code are usually *different*. Thus, in the above example, the function is called supplying an argument value that is a reference to $CategoryList. But, inside the function for this execution of the function, the value of the first argument iSomeList will be the value read from $CategoryList in the calling note.

But, elsewhere other action code might call the same function but using code `fMakeTable($MyList);`. For that execution, iSomeList within the function will hold the value of $MyList in the calling note. Or the calling code might pass a literal list, e.g. `fMakeTable([ant;bee;cow]);`, with iSomeList now being the list passed into the function, i.e. as a list `[ant;bee;cow]`.

Despite describing calling the function 'fMakeTable' with three differently described input, in every case, the supplied argument value is used internally via the argument's name, i.e. were 'iSomeList'.

So:

- Inside: inside the function the argument name is always are defined in the function code (e.g. $SomeList in the example).
- Outside: in code calling the function, a value is substituted where the argument is defined. These may be literal values (20, "Hello", `false`, etc.) or attribute references ($MyString).

Argument names as internal variables

When writing a function's code and including argument(s), an argument's name is implicitly defined as an internal variable. Using the example above, if a function's argument were defined as 'iSortList' then that exact name (i.e. the argument's *user-defined name*) can be used anywhere in the function's code as a variable that with have that value passed to the function via that argument input.

If the argument were instead named 'x' or 'abracadabra', the same applies, the argument name as defined in the function code is the name of the internal variable for that argument's input.

Therefore, the argument/variable name is set by the user by the action of writing the function code.

Note, that when copying another user's function code, as personal styles differ, a user may decide to use attribute names more to their own style. If doing so it is important to change not only the argument name in the function's argument list but any occurrences of the argument name in the whole code of the function. In this simplistic example, function 'fGetListItem' takes a list as an argument and returns the value of the third item in that list (not second, as .at[] is zero-based):

```
function fGetListItem(iSomeList){
 return iSomeList.at[2];
}
```

If having copied this code it was preferred to have an attribute 'L', then two substitutions are needed, one in the argument declaration and the second where it is used within the function:

```
function fGetListItem(L){
 return L.at[2];
}
```

Externally, the call is still the same, e.g.:

```
$MyString = fGetListItem($SomeThings);
```

as it is only the *internal* name of the argument that has changed (from 'iSomeList' to 'L').

Argument data types

From v9.5.0, a functions arguments can be given an explicit data type, i.e. when used within the function the stated data type will be used

For example,

```
function fAppendFoo(iList:list){
 return (iList+"foo");
}
```

Now 'iList' will be treated as a list without any type coercion needed. The argument's type is set in the function definition and in the above example the function would expect one argument and that it would be List-type date (attribute, literal list-string, etc. The range of data types, and how they are used is described in the article on variables, var().

Whilst it is not a requirement to give a type to any/all arguments, it is strongly recommended users do so—especially if new to function use. Why? Because previously, all inputs arrived as a string and if a definite type was needed inside a function, the input argument had to be passed to a type variable (or attribute) before use. For example,

```
function fAppendFoo(iList){
 vr:list vList
 return (vList+"foo");
}
```

By using explicit argument types, doing explicit type coercion within the function is avoided. So, less and clearer code.

Using argument input values within the function

Arguments are a received as a String data type (**regardless** of the source data in the calling code). Note that may change in future version, to allow enforced date-typing of arguments. But, at present assume type coercion occurs. That means most inputs will be assumed to be a String, List or Dictionary.

In the example above, calling the argument 'iSomeList' may indicate the term is a function argument and is *intended* to be List-type data, but it doesn't affect Tinderbox normal type coercion. A string with no semi-colons in it will be assumed to be a string. If semi-colons are found a list (essentially a List type) will be assumed.

To avoid incorrect parsing of inputs, it may be necessary to pass the input to a data-typed variable. Re-using the above example:

```
function fMakeTable(iSomeList:list){
 var:list vList = iSomeList;
 // ..tc.
}
```

Variable use is discussed further here.

It is equally possible to pass the argument to an attribute of the desired data type in order to assert data type for further use within the function.

Evaluation of input arguments

An argument my be an expression as well as an attribute value, variable or literal string/number. For instance:

```
$Result=fLastThreeWordsOf(fBestParagraphOf($Text));
```

Here, the actual value passed to function `fLastThreeWordsOf()` is the evaluated result of `fBestParagraphOf($Text)`, with $Text being the text of the calling note.

Simulating optional arguments

As stated above, *every* argument defined in the function's code, *must* receive an import or the function will not run correctly. To simulate an optional argument, pass a placeholder string ("none") and test that input within the function's code before using the argument, i.e. if "none" do nothing or else carry out some action using the argument's value. Here "none" is not a defined term. The value used in this context is up to the user. The point is that if using 'no value' value, the the input value must match whatever 'no value' string the user has defined in their function.

Next: Returning function values

Returning function values

Important note: all examples follow aTbRef naming conventions.

Returning function values

return

The return statement returns a value from a function to the calling action code object (which can even be another function). The **return** operator is used *only inside* a function. Execution of the function's code stops once an expression containing a **return** (i.e. up to the next semi-colon) is processed. Thus if function fTest returns a string:

```
function fTest(){
 return "Hello";
}
```

The returned value is passed back to the calling code:

```
$MyString = fTest();
```

with the result that $MyString would be "Hello", i.e. the value of the function's **return** statement. By contrast, simply calling the function without a left-side recipient:

```
fTest();
```

would still cause the fTest() function to be run. However, the function's returned value would be inaccessible to the calling code context.

Using the return operator in a function

Thus, when **return** is inserted in a function, the code/expression to the end of the line (or first semi-colon) is evaluated and returned. Note that as the **return** statement is closed by a semicolon, it is essentially a single line of code/expression started by the word 'return' (case-sensitive), a space, and then following action code code. For example:

- `return "Process complete";` returns a single literal string
- `return true;` returns a boolean value
- `return $MyNumber;` returns an attribute's value
- `return vMyValue;` returns a variable's value

However, if the to-be-returned value is not coded a single term, but rather as an expression whose output is returned, it *may be* enclosed in parentheses but does not require to beo The use of parentheses helps indicate, if only to the user, that some evaluation is required *before* the single result is used as the **return** value

The first example shows an evaluated result without parentheses, the other with parentheses:

```
return 2 + 2;
return ("Hello " + vName");
return (iPrice * $Tax);
```

Can return occur anywhere in the function?

In theory Yes, but in practice, No. This is because the function process completes after the return value is evaluated and passed back out. So extra code may occur after the return but *is not evaluated*.

One possible use for content in a function after a **return** statement is to add comments about a function placed at the end, after code so available for user reference but not pushing the actual code down the (viewable) screen so aiding usability.

Can there be a multi-value return?

No, a single value is returned but that value may be of a multi-item data type. Thus, if it is desired to pass multiple discrete values in the same **return** event, make the returned object a list or dictionary (or a variable of List, Set or Dictionary type).

Does there have to be a return statement?

No, it is not a requirement. For instance a logging function might take an input string and append it to the end of a specific note. In such a circumstance there is nothing for the function to return to the calling code.

Can there be more than one return statement in a function?

Yes. The function ceases running when a return statement is encountered but if the function has conditional branching code, each branch that represents a final result can have its own **return** statement. In such a case, the intent is to allow *different* returned value depending on the result if conditional tests.

So, whilst multiple **return** statements may exist overall, the code evaluates so only one **return** is ever processed (as the function stops at that point). For instance, for those unused to programming, initially this may be more easily understood using a single **return** statement to pass back one of 3 possible values:

```
function fTestLength(iString){
 var:string vOutput = "";
 if(iString){
  if(iString.size<=5){
   vOutput = "Small string";
  }else{
   vOutput = "Large string";
  }
 }
 if(vOutput=="")
  vOutput = "No string";
 }
 return vOutput
}
```

However, the above example can also be written as:

```
function fTestLength(iString){
 if(iString){
  if(iString.size<=5){
   return "Small string";
  }else{
   return "Large string";
 }
 return "No string";
}
```

In this case, there are three discrete possible **return** values but each closes and returns for a different *single* branch of the code. As in the first example, the logic could as easily set a single variable in each branch and then use a single **ret** at the end of the function.

As the function quits once a **return** is reached, using multiple **return** statements can avoid the function having to evaluate unnecessary extra code. Such efficiency measure are unlikely to affect most Tinderbox use but can act as way to help understand use of functions with conditional outcomes. *Both* examples evaluate to a similar outcome; use whichever form makes the most sense.

If using multiple **return** statements, take care to avoid exiting the function too early, i.e. that each **return** statement is truly a final point in the function's logic.

Next: Calling functions

Calling functions

Important note: all examples follow aTbRef naming conventions.

Calling functions from action code

Regardless of where stored, *every* function in the current document can be called from *any* action.

If a function has arguments, the caller must supply values for each argument, even if only as an empty string (or other default data-type value. Extra arguments are ignored. If a function is redefined, the most recent definition replaces any existing definitions.

If the called function returns a value, then the calling code needs to use an attribute or variable to receive that value.

Examples—with a return value

Calling a single literal value for an argument:

```
$MyNumber = fAddTax(500);
```

Calling a function with a single argument:

```
$Text = fMakeTable($CategoryList);
```

Calling a function with multiple arguments, that returns List-type data (see here for multi-value returns):

```
$MyList = fAlterList($CategoryList,$SomeString,"alternate");
```

A function with no arguments:

```
$MyNumber = fCheckSum();
```

Notice how in every case the function is called as the right side of an '=' (i.e. value assignment) operation so the return value can be stored and further used.

Example—with no return value

A function with no return value:

```
fReset();
```

Note in this case that whilst no left-side recipient is needed but function parentheses must be included.

Can a function call another function?

Yes. A function can call itself. This is called 'self-reference', and is an advanced technique that should be used with caution and tested in a test document before use at scale. This process is also known as recursion.

Next: Variable use in functions

Variable use in functions

Important note: all examples follow aTbRef naming conventions.

Uses of variables in functions

There are three ways that a variable are used in functions:

- as a local variable, just like any other use of variables in Action code;
- as an "argument" variables used to receive the data passed from the calling code;
- as "loop" variables that will be local to the loop code, just like any other use of loop variables.

Defining a variable in a function

A function variable is defined exactly the same way as in other action code:

```
var vThing;
```

As the basic assumed data type, if not set, is a String, it can be more helpful to use explicit typing, whether or not a value is applied straightway. The point is that the variable has a notified data type from the outset, rather than one coerced by Tinderbox based on context of use:

```
var:string vThing;
var:number vCount;
var:date vNow = date("today");
var:boolean vBoolean = true;
```

A variable needs to be defined *before* first use, as with attributes. It makes good sense to define variables at the start of the function, noting that loop variables are defined in the loop operator. Do consider also adding comments about the variable's purpose, as once written the code may only be re-read occasionally and the code's intent may get forgotten over time.)

Using variables to set data type of inputs

From v9.5.0, the data type of input arguments can be specified making the usage below obsolete. See Function arguments.

Legacy. Previously, as from v9.1.0, the function's arguments when accessed from within the function supply String-type data (this is regardless of the data type supplied in the action calling the function). Thus, to avoid unexpected type coercion, e.g. Date being misread as a String, it can be useful to pass an argument into a typed variable *before* using the argument value in the functions action code:

```
function fMakeTable(iSomeList){
 var:list vList = iSomeList;
 vList.each(anItem){
  // ..tc.
}
```

Variable duration

Variables created in a function are destroyed when the function execution completes. If a variables value is needed, other than as a **return** value, the value will need to be passed to an attribute for more persistent storage.

Variables only exist while the function is being executed after having been called. Whilst variables are most often used to temporarily keep intermediate results while the function is executing, if a variable holds some value that is needed elsewhere, it should either be passed back to the calling code via the return operator or an attribute should be assigned to the variable's value. Some examples of assigning attributes to function variables are:

```
// ...
$MyString = vString;
$SomeNumber ("config-note") = vCount;
// ...
```

In the first case, $MyString in the calling note is set. In the latter case, $SomeNumber in note "config-note" is set. The choice of where a stored value is saved is dependent on the wider context of use, i.e. there is no single 'correct' choice.

A variable may be used as an input argument for a call to another function:

```
// ...
var:string vString = "Testing!";
fAnotherFunction(vString){
 // ...
```

Variable scope

A variable created in a function is only accessible from within that same function. For experienced programmers this may seem unexpected (see more) and may change in the future. At present, a variable should only be used within the function in which it is defined.

Next: Attributes vs. variables in functions

Attributes vs. variables in functions

Important note: all examples follow aTbRef naming conventions.

In brief pre-summary, attributes are heavy-weight: persistent, accessible through the document, must be cleaned up if used for storing intermediate states during an action. Variables are light-weight: they can be defined as needed, they are destroyed (cleaned-up once the function has run) but only accessible within the current function.

Attributes

In a function, any attribute reference, e.g. `$MyString`, refers to an attribute in the calling note (i.e. the current item in a selection). It is still possible to refer to the same attribute in another note via an offset address, e.g. `$MyString("SomeNote")`. Or, designators may be used `$MyString(parent)` gets the MyString value of the current item; in an alias, it is the alias' parent, or `$MyString(parent(original))` will get the value of the same attribute bu from the *parent* of the alias.

If the function needs to store a lot of interim values, as in some string manipulation tasks, this can involve using quite a few attributes. It is no problem for Tinderbox to create additional user attributes, even if only for this task, buy if many are needed do consider variables as an alternative. Note that these attributes need to be created *before* the function is used. Attributes can be created via the Add Displayed Attributes configuration panel (attribute name and data type only), or th Document Inspector's User tab. Only the Inspector allows configuration of all aspects of a user attribute.

If using attributes consider:

- Attributes have global scope vs. local scope for variables. This means that once saved in an attribute, a value is available from any action in the document (albeit you may need to use an offset address, where the code references $AttributeName(name or path of target note)). So:
  - Attributes have permanence—they are integral to the document, and saved with it.
- Functions that reference attributes can interact with multiple notes located anywhere within the current document (via offset addressing).
- Attributes must be defined outside of the context of Action code and they must already exist in a document before they can be used.
- Using attributes causes the function to be dependent on the document and its structure, whereas variables are self-contained within the function.

To minimise the build up of forgotten data consider:

- clear attributes, once they have used, with a reset at the end of the function (though before the return, if used).
- use attributes within a single note, or perhaps a note per function if complex, so as to compartmentalise *where* stored data is generated.

Alternatively, consider action code variables, as described below

(Action code) Variables

Using action code variables within functions can help their use without needing to use any attributes, other than explicitly storing output results. Variables are created when defined and cease to exist (i.e. including the data they store) when th action containing them, such as a function, has finished being run. So if a function declares a variable and stores data in it, that value is only accessible within the function _and_ once the function is run the values are lost forever.

Such use makes a function more self-contained, especially if planning to use the same function (or library of functions) in different Tinderbox documents. This is because the function creates/destroys variables as needed and there is no need for user attributes (see above) which might not (yet) have been created in the current document. Using variables avoids that extra set-up of configuring attributes.

Variables created within a function only work within the scope of that function, as discussed here.

Which approach is better?

…is the wrong question to ask! It is perfectly fine to use either, or a mix of both. Use whichever appeals and makes sense for the user's style of work. The choice will likely depend on a mix of personal style and the nature of use of the functions.

Next: Additional examples

Additional examples

Important note: all examples follow aTbRef naming conventions.

If you are starting here for cut-'n-paste samples, do take time to read this whole section. If new to action code functions, it will help to have a basic understanding of what functions do.

Basic example

A simple function might be written to return a number with an 18% tax uplift applied to the input number:

```
    function fAddTax(iPrice){return(1.18*iPrice);};
```

or, split out only lines for clarity:

```
function fAddTax(iPrice){
 return(1.18*iPrice);
};
```

Arguments arrive as String type data and Tinderbox coerces them as best it can. So here iPrice is a string value containing a number. However Tinderbox knows that if a number, e.g. 1.18 is multiplied by a String (containing a number) the String-type should be coerced to a Number-type before use and the resulting value will be Number data type.

Though more verbose, a safer approach—especially if unclear as to Tinderbox's coercion logic—is to be more specific, and insert comments, as shown below :

```
function fAddTax(iPrice){
 // Arguments: iPrice - a number, the starting price
 //
 // make a variable (expected type of Number) and set it to argument #1
 var:number vNum = iPrice;
 // multiply variable by tax rate
 vNum = 1.18 * vNum;
 //return the tax-adjusted price
 return vNum;
};
```

This defines a new function named 'fAddTax' that can be used in any action or expression (including ^action()^ in templates) to apply 18% tax to to the input figure, i.e. $1.00 → $1.18. It has one input argument, and returns a single value. Thus:

```
    $MyNumber = fAddTax(500);
```

would set MyNumber's value to 1.18*500, i.e. 590. If we assume this is money, a $500.00 input becomes $590.00 including tax (be it US Dollars or any currency).

But hard-coding (i.e. 1.18) values like tax rates is generally considered bad practice as values can (will!) change. A better function might be this, comments, line breaks for for clarity:

```
function fAddTax(iPrice){
 // Arguments: iPrice - a number, the starting price
```

```
 //
 // var:number vTaxRate = 1.18;
 // make a variable (expected type of Number) and set it to argument #1
 var:number vNum = iPrice;
 // multiply variable by tax rate
 vNum = vTaxRate * vNum;
 //return the tax-adjusted price
 return vNum;
};
```

But, supposedly fixed values like tax rates can change when governments change. So…

Adding an additional function argument

What if the tax rate changes? This can be accommodated by adding a second argument for the tax rate. Extra comments, line breaks for for clarity:

```
function fAddVarTax(iPrice,iTaxRate){
 // Arguments:
 // iPrice - a number, the starting price
 //  iTaxRate - tax rate percentage as a number (15 not 0.15)
 // --------------

 // make a variable (expected type of Number) and set it to argument #1
 var:number vNum = iPrice;
 // multiply variable by tax rate, assuming the tax rate input is a decimal percent, e.g. '0.18' for 18%
 var:number vTax = iTaxRate;
 vNum = (1 + vTax) * vNum;
 //return the tax-adjusted price
 return vNum;
};
```

Thus:

```
    $MyNumber = fAddVarTax(500,20);
```

would pass a 20% tax rate, set MyNumber's value to 1.20*500, i.e. 600.

Further examples:

```
function fAnswer() {
 return 42;
}
```

always returns 42. Or:

```
function fFibonacci(iNum){
 if(iNum<2){
  return (iNum);
 };
 return fFibonacci(iNum-1)+fFibonacci(iNum-2);
}
```

returns the **iNum**th Fibonacci number. This is a more advanced example showing the concept of recursion, whereby a function calls itself.

Using more concise code

For the experts who understand the flow of the code, the function defined above could equally well be defined in a terser form:

```
function fVarTax2(N,T){
 return ((1+(T/100))*N);
};
```

Those with coding expertise my prefer this form. However, aTbRef code examples here are written assuming a reader with little or no coding expertise, thus the use of a more explicit and verbose format.

Next: Comments in functions

Comments in functions

Important note: all examples follow aTbRef naming conventions.

As Mark Bernstein's book *The Tinderbox Way* notes: "...the primary audience of a note is almost always our future self." (pg. 33). To wit, comments are notes to our future debugging self.

Action code allows commenting and adding comments to a function is a good idea. The scope and depth of notes is a matter of personal taste, but if making a library of functions for other people, it is a good idea to explain the purpose of the function and the data types of argument and return values.

For example:

```
function fAddTax(iPrice){
 // Arguments: iPrice - a number, the starting price
 //
 // make a variable (expected type of Number) and set it to argument #1
 var:number vNum = iPrice;
 // multiply variable by tax rate
 vNum = 1.18 * vNum;
 //return the tax-adjusted price
 return vNum;
};
```

To make it easier to see the function code without scrolling, one approach is to put a comment at the start of the function indicating the main detail is at the bottom of the function.

For example, re-using the example above:

```
function fAddTax(iPrice){
 // Comments at end

 var:number vNum = iPrice;
 vNum = 1.18 * vNum;
 return vNum;

 // Arguments: iPrice - a number, the starting price
 // make a variable of 'number' type and set it to argument #1 to enforce data type
 // multiply variable by tax rate
 // return the tax-adjusted price
};
```

This latter form yields little benefit in a small function like above, but as functions grown in length and number of arguments or return(s) values, coding becomes a big help to future self.

The need for comments really reflects two factors:

- Sharing. If this function will be shared with others, will your logic be self-evident to another user?
- Complexity. Will you—or another user—remember later on the original intent or logic in a large function? What was clear when writing the function may not be so months later if there is next a need to look at the code, e.g. to enhance o correct it, or simply understand how it functions!

Next: Nesting functions

Nesting functions

Important note: all examples follow aTbRef naming conventions.

NOTE: this subject is only pertinent for more expert users

Can a function code contain another complete function?

In principle, Yes, this is possible. Those with a programming background may be used to techniques like nesting one function inside another. However, there is no difference in use by doing so and it is currently **not recommended**. Why:

- Tinderbox may harden scoping rules in the future, which would break existing code.
- It is confusing, and likely to give a headache when trying, a month from now, to figure out what the code is doing.

Variables in the outer function are (likely) accessible to the nested function but again, this is currently *not recommended*.

Next: Re-using variables from other functions

Re-using variables from other functions

Important note: all examples follow aTbRef naming conventions.

NOTE: this subject is only pertinent for more expert users

Those with a programming background my be used to techniques like re-using variables.

Can a function use variables from another function?

In theory, yes, as the example below shows. However, this is not recommended. Consider this code

```
function fTest(iVal) {
 var:number vFactor=3;
 return fDoIt(iVal);
};

function fDoIt(iVal2) {
 return vFactor*iVal2;
};

$MyNumber=fTest(7); // returns 21
```

As stated above, this usage—whilst possible—is **not recommended**.

If a variable needs to be shared in a wider scope, using a (user) attribute—either in the calling note i=or a specified note—is a perfectly practical alternative.

Operators

More complex actions than simple value assignments are effected via action 'operators'. There are 311 built-in operators of varying complexity and purpose as described in the various listings below:

- Full Operator List
- Action Operator Types
- Action Operator Scope
- Action Operator Functional Types
- Listing of dot-operators
- Listing of non dot-operators with dot-operator versions
- Listing of operators with link type filters
- Listing of operators emitting styled text
- Listing of operators for Stream Parsing
- Listing of operators that can use regular expressions
- Listing of operators with scoping arguments
- Listing of operators with optional arguments
- Listing of operators with conditional arguments
- Listing of operators with loop variables

Full Operator List

NOTE: action code operator names are camelCase *case-sensitive*, e.g. firstWord() not Firstword() or firstword(). A few deprecated action codes/usages are listed separately here. Optional operator arguments are shown in [square] brackets.

Note: meaning of square brackets in code syntax.

A list of all current action operator codes:

- - (i.e. subtraction)
- -= (i.e. decrement)
- != (i.e. value inequality)
- ... (i.e. range)
- (!$AttributeName) (i.e. a short form test for no value)
- * (i.e. multiplication)
- / (i.e. division)
- & (i.e. query logical AND join)
- &= (i.e. logical AND assignment)
- %matches (query back-references)
- + (i.e. addition)
- + (i.e. string concatenation)
- += (i.e. increment)
- < (i.e. less than)
- <= (i.e. less than or equal to)
- = (i.e. value assignment)
- == (i.e. value equality)
- > (i.e. greater than)
- >= (i.e. greater than or equal to)
- | (i.e. query logical OR join)
- |= (i.e. logical OR assignment)
- $AttributeName (i.e. a short form test for value)
- $AttributeName[(scope)]
- $N (query back-reference)
- abs(sourceNum)
- action([scope,]codeStr)
- any(scope, condition)
- atan(radiansNum)
- attribute(attributeNameStr).keys
- attribute(attributeNameStr)[keyStr]
- attributeEncode(dataStr)
- avg_if(scope, condition, expressionStr)
- avg(scope, expressionStr)
- between(valueNum, minNum, maxNum)
- capitalize(dataStr)
- ceil(sourceNum)
- changed([scope])
- collect_if(scope, condition, expressionStr)
- collect(scope, expressionStr)
- Color.blue()
- Color.brightness()
- Color.format()
- Color.green()
- Color.hue()
- Color.red()
- Color.saturation()
- compositeFor(nameStr)
- compositeFor(nameStr):count
- compositeFor(nameStr):kind
- compositeFor(nameStr):name
- compositeFor(nameStr):role(roleStr)

- compositeFor(nameStr):roles
- compositeWithName(compositeNameStr)
- contains(item)
- cos(radiansNum)
- count_if(scope, condition)
- count(scope)
- covid([stateStr, countryStr|zipCodeStr], aDate, keywordStr)
- create([containerStr,]nameStr)
- createAdornment([containerStr,] nameStr)
- createAgent([containerStr,]nameStr)
- createAttribute(nameStr[, dataType])
- createLink(sourceItem, destinationItem[, linkTypeStr])
- Date.day()
- Date.format(formatStr)
- Date.hour()
- Date.minute()
- Date.month()
- Date.second()
- Date.week()
- Date.weekday()
- Date.year()
- date(dateStr)
- date(yearNum, monthNum, dayNum[, hourNum, minNum])
- day(aDate[, dayNum])
- days(firstDate, lastDate)
- degrees(radiansNum)
- delete(scope)
- descendedFrom(item)
- Dictionary.add(itemDict)
- Dictionary.contains(keyStr)
- Dictionary.count()
- Dictionary.empty()
- Dictionary.extend(itemDict)
- Dictionary.icontains(keyStr)
- Dictionary.keys()
- Dictionary.size()
- dictionary(dictionaryStr)
- Dictionary[keyStr]
- distance(startItem, endItem)
- distanceTo(startItem, endItem)
- do(macroStr[,argumentsList])
- document.keys()
- document()
- document[keyStr]
- drivingTimeTo(item)
- eachLink(loopVar[,scope]){actions}
- escapeHTML(dataStr)
- eval([item], expressionStr)
- every(scope, condition)
- exp(powerNum)
- exportedString(item[, templateStr])
- fail()
- fetch(urlStr,headersDict,attrNameStr,cmdStr[,httpMethod])
- find(scope)
- first(item[, childrenNum])
- firstWord(dataStr)
- floor(sourceNum)
- format(dataStr, formatStr[, additionalArguments])
- function
- hasLocalValue(attributeNameStr[, item])
- hour(aDate[, hoursNum])
- hours(startDate, endDate)
- idEncode(dataStr)
- if(condition){actions}[else{actions}]
- indented(depthNum[, item])
- inheritsFrom([item,]prototypeStr)
- inside(item)
- Interval.day()
- Interval.format(formatStr)
- Interval.hour()
- Interval.minute()
- Interval.second()
- interval(dataStr)
- interval(startDate, endDate)
- isbn10(dataStr)
- isbn13(dataStr)
- isDuplicateName(item)
- JSON.each([pathStr]){actions}
- JSON.json[itemNum]
- JSON.json[keyStr]
- JSON.jsonValue(pathStr)
- jsonEncode(dataStr)
- last(item[, childrenNum])
- lastWord(dataStr)
- linkedFrom(scope[, linkTypeStr])
- linkedTo(scope[, linkTypeStr])
- linkFrom(scope[, linkTypeStr])
- linkFromOriginal(scope[, linkTypeStr])
- linkPath(pathNameStr[, startStr, endStr])
- links[(scope)].[directionStr].[linkTypeRegex].attributeNameRefStr
- linkTo(scope[, linkTypeStr])
- linkToOriginal(scope[, linkTypeStr])
- List.isort([attributeRefStr])
- List.nsort([attributeRefStr])
- List.select()

- List.sort([attributeRefStr])
- List.unique()
- list(expressionList)
- List/Set.any(loopVar, expressionStr)
- List/Set.asString()
- List/Set.at(itemNum)
- List/Set.avg()
- List/Set.collect_if(loopVar, condition, expressionStr)
- List/Set.collect(loopVar, expressionStr)
- List/Set.contains(matchStr)
- List/Set.count_if(loopVar, condition)
- List/Set.count()
- List/Set.countOccurrencesOf(literalStr)
- List/Set.each(loopVar){actions}
- List/Set.empty()
- List/Set.every(loopVar, expressionStr)
- List/Set.first()
- List/Set.format(formatStr)
- List/Set.icontains(matchStr)
- List/Set.intersect(aSet)
- List/Set.last()
- List/Set.lookup(keyStr)
- List/Set.max()
- List/Set.min()
- List/Set.randomItem()
- List/Set.remove(matchValue)
- List/Set.replace(regexMatchStr, replacementStr)
- List/Set.reverse()
- List/Set.size()
- List/Set.sum_if(loopVar, condition[, expressionStr])
- List/Set.sum()
- List/Set.tr(inStr, outStr)
- List/Set[itemNum]
- locale([localeCodeStr])
- log(sourceNum)
- lowercase(dataStr)
- max(numberList)
- min(numberList)
- minute(aDate[, minutesNum])
- minutes(startDate, endDate)
- mod(sourceNum, modulusNum)
- month(aDate[, monthsNum])
- months(startDate, endDate)
- neighbors(scope, distanceNum[, linkTypeStr])
- neighbors2(scope, distanceNum[, linkTypeStr])
- neighbors2Within(scope, distanceNum[, linkTypeStr])
- neighborsWithin(scope, distanceNum[, linkTypeStr])
- notify(headlineStr[, detailsStr, deliveryDateTime])
- Number.ceil()
- Number.floor()
- Number.format(decimalsNum[, widthNum, padStr]|formatStr)
- Number.precision(decimalsNum)
- Number.round()
- originalLinkedFrom(scope[, linkTypeStr])
- originalLinkedTo(scope[, linkTypeStr])
- play(soundNameStr)
- pow(sourceNum, powerNum)
- radians(degreesNum)
- rand([maxNumber])
- require(featureName)
- return
- rgb(redNum, greenNum, blueNum)
- round(sourceNum)
- runCommand(commandStr[, inputsStr, dirStr])
- seconds(startDate, endDate)
- select()
- select(scope)
- show(msgString[, backgroundColor[,colorString]])
- similarTo(item[, notesNum])
- sin(sourceNum)
- sqrt(sourceNum)
- stamp([scope,]stampName)
- String.beginsWith(matchStr)
- String.capitalize()
- String.captureJSON()
- String.captureLine([targetAttributeStr])
- String.captureNumber([targetAttributeStr])
- String.captureRest([targetAttributeStr])
- String.captureTo(matchStr[, targetAttributeStr])
- String.captureToken([targetAttributeStr])
- String.captureWord([targetAttributeStr])
- String.captureXML()
- String.contains(regexStr)
- String.containsAnyOf(regexList)
- String.countOccurrencesOf(literalStr)
- String.deleteCharacters(characterSet)
- String.eachLine(loopVar[:condition]){actions}
- String.empty()
- String.endsWith(matchStr)
- String.expect(matchStr)
- String.expectNumber()
- String.expectWhitespace()
- String.expectWord()
- String.extract(regexStr[, caseInsensitiveBln])
- String.extractAll(regexStr[, caseInsensitiveBln])

- String.failed()
- String.find(matchStr)
- String.following(matchStr)
- String.highlights([aColor])
- String.icontains(regexStr)
- String.icontainsAnyOf(regexList)
- String.json()
- String.jsonEncode()
- String.lowercase()
- String.next()
- String.nounList()
- String.paragraph(paraNum)
- String.paragraphCount()
- String.paragraphList()
- String.paragraphs(parasNum)
- String.replace(regexMatchStr, replacementStr)
- String.reverse()
- String.sentence([sentenceNum])
- String.show([backgroundColor[,colorString]])
- String.size()
- String.skip(charsNum)
- String.skipLine()
- String.skipTo(matchStr)
- String.skipToNumber()
- String.skipWhitespace()
- String.speak([voiceNameStr])
- String.split(regexStr)
- String.substr(startNum[, lengthNum])
- String.toNumber()
- String.tr(inStr[, outStr])
- String.trim([filterStr])
- String.try{actions}[.thenTry{actions}]
- String.uppercase()
- String.wordCount()
- String.wordList()
- String.words(wordsNum)
- StyledString.bold()
- StyledString.fontSize(pointSizeNum)
- StyledString.italic()
- StyledString.plain()
- StyledString.strike()
- StyledString.textColor(aColor)
- substr(dataStr, startNum[, lengthNum])
- sum_if(scope, condition, expressionStr)
- sum(scope, expressionStr)
- tan(radiansNum)
- time(aDate, hoursNum, minutesNum, secondsNum)
- time(aDate)
- twitter(usernameStr, statusStr)
- type(attributeNameStr)
- unlinkFrom(scope[, linkTypeStr])
- unlinkFromOriginal(scope[, linkTypeStr])
- unlinkTo(scope[, linkTypeStr])
- unlinkToOriginal(scope[, linkTypeStr])
- update(scope)
- uppercase(dataStr)
- urlEncode(dataStr)
- values([scope,]attributeNameStr)
- var
- version()
- weeks(startDate, endDate)
- while(condition){}
- word(dataStr)
- wordsRelatedTo(dataStr[, wordsNum])
- XML.each(pathStr){action}
- XML.xml(pathStr)
- year(aDate[, yearsNum])
- years(startDate, endDate)

- (i.e. subtraction)

Operator Type: Operator [other Operator type actions]
Operator Scope of Action: Item [operators of similar scope]
Operator Purpose: Mathematical [other Mathematical operators]
Operator First Added: Baseline
Operator Last Altered: As at baseline

- (i.e. subtraction)

The subtraction operator, - (minus sign character), returns the remainder of the argument before it when the argument after it is subtracted.

```
$MyNumber = 3-4;
```

When mixing data types (a '-' might be intended as a hyphen) or if working with lists—i.e. List & Set types—see notes under concatenation.

-= (i.e. decrement)

Operator Type: Operator [other Operator type actions]
Operator Scope of Action: Item [operators of similar scope]
Operator Purpose: Mathematical [other Mathematical operators]
Operator First Added: Baseline
Operator Last Altered: As at baseline

-= (i.e. decrement)

This assignment operator is a convenient shorthand for decrementing an attribute. For example, the two following statements are equivalent:

```
$MyNumber -= 3;
```

```
$MyNumber = $MyNumber - 3;
```

This operator may also be used with Lists and Sets.

!= (i.e. value inequality)

Operator Type: Operator [other Operator type actions]
Operator Scope of Action: Query [operators of similar scope]
Operator Purpose: Query Boolean [other Query Boolean operators]
Operator First Added: Baseline
Operator Last Altered: As at baseline

!= (i.e. inequality test)

The operator to test inequality (i.e. 'is **not** the same as') is '!=', an exclamation mark followed by an equals sign.

This operator is used either in agent queries or in the conditional part of an if(condition){action} code. It is the functional opposite of the '==' equality test.

This test cannot be meaningfully applied to Set or List type attribute data, as the entire attribute value is matched, rather than individual values as might otherwise be assumed. For these data types use the .contains() or .icontains() operator instead, noting the scope for ambiguous matching due to stemming of words ("car" with match "car", "cars" and "carrot").

Because equality comparisons of Date-type data match at day scope, rather than full date/time values (for legacy reasons), use interval() to compare Date-type inequality.

Equality testing can be negated, i.e. nested for a non-match, or combined with greater/less than for a range of tests as further explored in Basic Comparison Codes.

For a case-insensitive lexical equality test, use a lowercase on-the-fly transform:

```
"Absquatulate".lowercase != "Absquatulate"
```

If we set $MyString to "Absquatulate", then:

```
$MyString.lowercase != "Absquatulate"
```

If $MyOtherString(Some note) has the value "Absquatulate", then:

```
$MyString.lowercase != $MyOtherString(Some note)
```

Note the stored left-side value is not altered, but its transformed version is used in the test giving a case-insensitive comparison. This method only works for upper/lower case comparisons; accented characters are lexically different characters regardless of case.

... (i.e. range)

Operator Type: Operator [other Operator type actions]
Operator Scope of Action: Item [operators of similar scope]
Operator Purpose: Mathematical [other Mathematical operators]
Operator First Added: Baseline
Operator Last Altered: As at baseline

... (i.e. range)

The range operator constructs a list of numbers from a specified starting point to a specified end point. Note: the range operator is written as 3 dots and *not* an ellipsis character. White space immediately before or after the operator is ignored as with other mathematical operators. For example:

```
1...3 → "1;2;3"
3 … 1 → "3;2;1"
```

Note do **not** use enclosing quotes or square brackets with the range operator as the operator not not be correctly evaluated, resulting in the wrong outcome.

```
[1...3] → "1 ... 3" WRONG!
"1...3" → "1 ... 3" WRONG!
```

The range operator binds more tightly than arithmetic operators. Thus

```
1...3 * 2 → "2;4;6"
```

and is the is the same as

```
(1..3) * 2 → "2;4;6"
```

i.e. the parentheses in the second example are not needed.

The range operator can be useful for performing a task a specific number of times using the List.each() operator:

```
1...10.each(x){var vPath="/container/item "+x; create(vPath);}
```

and in that example the value of the loop variable `x` is the value for the source list generated by the range operator, i.e. it is the same as would occur in this literal example:

```
"1;2;3;4;5;6;7;8;9;10".each(x){var vPath="/container/item "+x; create(vPath);}
```

In both cases, on the second iteration of the `.each()` loop, the loop variable `x` would have the value `2`.

Using non-literal range specifiers

It may be useful to define one or both range limits. For example, if $MyNumber is 3:

```
$MyList = [1 ... $MyNumber]; gives "1;2;3"
```

or if variable vNum is 4:

```
$MyList = 1 ..vNum; gives "1;2;3;5"
```

But for more complex expressions using chained operators, parentheses my be needed to hint intent to Tinderbox. Thus, if $MySet has 4 items:

```
$MyList = 1 ... $MySet.size; gives "1" WRONG
```

so add parentheses around the expression:

```
$MyList = 1 ... ($MySet.size); gives "1;2;3;4" CORRECT
```

If chaining operators to range specified using expressions, consider using parentheses around the whole range definition:

```
(1 ... ($MySet.size)).each(x){$Text+= x+"\n";};
```

The last example actually works without the parentheses but illustrates the concept.

Using range to supply a loop counter

When working with List.each(), it can be useful to know which source list item is being currently in scope. It is possible to use a variable (see here), but the range operator offers another method. In this method a range-generated list is iterated and in-loop the list being processed is called via List.at(). It in important to note that as .at() numbers from zero, either the range must start from zero, or the range item used with at must be adjusted by -1.

In the following example, $MyList holds the data of interest, and a zero-based range with be generated:

```
(0 ... ($MyList.count)).each(N){
 $MyList.at(N) ..tc.
}
```

Thus for loop #3, the value of N will be `2` (recall the range code is making a zero based list of numbers). Thus in-loop, $MyList.at(N) will be the same as $MyList.at(2), i.e. addressing the the *third* item in $MyList.

(!$AttributeName) (i.e. a short form test for no value)

Operator Type: Operator [other Operator type actions]
Operator Scope of Action: Item [operators of similar scope]
Operator Purpose: Query Boolean [other Query Boolean operators]
Operator First Added: Baseline
Operator Last Altered: As at baseline

(!$AttributeName)

In queries and conditional action code expressions, preceding a '$'-prefixed attribute name with an exclamation mark functions as a shortened form of the Boolean test for a *false* value. Due to limitations in the underlying parser, an inequality test expression **must** be enclosed in parentheses.Thus the following are functional equivalents:

```
$MyBoolean==false
(!$MyBoolean)
```

Do not use this form or you may get unexpected results:

```
!$MyBoolean
```
Do not use this format, without parentheses !

In both cases the result is *false* if the value of $MyBoolean is *false*.

Usefully Tinderbox also useful supports such short-form !$AttributeName tests for all the other attribute data types, returning true if the attribute has a *default* value. Per data type, this equates to long form tests like:

```
$MyBoolean==false
$MyColor==""
```

`$MyDate=="never"`
`$MyDictionary==""`
`$MyFile==""`
`$MyInterval=="00:00"`
`$MyList==[]` (or (deprecated) `$MyList==""`)
`$MyNumber==0`
`$MySet==[]` (or (deprecated) `$MyList==""`)
`$MyString==""`
`$MyURL==""`

This also holds for system-only data types:

`$MyAction==""`
`$MyFont==""`

In all these cases a short-form test returns *true* if the attribute value *is* the default for that data type.

This test is the logical opposite of the $AttributeName test.

* (i.e. multiplication)

Operator Type: Operator [other Operator type actions]
Operator Scope of Action: Item [operators of similar scope]
Operator Purpose: Mathematical [other Mathematical operators]
Operator First Added: Baseline
Operator Last Altered: As at baseline

- **(i.e. multiplication)**

The multiplication operator, * (asterisk character), returns the multiplication of the arguments before and after it.

```
$MyNumber = 3*4;
```

Both arguments are normally Number-type data.

Multiplying Strings

Sometimes there is a need to expand or repeat a String. For this, the multiplication operator * may be used. Due to Tinderbox's internal type coercion effects, the number argument **must** be used as the *first* argument; placing it after the * wi cause a silent fail with no resulting output. Thus, the expression:

```
$MyString = 3*"xyz ";
```

evaluates to a string "xyz xyz xyz". But this:

~~`$MyString = "xyz "*3;`~~ WRONG!

will return nothing.

Multiplying Lists and Sets

Lists and Sets of numbers may be multiplied by a number. Two lists of numbers may be multiplied if they have the same length, in which case their elements are multiplied.

`[1;2;3;4] * [1;2;3;4];` ➛ `[1;4;9;12]`

or, using the range operator:

`1...4 * 1...4` ➛ `1;4;9;12`

/ (i.e. division)

Operator Type: Operator [other Operator type actions]
Operator Scope of Action: Item [operators of similar scope]
Operator Purpose: Mathematical [other Mathematical operators]
Operator First Added: Baseline
Operator Last Altered: As at baseline

/ (i.e. division)

The division operator, / (forward slash character), returns the argument before it divided by the argument after it.

```
$MyNumber = 3/4;
```

By default, up to 5 decimal places are returned, so:

```
$MyNumber = 10/3;
```

results in $MyNumber having a value of 3.33333.

Be aware the forward slash character has many other symbolic meanings (POSIX folder delimiter, linguistic use e.g. 'and/or', date component delimiters, etc.). Be careful to ensure to quote or escape the / character where you do not want Tinderbox to assume a division.

& (i.e. query logical AND join)

Operator Type: Operator [other Operator type actions]
Operator Scope of Action: Query [operators of similar scope]
Operator Purpose: Query Boolean [other Query Boolean operators]
Operator First Added: Baseline
Operator Last Altered: As at baseline

& (i.e. query logical AND join)

The ampersand, &, is used in queries and conditional statements as a logical AND join. Thus:

```
if ($HasStock == true & $Price > 20) {$Badge="ok";}
```

The note's $Badge is only set if **both** the first **and** the second argument are `true`.

See also the OR join and conditional statements using multiple arguments.

&= (i.e. logical AND assignment)

Operator Type: Operator [other Operator type actions]
Operator Scope of Action: Item [operators of similar scope]
Operator Purpose: Assignment [other Assignment operators]
Operator First Added: Baseline
Operator Last Altered: As at baseline

&= (i.e. logical AND assignment)

To make it easier to write rules succinctly, you can use the assignment:

```
$TheAttribute &= the_expression;
```

…which sets *$TheAttribute* to the value of *the_expression* only if it is currently evaluated as `true` **AND** *the_expression* is also evaluates as `true`. Thus, if *$TheAttribute* is `false`, the expression is still evaluated. An attribute that has no value set (or inherited) is evaluated as *false*.

For most attributes, especially new user attributes, the default value will evaluate as `false`. But, *$TheAttribute*'s data type does not have to be Boolean.

For new String-type attributes (and string-based Action/Color/File/Font/Interval/List/Set/URL types) the default is an empty string "". For Number-type, is it 0 (zero). For Date-type, it is the string "never". For Boolean-type, is it 'false' (with no quotes, and shown as un-ticked if displayed via a tick-box). The "", 0 and "never" values—for the appropriate data types—evaluate as `false`. But, be aware that not all system attributes follow this assumption. For instance, the $Color default is preset to use a named Tinderbox colour, so its default value (even if inherited) would evaluate as `true`.

Prototypes have no direct effect as the outcome, as it initially depends on the evaluation of the left-side attribute value regardless of whether document default, prototype inherited or locally set.

For example, for the Boolean-type attribute $Urgent:

```
$Urgent &= any(children,$Urgent);
```

Thus a project is considered urgent if it has been declared to be urgent itself and if any child is urgent.

See also the logical OR assignment.

Errors to the function logic were corrected.

%matches (query back-references)

Operator Type: Property [other Property type actions]
Operator Scope of Action: Query [operators of similar scope]
Operator Purpose: Data manipulation [other Data manipulation operators]
Operator First Added: 9.6.0
Operator Last Altered: As at baseline

Syntax note: *Operators without any defined mandatory arguments may omit their empty closing parentheses*

%matches

The expression %matches represents a list that contains all available back-references to regular expression matches. This is equivalent to the list of back reference values `[$0;$1;$2….$N]`, where N is the count of available regular expression matches.

If $MyString is "I do not like green eggs and ham.", then for this code:

```
if($MyString.contains("like (((green) (eggs)) and ham)")){
 $MyList = %matches;
};
```

$MyList now holds 5 back-references:

- $0: `like green eggs and ham`
- $1 `green eggs and ham`
- $2 `green eggs`
- $3 `green`
- $4 `eggs`

Query back-references are discussed in fuller detail here.

+ (i.e. addition)

Operator Type: Operator [other Operator type actions]
Operator Scope of Action: Item [operators of similar scope]
Operator Purpose: Mathematical [other Mathematical operators]
Operator First Added: Baseline
Operator Last Altered: As at baseline

+ (i.e. addition)

The addition operator, + (plus sign character), returns the sum of the arguments before and after it.

```
$MyNumber = 3+4;
```

The + sign is also used for string concatenation. Depending on context, Tinderbox will decide which operation is intended by the + sign. This is a very good reason to always enclose literal text strings in quotes.

+ (i.e. string concatenation)

Operator Type: Operator [other Operator type actions]
Operator Scope of Action: Item [operators of similar scope]
Operator Purpose: Mathematical [other Mathematical operators]
Operator First Added: Baseline
Operator Last Altered: As at baseline

+ (i.e. string concatenation)

The concatenation operator, + (plus sign character), joins the strings/references/expressions before and after it. Thus, the expression

```
$FullName = $FirstName+" "+$LastName;
```

evaluates a first name "Jane" and last name "Doe" to give a string value "Jane Doe" for the FullName attribute. If the right-side code in more complex, consider adding parentheses to help signal user intent to Tinderbox.

Mixing data types

If the right side inputs mix text and numbers—they need treating as numbers (even if stored in strings)—see more on concatenation versus addition.

The + (plus sign) character is also used for numerical addition. Depending on context, Tinderbox will decided which operation is intended by the + sign. This is a very good reason to always enclose literal text strings in quotes.

Concatenating Lists (List & Set type data)

Note that here both List-type and Set-type attributes can be considered the same as data sources, i.e. right-side inputs.

To concatenate lists, it might appear logical to do this:

```
$MyList = $SomeList+$SomeSet;
```
WRONG!

The result is no value is passed to $MyList. When adding lists to lists there are a variety of approaches. Note that depending on the nature of the task, and the method below that is used, it may first be necessary to reset the receiving list to the default (empty) value.

Use the '+=' increment operator:

```
$MyList += $SomeList+$SomeSet;
```

Add to the existing list (older older style—use the above method):

```
$MyList = $MyList + $SomeList+$SomeSet;
```

Or, if if intentionally replacing current $MyList values with new ones, this may be used:

```
$MyList = ($SomeList+$SomeSet);
```

The additional parentheses help Tinderbox to understand all the right-side lists need to be made into one list *before* being passed to the left-side.

+= (i.e. increment)

Operator Type: Operator [other Operator type actions]
Operator Scope of Action: Item [operators of similar scope]
Operator Purpose: Mathematical [other Mathematical operators]
Operator First Added: Baseline
Operator Last Altered: As at baseline

+= (i.e. increment)

This assignment operator is a convenient shorthand for incrementing an attribute. For example, the two following statements are equivalent:

```
$MyNumber += 3;
$MyNumber = $MyNumber + 3;
```

This operator may also be used with Lists and Sets, and += may be used to append strings:

```
$MyString += "!"
```

< (i.e. less than)

Operator Type: Operator [other Operator type actions]
Operator Scope of Action: Query [operators of similar scope]
Operator Purpose: Query Boolean [other Query Boolean operators]
Operator First Added: Baseline
Operator Last Altered: As at baseline

< (i.e. less than)

The operator used to test if the left side of an expression is less than the right side. This operator is used either in agent queries or in the conditional part of an if(condition){action} code. If $MyNumber is 3:

`$MyNumber < 4` is `true`
`$MyNumber < 2` is `false`

Further explored in Basic Comparison Codes.

<= (i.e. less than or equal to)

Operator Type: Operator [other Operator type actions]
Operator Scope of Action: Query [operators of similar scope]
Operator Purpose: Query Boolean [other Query Boolean operators]
Operator First Added: Baseline
Operator Last Altered: As at baseline

<= (i.e. less than or equal to)

The operator used to test if the left side of an expression is less than or equal to the right side. This operator is used either in agent queries or in the conditional part of an if(condition){action} code. If $MyNumber is 3:

`$MyNumber <= 4` is `true`
`$MyNumber <= 3` is `true`
`$MyNumber <= 2` is `false`

Further explored in Basic Comparison Codes; in some date comparison cases, a two term query may be needed instead.

= (i.e. value assignment)

Operator Type: Operator [other Operator type actions]
Operator Scope of Action: Item [operators of similar scope]
Operator Purpose: Assignment [other Assignment operators]
Operator First Added: Baseline
Operator Last Altered: As at baseline

= (i.e. value assignment)

A single = symbol is used **only** as a method of assignment (for tests of equivalence, use a double equals sign e.g. ==):

`$AttributeA = $AttributeB;` (attribute value
`$AttributeA = data;` (literal data)
`$AttributeA = 4 * ($Price / 2);` (expression)

The assignment is always from right to left; the left-side attribute takes the value of the right side attribute/expression. It is more usual for the right side to be evaluated (an expression) than the left, though the latter can occur.

Thus assignment sets the value of the specified left-side attribute the given right-side evaluated result; most often this is simply an attribute value. Where the right-side value is an attribute name, the $-prefix must be used. Take a note that has a $Color value of "red" and the $Rule:

```
$MyUserColor = "Color";
```

The result is not the text value *red* but rather it is a text value *"Color"*. Most likely there is no defined named colour named "Color" so some some other colour value, not the intended one, stored in $MyUserColor. If you want the value of an attribute, i.e. the $Color value of 'red' in this example, then your $Rule should be:

```
$MyUserColor = $Color;
```

You must use the $-prefix: see further detail below.

Actions and rules are allowed to specify a different referenced note, just as they can with a source attribute reference:

```
$AttribName(parent)="theValue";
$AttribName(/path/to/note)="theValue";
$AttribName(/path/to/note)=$MyValue(parent);
$AttribName($AnotherAttribute)="theValue";
```

In the last case the secondary attribute will hold a note name or path. Complex use of left-side expressions is allowed.

Do not mix $Attribute and Attribute(regex) syntax in a single call, i.e. $Attribute(regex), as this will cause expected results. *Use one syntax or the other*. This possible conflict should only ever occur in the context of queries or operations that allow query-style code.

If a value only needs to be assigned once, consider using a logical OR join.

Using Paths (offset addressing)

$AttributeA = $AttributeB(note/item/path)

This sets the value of the **$AttributeA** to that of the **$AttributeB** of the same note (i.e. this or current), if no argument is specified, or of a note specified through **name**, **item** or **path**. (See more on paths). From v4.6, paths may also be used o the left side of the overall expression:

$AttributeA(note/item/path) = $AttributeB

$AttributeA(note/item/path) = $AttributeB(note/item/path)

Using query back-references

NOTE: the following syntax can only be used in the context of a query. In a query (an agent's query or an if() condition in an action) it is possible to combine a regex query with an action that uses the value of the found regex:

query: `$Text.contains("email: <(.+)>")`

action: `$TheAddress=$1;`

…will set TheAddress attribute value to the regex found in text. It is assumed that back-references $1-$9 may be used, assuming the regex generates more than one such references.

== (i.e. value equality)

Operator Type: Operator [other Operator type actions]
Operator Scope of Action: Query [operators of similar scope]
Operator Purpose: Query Boolean [other Query Boolean operators]
Operator First Added: Baseline
Operator Last Altered: As at baseline

== (i.e. value equality)

The operator to test equality (i.e. 'is the same as') is '==', two equals signs. Note that this replaces older syntax where a single equals sign was used contextually for both assignment and equality tests.

This operator is used either in agent queries or in the conditional part of an if(condition){action} code. It is the functional opposite of the '!=' inequality test.

This test cannot be meaningfully applied to Set or List type attribute data, as the entire attribute value is matched, rather than individual values as might otherwise be assumed. For these data types use the .contains() or .icontains() operator instead, noting the scope for ambiguous matching due to stemming of words ("car" will match "car", "cars" and "carrot").

Because equality comparisons of Date-type data match at day scope, rather than full date/time values (for legacy reasons), use interval() to compare Date-type equality.

Equality testing can be negated, i.e. tested for a non-match, or combined with greater/less than for a range of tests as further explored in Basic Comparison Codes.

For a case-insensitive lexical equality test, use a lowercase on-the-fly transform:

```
"Absquatulate".lowercase == "absquatulate"
```

If we set $MyString to "Absquatulate", then:

```
$MyString.lowercase == "absquatulate"
```

If $MyOtherString(Some note) has the value "absquatulate", then:

```
$MyString.lowercase == $MyOtherString(Some note)
```

Note the stored left-side value is not altered, but its transformed version is used in the test giving a case-insensitive comparison. This method only works for upper/lower case comparisons; accented characters are lexically different characters regardless of case.

Equality and List/Sets

Using == (and !=) with Lists & Sets means you are checking the *entire* literal contents, i.e. string like "ant;bee;cow" rather than by individual sub-value: "ant" and "bee" and "cow". Thus the equality test cannot be used to check if the attribute contains a discrete value, use .contains() instead (or .icontains() for case-insensitive matches). Importantly, when testing a string (or expression resolving to a string) equivalence against a list, it is the list that must be tested using .contains(). This is best shown using string literals representing a String type and a List type:

`"cow".contains("ant;bee;cow")` (does not work as expected)

`"ant;bee;cow".contains("cow")`

`"ant;bee;Cow".contains("cow")`

The first, testing the string, resolves to `false` but the second, testing the list, gives `true`. The third is `false` but would be `true` if using an .icontains() test. Thus when equivalence testing a string against a list, always run the .contains() on the *list* and not the string.

To test two lists hold the same values (and only those values), in the same case, in the same order, the == equivalence operator can be used as this tests the stored concatenated value lists in each case. To check common items shared by two lists use List/Set.intersect().

Testing a List vs. a Set, it would be sensible to apply a . sort() or .isort() to each, reflecting that the sort state of a list is unknown and a == test compares the stored concatenated value string: the test would fail if the lists held the same values but stored in different orders. This shows up a difference between lists and sets. Although the literal value of a set may hold values in any order, when tested in code, they are being tested after sorting into (some**) order. Incidentally, this is why you cannot set a sort order as you can with a list as internally your given sort order is ignored. Consider $MySetA and $MyListA both with the values [ant;bee;cow]. $MySetB and $MyListB both have the value [bee;ant]. So:

`$MySetA == $MySetB` gives `false`

`$MyListA == $MyListB` gives `false`

both are expected. Now, make both the B attributes value "bee;ant;cow"

`$MySetA == $MySetB` gives `true`

`$MyListA == $MyListB` gives `false`

This is because the Set compares the literal result of sorted values, whereas the List

does not. But:

`$MyListA == $MyListB.sort` gives `true`

This is in effect what's happening with the sets, as in:

`$MyListA.sort == $MyListB.sort`

whereas in our last example above, $MyListA was already in default sort order so no applied sort was required. From experiment, this seems to be a (case-sensitive) computer lexical sort, i.e. [Bee;ant;bee] with capitals preceding lowercase letters in the sort.

> (i.e. greater than)

Operator Type: Operator [other Operator type actions]
Operator Scope of Action: Query [operators of similar scope]
Operator Purpose: Query Boolean [other Query Boolean operators]
Operator First Added: Baseline
Operator Last Altered: As at baseline

> (i.e. greater than)

The operator used to test if the left side of an expression is greater than the right side. This operator is used either in agent queries or in the conditional part of an if(condition){action} code. If $MyNumber is 3:

`$MyNumber > 2` is `true`

`$MyNumber > 4` is `false`

Further explored in Basic Comparison Codes.

>= (i.e. greater than or equal to)

Operator Type: Operator [other Operator type actions]
Operator Scope of Action: Query [operators of similar scope]
Operator Purpose: Query Boolean [other Query Boolean operators]
Operator First Added: Baseline
Operator Last Altered: As at baseline

>= (i.e. greater than or equal to)

The operator used to test if the left side of an expression is greater than or equal to the right side. This operator is used either in agent queries or in the conditional part of an if(condition){action} code. If $MyNumber is 3:

`$MyNumber >= 2` is `true`

`$MyNumber >= 3` is `true`

`$MyNumber >= 4` is `false`

Further explored in Basic Comparison Codes; in some date comparison cases, a two term query may be needed instead.

| (i.e. query logical OR join)

Operator Type: Operator [other Operator type actions]
Operator Scope of Action: Query [operators of similar scope]
Operator Purpose: Query Boolean [other Query Boolean operators]
Operator First Added: Baseline
Operator Last Altered: As at baseline

| (i.e. query logical OR join)

The pipe, |, is used in queries and conditional statements as a logical OR join. Thus:

```
if ($HasStock == true | $Price > 20) {$Badge="ok";}
```

The note's $Badge is only set if **either** the first **or** the second argument are `true`.

See also the AND join and conditional statements using multiple arguments.

|= (i.e. logical OR assignment)

Operator Type: Operator [other Operator type actions]
Operator Scope of Action: Item [operators of similar scope]
Operator Purpose: Assignment [other Assignment operators]
Operator First Added: Baseline
Operator Last Altered: As at baseline

|= (i.e. logical OR assignment)

To make it easier to write rules succinctly, you may use the assignment:

```
$TheAttribute |= the_expression;
```

…which sets *$TheAttribute* to the value of *the_expression* if it is already `true` **OR** if *the_expression* is `true`. An attribute that has no locally set (or inherited) value is evaluated as `false`. Thus there are two possible outcomes:

- if *$TheAttribute* is set to a value that evaluates (the left side of the code) as `true`. Therefore, **regardless** of the state of *the_expression*, *the_expression* (the right side of the code) is **not** evaluated and *$TheAttribute* retains its current value.
- if *$TheAttribute* is set to a value evaluates (the left side of the code) as `false`. Therefore *the_expression* (the right side of the code) **is** evaluated. If *the_expression* evaluates as `true` *$TheAttribute* takes the value of *the_expression*. Or subsequent iteration of the code *$TheAttribute* thus evaluates as `true`. But, if *the_expression* evaluated to `false`, both it and the overall action evaluate as `false`, i.e. *$TheAttribute*'s value remains unchanged.

A more verbose (and less efficient at scale) way to write the same test without the `|=` operator is

```
if((!$TheAttribute)){ ..o the_expression;}
```

where '(!$TheAttribute)' is a short-form test meaning if the value of attribute named *TheAttribute* tests as `false`. A `false` value arises differently for for different attribute data types, but for for a string, the above can be further unpacked as:

```
if($TheAttribute!=""){ ..o the_expression;}
```

In practical terms this means the left side, usually an attribute, is set to the right side value *only* if it is not already set locally at note level. This is because for most attributes, especially new user attributes, the default value will evaluate as `false`. But, *$TheAttribute*'s data type does not have to be Boolean.

For new String-type attributes (and string-based Action/Color/File/Font/Interval/List/Set/URL types) the default is an empty string "". For Number-type, is it 0 (zero). For Date-type, it is the string "never". For Boolean-type, is it ' `false`' (with r enclosing quotes, and shown as un-ticked if displayed via a tick-box). The "", 0 and "never" values—for the appropriate data types—evaluate as `false`. But, be aware that not all system attributes follow this assumption. For instance, the

$Color default is preset to use a named Tinderbox colour, so its default value (even if inherited) would evaluate as `true`.

Prototypes have no direct effect as the outcome, as it initially depends on the evaluation of the left-side attribute value regardless of whether document default, prototype inherited or locally set.

This operator's behaviour makes the |= usage very useful for doing tasks like making code run only once; on the second pass the left side already has a value so no change occurs. This avoids scenarios like successive applications of a rule causing multiple concatenation of strings (one extra each iteration).

For example, for the Boolean-type attribute $Urgent:

```
$Urgent |= any(children,$Urgent);
```

A project is urgent if it has been declared to be urgent itself, or if any child is urgent.

If using |= assignments, it can be useful to have a means to reset an attribute to default to re-enable |= value assignment.

See also the logical AND assignment (which is likely used less often).

$AttributeName (i.e. a short form test for value)

Operator Type: Operator [other Operator type actions]
Operator Scope of Action: Item [operators of similar scope]
Operator Purpose: Query Boolean [other Query Boolean operators]
Operator First Added: Baseline
Operator Last Altered: As at baseline

$AttributeName

In queries and conditional action code expressions, using just an attribute name preceded by a $ character functions as a shortened form of the Boolean test for a `true` value. Thus the following are functional equivalents:

```
$MyBoolean==true
$MyBoolean
```

In both cases the result is `true` if the value of $MyBoolean is `true`.

In all other contexts, the $AttributeName syntax implies a reference to that attribute.

Usefully Tinderbox also useful supports such short-form $AttributeName tests for all the other attribute data types, returning `true` if the attribute has a *non-default* value. Per data type, this equates to long form tests like:

```
$MyAction!=""
$MyColor!=""
$MyDate!=never
$MyFile!=""
$MyInterval!="00:00"
$MyList!=[] or $MyList!=""
$MyNumber!=0
$MySet!=[] or $MySet!=""
$MyString!=""
$MyURL!=""
```

In all these cases a short-form test returns `true` if the attribute value *is not* the default for that data type.

A reverse short form test is also offered by (!$AttributeName), i.e. the same syntax with a preceding exclamation mark *and* enclosing parentheses. In theory, the latter parentheses are not necessary, but in practice it helps Tinderbox when parsing a query.

$AttributeName[(scope)]

Operator Type: Property [other Property type actions]
Operator Scope of Action: Item [operators of similar scope]
Operator Purpose: Assignment [other Assignment operators]
Operator First Added: Baseline
Operator Last Altered: As at baseline
Operator Uses Scoped Arguments: [More on scoped arguments in Action Code]

$AttributeName

In action code contexts, an attribute's name prefixed with a $-character implies a reference to that attribute, thus a placeholder for that attribute's value. In a coding context is acts as a variable name where the variable name must be that of an existing attribute and the variable's value is got from or set to that attribute.

A reference on the right side of an expression fetches the attribute's value. A reference on the left side of an expression has its value set to the result of the right side. The following example combines these uses whereby in the current object (a note, agent, adornment, etc.), the user attribute $MyNumber is being set to the value of system attribute $ChildCount:

```
$MyNumber = $ChildCount;
```

In queries and conditional expressions, and attribute reference implies a shortened Boolean test.

The test can be turned in a negative (i.e. that no value is set) by using a '!' prefix: see !$AttributeName, occasionally if the !-prefix does not work, try enclosing the expression in parentheses (!$AttributeName) to help signal intent to Tinderbox' action code parser.

$AttributeName(scope)

Besides the basic form above, which is essentially `$AttributeName(this)`, an offset address can also be made to the value of an attribute in another object— i.e. *other than the one in current focus*, by using an extended syntax where th **scope** argument (defining **scope**) to refer to one or more note(s).

Importantly, here the **scope** argument *is not evaluated for expressions* unlike in some action code operators. This means that here, **scope** cannot be a complex code expression. However, where that need arises eval() may offer a workarour or consider using a user attribute to hold the output of the expression and then use that attribute's value (i.e. in the manner of bullet #4 above).

The extended syntax form may be used on the left or right side of an action code expression. In other words, you can use this syntax to fetch (right side) or set (left side) an attribute value from some other object. Generally, offset addresses ar used on the right side to fetch data from another object.

Examples, right side:

```
$MyString = $MyString("Some other note");
$MyNumber = $MyNumber("A root container/Some Container/Some other note");
$MyString = $MyString(agent); (this designator only works in agents)
$MyString = $MyString(adornment); (this designator only works in adornments)
$MyList = $MyString("Some other note;Another note;Different note");
```

Examples, left-side:

```
$MyString("Another note") = $MyString("A note");
$MyDate(parent) = $MyDate;
```

In fact, an offset can even be used on both sides of the expression. For instance, the rule in a note "Some note" might use the following code to refer to attributes in two other notes:

```
$MyColor("A note") = $MyColor("Another note");
```

$N (query back-reference)

Operator Type: Property [other Property type actions]
Operator Scope of Action: Query [operators of similar scope]
Operator Purpose: Data manipulation [other Data manipulation operators]
Operator First Added: Baseline
Operator Last Altered: As at baseline

Syntax note: *Operators without any defined mandatory arguments may omit their empty closing parentheses*

$N

The value of the numbered back-reference for the current query (agent or if() conditional).

The valid range for for N is from `0` (zero) to `9`.

The back-reference $0 always refers to the the whole matched string (or sub-string) for the stated query regex, i.e. it may match all or part of the target string. $1 to $9 refer to any further defined back-references *within* the overall regex, i.e. sub-strings within $0.

Back-references are numbered in the order created. The order is usually left-to right in order the parentheses open, noting that this allows for nesting of back-references.

Query back-references are discussed in fuller detail here.

abs(sourceNum)

Operator Type: Function [other Function type actions]
Operator Scope of Action: Item [operators of similar scope]
Operator Purpose: Mathematical [other Mathematical operators]
Operator First Added: Baseline
Operator Last Altered: As at baseline

abs(sourceNum)

abs() computes the absolute value of its **sourceNum** argument. This is the non-negative value of **sourceNum** without regard to its sign. Thus both these result in a $MyNumber value of 3.5:

```
$MyNumber = abs(3.5);
$MyNumber = abs(-3.5);
```

The operator can also evaluate a numerical attribute:

```
$MyNumber = abs($SomeNumber);
```

A numerical string attribute will be parsed to a number. If $MyString is a value of "-5", $MyNumber will be 5:

```
$MyNumber = abs($MyString);
```

action([scope,]codeStr)

Operator Type: Function [other Function type actions]
Operator Scope of Action: Item [operators of similar scope]
Operator Purpose: Data manipulation [other Data manipulation operators]
Operator First Added: Baseline
Operator Last Altered: 9.5.0
Operator Has Optional Arguments: [More on optional operator arguments]

action([scope,]codeStr)

action(codeStr)

This function allows execution of the **codeStr** without the need for a return value. This allows an action be set within an attribute, and perform that action as part of a rule or agent action. From v9.5.0, an additional optional **scope** argument was added allowing the action to be run on a note other than the current note

For example, the Rule

```
action('$Color= "bright red"');
```

will set the note's primary colour ($Color).

Note that that in its simplest form, an action() call is a *quote-enclosed* string. As action codes tend to use double-quoted strings, it may often be necessary to use single quotes for the outer enclosure, as in the example above.

The **action()** function returns the evaluated result of the **codeStr** within it. In the above case if $MyString were set to store the result, the value would be `true`—because the action completed successfully. So, in *most* cases, there is no requirement for a left-side attribute to accept any output. By comparison, the related eval(expression) function evaluates an expression and returns a value. Whereas eval() is designed to intentionally return an output, action() performs an action such as an assignment.

An action() call can be used with a `do(macro)` call to create a form of function as macros allow input arguments (its arguments): see do().

The action() call is particularly useful during export where it is desirable to run some action code in the context of the template during template rendering (evaluation)—see more below.

Calling stamp code via action()

If a stamp's code is long/complex it can be convenient to store it in a code note. Thus if a code note 'Test-stamp' held the action code `$Colour="red";`, then a stamp with the code:

```
action($Text("Test-stamp"));
```

when run would result in the stamped note(s) turning red. The example is trivial but shows the technique. Note the offset address in the stamp to the code note is case sensitive and should use a unique $Name (or else cite the full $Path to the code note). Local attribute references, i.e. $Color or $ChildCount, are bound to the note being stamped: it is not possible to reference values in the code note using a designator.

On-the-fly Attribute references using action()

Occasionally it is necessary to make at attribute value reference string from a variable, i.e. if the variable holds 'Path' the result being '$Path' as opposed to the value of $Path. Whilst $MyString refers to the value of attribute MyString, what if the attribute name is itself the value of a variable in the current code? Here action() solves the problem. For instance, if vAnAttr holds as its value the name of an attribute, e.g. "SomeAttribute" then the following *does not work*:

```
$MyString = "$" + vAnAttr;
```
WRONG ($MyString is the value of $SomeAtttribute)

as $MyString is not set to the value of $SomeAttr. Instead use action():

```
action('$MyString = "$" + vAnAttr');
```
CORRECT ($MyString is the string '$SomeAtttribute')

To build a series of $-prefixed attribute references in a loop, use action() instead of eval(), as described in more detail here.

values(scope, code)

From v9.5.0, if an optional first **scope** argument is provided, the **code** is not run on the current note but on that defined by **scope** (defining **scope**). Thus:

```
values("Some note","$Color = 'blue';");
```

will set the colour of note "Some note" to blue.

This option can avoid needing to use offset references in code, especially if the offset address is being defined by a variable.

any(scope, condition)

Operator Type: Function [other Function type actions]
Operator Scope of Action: Group [operators of similar scope]
Operator Purpose: Non-query Boolean [other Non-query Boolean operators]
Operator First Added: Baseline
Operator Last Altered: As at baseline
Operator Uses Regular Expressions: [More on regular expressions in Tinderbox]
Operator Uses Scoped Arguments: [More on scoped arguments in Action Code]
Operator Has Conditional Arguments: [More on conditional operator arguments]

any(scope, condition)

This Boolean operator examines notes at **scope** (defining scope) and determines whether any note(s) in the defined group evaluates **condition** as `true`.

scope may be any group designator, or group defined by find().

The overall **condition** must not be enclosed in quotes, though a literal string value within the query will need to be quoted. For example:

```
any(children,$Status=="Important")
any(children,$Overdue==true)
any(children,$Overdue) (using short form test)
any(children,$Overdue==false)
any(children,!$Overdue) (using short form test)
```

If trying to resolve contains() for multiple matches, use any(children,$Name=="string"). Thus of it is desired for an agent to list the parent containers of all notes titled 'foo', the agent query would be:

```
any(children,$Name=="foo")
```

If agents are present, it may be sensible to filter for aliases:

```
any(children,$Name=="foo") & !$IsAlias
```

See also: every().

atan(radiansNum)

Operator Type: Function [other Function type actions]
Operator Scope of Action: Item [operators of similar scope]
Operator Purpose: Mathematical [other Mathematical operators]
Operator First Added: Baseline
Operator Last Altered: As at baseline

atan(radiansNum)

atan() converts its **radiansNum**, in *radians*, to the arctangent of that value.

```
$MyNumber = atan(6);
```

returns 1.405647649 for an input of 6 radians.

attribute(attributeNameStr).keys

Operator Type: Function [other Function type actions]
Operator Scope of Action: Document [operators of similar scope]
Operator Purpose: Document configuration [other Document configuration operators]
Operator First Added: Baseline
Operator Last Altered: As at baseline

---

attribute(attributeNameStr).keys

This returns the attribute dictionary keys for the attribute named in the **attributeNameStr** argument. Keys are returned in the order listed for attribute(), and *not in alphabetical order*—as might be intuited. Being a Dictionary operator, it is possible to retrieve all the above as a list of key:value pairs. Note that any attribute name (any valid name) must be supplied as first argument:

```
$MyList = attribute("Width").keys;
```

returns the entire contents of the Dictionary key:value pairs for the attribute `Width`.

The attribute name can be a variable , so a more useful method is to iterate the list of keys:

```
var:list vKeys;
var:string vOutput;
vKeys = attribute(anAttribute).keys;
vKeys.each(aKey){
 vOutput += aKey + ": " + attribute(anAttribute)[aKey] + "\n";
};
$Text = vOutput;
```

By combining with document(), it is possible to make a stamp that creates a listing of all user attributes in the current document, iterate each one, report only keys with a value and place the resulting data in the $Text of the stamped note the stamp code is;

```
var:list vKeys;
var:string vOutput;
var:list vAttributes = document["user-attributes"];
vOutput += "TBX filename " + document[name] + "\n";
vOutput += "Number of  user attributes: " + vAttributes.count + "\n--------------------\n";
vAttributes.each(anAttribute){
 vOutput += "--------------------\n" + anAttribute + "\n" +"----------\n";
 vKeys = attribute(anAttribute).keys;
 vKeys.each(aKey){
  if(attribute(anAttribute)[aKey] != "" & aKey != "category"){
   vOutput += aKey + ": " + attribute(anAttribute)[aKey] + "\n";
  };
 };
};
$Text = vOutput;
```

attribute(attributeNameStr)[keyStr]

Operator Type: Function [other Function type actions]
Operator Scope of Action: Document [operators of similar scope]
Operator Purpose: Document configuration [other Document configuration operators]
Operator First Added: Baseline
Operator Last Altered: 9.5.0, 9.6.0

---

attribute(attributeNameStr)[keyStr]

The **attribute()** operator, for the specified attribute **attributeNameStr**, returns a Dictionary of key values that describe that attribute. Current keys include (names are all-lowercase and case-sensitive):

- **category:** the category (group) in which the attribute appears
- **default:** the default value
- **suggested:** the suggested values (if set)
- **type:** a string describing the data type of the attribute
- **description:** a short description the the attribute (for user attributes, if set)
- **lines**: the number of lines used for display of values in Displayed Attributes and Get Info tables.

Key values

If any key is not set, e.g. there are no 'suggested' values, then an empty string is returned for that key.

All keys return as string, called via **keyStr**, but the most appropriate attribute type is shown as the recipient of the data:

```
$MyList = attribute("Width")["default"]
$MyList = attribute("Width")["suggested"]
$MyString = attribute("Width")["category"]
$MyString = attribute("Width")["type"]
$MyString = attribute("Width")["description"]
```

Note that when using literal (i.e. actual) attribute name or keys it is recommended the word(s) are quote enclosed as above. If using variables, as in the earlier code examples, quotes are *not* used as this aids the Tinderbox parser detecting literal vs. variable usage.

Editing key values

Some keys can be modified via action code, all others are *read-only*. Actions may modify the **default** or **suggested** values of an attribute, and from v9.6.0 also **description** and **lines**:

```
attribute("attributeName")["suggested"]="value 1; value 2";
```

sets two values for the attribute's suggested values list. Likewise a default value can be set:

```
attribute("attributeName")["default"]="value 1";
```

Or a description:

```
attribute("Price")["description"]="The price of the item.";
```

Thus, for instance, action code might find the discrete values for an attributes (use values() or collect(), etc.) and use that list to set/update the attributes suggested values (i.e. the **suggested** key value).

Setting property values

From v9.5.0, both the attribute name and/or the facet to be set may be enclosed in quotation marks (previously, use of quotes caused a failure):

```
attribute(AttributeName)[facet]="testing";
```

or

```
attribute("AttributeName")["facet"]="testing";
```

attributeEncode(dataStr)

Operator Type: Function [other Function type actions]
Operator Scope of Action: Item [operators of similar scope]
Operator Purpose: Formatting [other Formatting operators]
Operator First Added: Baseline
Operator Last Altered: As at baseline

---

attributeEncode(dataStr)

This function encodes *all* instances of the following in the argument string **dataStr** to numeric HTML entities (e.g. like '
'):

- left angle bracket
- right angle bracket
- ampersand
- (straight) double quote
- (straight) apostrophe
- line feed character (ASCII #10). This assists with OPML work as paragraph breaks are not allowed within XML attribute values.

For example, if the source **dataStr** is "this & that":

`$MyString = attributeEncode("this & that")` outputs `"this & that"`.

An older export code equivalent was ^opmlEncode()^; the latter is *deprecated* in favour of the form ^value(attributeEncode(...))^.

avg_if(scope, condition, expressionStr)

Operator Type: Function [other Function type actions]
Operator Scope of Action: Conditional Group [operators of similar scope]
Operator Purpose: Mathematical [other Mathematical operators]
Operator First Added: Baseline
Operator Last Altered: As at baseline
Operator Uses Scoped Arguments: [More on scoped arguments in Action Code]
Operator Has Conditional Arguments: [More on conditional operator arguments]

---

avg_if(scope, condition, expressionStr)

The function avg_if() computes the arithmetic mean of a list of the values of **scope** items, and returns a List-type data. Each of the items in **scope** (defining **scope**) is tested and ignored it if it fails to meet **condition**, otherwise items supply a value to then adding the value of each list item evaluated using the designated **expressionStr**. Be aware that the per-item value might be a literal value, an attribute value, or a value/string of content based on that item's **expressionStr**.

For a related, less focussed, operator see avg().

This computes the arithmetic mean of the value of a **expressionStr** in a **group**, as filtered by a **condition** expression. See avg() for a for a related non-conditional operator.

scope describes the note(s) to be examined and may be any group designator including a find() query.

condition is action code forming a valid conditional query test, i.e. it equates to `true` when matched. Some query-style operators terms may allow use of regular expressions.

expressionStr may be any valid expression, but will usually be a reference to an attribute; short form Boolean attribute expressions are acceptable.

For example:

```
$MyNumber = avg_if(children(/Catalog),$Price!=0,$Price);
```

sets $MyNumber to the average price of all the items in Catalog, ignoring any items that have a price of 0.

avg(scope, expressionStr)

Operator Type: Function [other Function type actions]
Operator Scope of Action: Group [operators of similar scope]
Operator Purpose: Mathematical [other Mathematical operators]
Operator First Added: Baseline
Operator Last Altered: As at baseline
Operator Uses Scoped Arguments: [More on scoped arguments in Action Code]
Operator Has Newer Dot-Operator Variant: Yes

---

avg(scope, expressionStr)

This computes the arithmetic mean of the value of a **expressionStr** in a **scope**. See avg_if() for a for a related conditional operator.

scope describes the notes to be examined and may be any group designator including a find() query.

expressionStr may be any valid expression, but will usually be a reference to an attribute; short form Boolean attribute expressions are acceptable.

For example:

```
$MyNumber = avg(children(/Catalog),$Price);
```

returns the average price of all the items in Catalog.

between(valueNum, minNum, maxNum)

Operator Type: Function [other Function type actions]
Operator Scope of Action: Item [operators of similar scope]
Operator Purpose: Query Boolean [other Query Boolean operators]
Operator First Added: Baseline
Operator Last Altered: As at baseline

---

between(valueNum, minNum, maxNum)

Returns Boolean *true* if the **valueNum** is greater or equal to **minNum** and less than **maxNum**. The comparison method is based on the type of **valueNum**; numerical, lexical, string and set comparisons are chosen as needed.

The logic is:

((**valueNum** >= **minNum**) & (**valueNum** < **maxNum**))

Thus between() is `true` if **valueNum**==**minNum** but `false` if **valueNum**==**maxNum**.

If $MyNumber is 7, then:

`$MyBoolean = between($MyNumber, 1, 5);` is `false`

`$MyBoolean = between($MyNumber, 1, 9);` is `true`

More realistically the operator would be used in a query or a conditional expression:

`if(between($MyNumber, 1, 5)){…}else{…};` would test as `true` and execute the code in the first conditional branch.

Do not use this operator for testing Date-type attributes directly. Either use days() instead or use between() with format() or Date.format() to create a suitable string values for testing.

In the query creation pop-ups of agent and Find dialogs this function is listed as "is between".

Legacy issues

This operator replaces the legacy #between query operator.

capitalize(dataStr)

Operator Type: Function [other Function type actions]
Operator Scope of Action: Item [operators of similar scope]
Operator Purpose: Formatting [other Formatting operators]
Operator First Added: Baseline
Operator Last Altered: As at baseline
Operator Has Newer Dot-Operator Variant: Yes

---

capitalize(dataStr)

The contents of text string **dataStr** is returned with the first letter of each word in upper case.

Functionally equivalent to String.capitalize.

if $MyString is "hello world":

```
$MyString = $MyString.capitalize();
```

sets it to "Hello World".

ceil(sourceNum)

Operator Type: Function [other Function type actions]
Operator Scope of Action: Item [operators of similar scope]
Operator Purpose: Mathematical [other Mathematical operators]
Operator First Added: Baseline
Operator Last Altered: As at baseline
Operator Has Newer Dot-Operator Variant: Yes

ceil(sourceNum)

rounds the number value of **sourceNum** up to next whole integer.

See also Number.floor() and Number.round().

If $MyNumber is 3.2 then:

```
$MyNumber = ceil($MyNumber);
```

sets $MyNumber to 4. Note unlike normal rounding the value is set upwards to the next integer (i.e. whole number).

changed([scope])

Operator Type: Operator [other Operator type actions]
Operator Scope of Action: Item [operators of similar scope]
Operator Purpose: Data manipulation [other Data manipulation operators]
Operator First Added: 9.6.0
Operator Last Altered: As at baseline
Operator Uses Scoped Arguments: [More on scoped arguments in Action Code]
Operator Has Optional Arguments: [More on optional operator arguments]

changed()

changed([scope])

New to v9.6.0, the **changed()** operator tells the current item to update itself. A likely use for the operator is to refresh Poster notes. There is an optional **scope** argument, indicating which note(s) need updating. If no scope argument is passe the target is assumed to be the currently selected note.

The operator needs no left-side argument, so may be called as a simple expression:

```
changed();
```

or, more specifically:

```
changed("Some poster note");
```

collect_if(scope, condition, expressionStr)

Operator Type: Function [other Function type actions]
Operator Scope of Action: Conditional Group [operators of similar scope]
Operator Purpose: Dictionary, Set & List operations [other Dictionary, Set & List operations operators]
Operator First Added: Baseline
Operator Last Altered: As at baseline
Operator Uses Regular Expressions: [More on regular expressions in Tinderbox]
Operator Uses Scoped Arguments: [More on scoped arguments in Action Code]
Operator Has Conditional Arguments: [More on conditional operator arguments]
Operator Has Newer Dot-Operator Variant: Yes

collect_if(group, condition, expressionStr)

The function collect_if() returns a List by collecting all the notes in **scope** (defining **scope**), testing each note in **scope** and ignoring it if it fails to meet **condition**, then adding the value of each list item evaluated using the designated **expressionStr**. Be aware that the per-item value might be a literal value, an attribute value, or a value/string of content based on that item's **expressionStr**.

For a related, less focussed, operator see collect().

scope may be any group designator, or group defined by find(). In addition, **scope** may be argument that designates a particular (single) note other than 'this' collect() omits notes for which $Searchable is `false`.

condition is action code forming a valid conditional test, i.e. it equates to `true` when matched.

expressionStr can be any expression, but is typically an attribute's name. If the collected per-item value type is multi-value, i.e. Set or a List type data , collect() adds its elements to the returned list. Thus if an item's evaluated value is a list of 3 terms, that item contributes 3 items to the operator's returned list rather than 1 item.

If the collected **expressionStr** is not a Set or List type attribute, but contains a semicolon, quotation mark, or parentheses, the value will be added to the result as a quoted string. This behaviour is designed correct a variety of confusing edge cases.

For example,

```
$MyList = collect_if(children,$Status=="Important",$Name);
```

will construct a List of the names of all of this note's important children.

Note that collect_if's function is related to agents; many tasks you might perform with collect_if could be done as well, or better, with an agent.

If a list of unique values is required, i.e. set rather than a list, simply pass the output to a Set attribute. Thus if:

```
$MyList = collect_if(children,$Age>6,$FavFruit);
```
$MyList is [Apples;Oranges;Pears;Apples]

```
$MySet = collect(children,$Age>6,$FavFruit);
```
$MySet is [Apples;Oranges;Pears]

If the collected attribute is a set or a list, collect() adds its elements to the result. If the collected attribute is not a set or a list, but contains a semicolon, quotation mark, or parentheses, the value will be added to the result as a quoted string. This should avoid a variety of confusing edge case outcomes.

collect_if() resets its regular expression match list for each note it tests. Thus, $1 will be the first matched subexpression for *this* note, rather than the first matched subexpression for the *entire* collect_if() statement.

collect_if() vs. List/Set.collect_if()

For working with **List**s or **Set**s of actual values, also see List/Set.collect_if().

collect(scope, expressionStr)

Operator Type: Function [other Function type actions]
Operator Scope of Action: Group [operators of similar scope]
Operator Purpose: Dictionary, Set & List operations [other Dictionary, Set & List operations operators]
Operator First Added: Baseline
Operator Last Altered: As at baseline
Operator Uses Scoped Arguments: [More on scoped arguments in Action Code]
Operator Has Newer Dot-Operator Variant: Yes

collect(scope, expressionStr)

The function collect returns a List by collecting all the notes in **scope** (defining **scope**), and adding the value of each list item evaluated using the designated **expressionStr**. Be aware that the per-item value might be a literal value, an attribu value, or a value/string of content based on that item's **expressionStr**.

For a related, more focussed, operator see collect_if().

scope may be any group designator, or group defined by find(). In addition, **scope** may be argument that designates a particular (single) note other than 'this'. collect() omits notes for which $Searchable is `false`.

expressionStr can be any expression, but is typically an attribute's name. If the collected per-item value type is multi-value, i.e. Set or a List type data , collect() adds its elements to the returned list. Thus if an item's evaluated value is a list of 3 terms, that item contributes 3 items to the operator's returned list rather than 1 item.

If the collected **expressionStr** is not a Set or List type attribute, but contains a semicolon, quotation mark, or parentheses, the value will be added to the result as a quoted string. This behaviour is designed correct a variety of confusing edge cases.

For example,

```
$MyList = collect(children,$Name);
```

constructs a set with the name of each child of the note.

For example,

```
$MyList = collect(children(/agents/books),$Name);
```

does the same for children of the note 'books' inside 'agents'.

```
$MyList = collect(children,$Width * $Height);
```

collects a series of numerical values of each child's map icon height/width in Tinderbox map units. Thus if a child had a $Width of 4 and $Height of 2, the collect() result for that item would be 8.

If a list of unique values is required, i.e. set rather than a list, simply pass the output to a Set attribute. Thus if:

```
$MyList = collect(children,$FavFruit);
```
$MyList is [Apples;Oranges;Pears;Apples]

```
$MySet = collect(children,$FavFruit);
```
$MySet is [Apples;Oranges;Pears]

collect() vs. List/Set.collect()

For working with **List**s or **Set**s of actual values, also see List/Set.collect().

Color.blue()

Operator Type: Property [other Property type actions]
Operator Scope of Action: Item [operators of similar scope]
Operator Purpose: Color [other Color operators]
Operator First Added: Baseline
Operator Last Altered: As at baseline

Syntax note: *Operators without any defined mandatory arguments may omit their empty closing parentheses*

Color.blue()

Color.blue

This property sets or returns the value of the blue channel of an RGB colour. The value may be set with a number (0–255) or hex number (#00–#ff). The return value (if coerced to a string or number) is always a number, i.e. 255 and not #ff. By comparison the rgb() operator requires that the values for all three colour channels be set. Examples:

```
$MyColor.blue = 255;
$MyColor.blue = "#ff";
$MyOtherColor.green = $MyColor.blue;
```
(value of $MyOtherColor green channel is that of $MyColor blue channel)

```
$MyNumber = $MyColor.blue;
```
(gives 255)

```
$MyString = $MyColor.blue;
```
(gives "255")

Color.brightness()

Operator Type: Property [other Property type actions]
Operator Scope of Action: Item [operators of similar scope]
Operator Purpose: Color [other Color operators]
Operator First Added: Baseline
Operator Last Altered: As at baseline

Syntax note: *Operators without any defined mandatory arguments may omit their empty closing parentheses*

Color.brightness()

Color.brightness

This property sets or returns the brightness value of the associated Color-type attribute. The value is a number in the range 0-100. This operator is intended to replace the existing brightness() operator. Examples:

```
$MyColor.brightness = 50;
```
(sets 50% brightness)

```
$MyColor2.brightness = $MyColor.brightness;
```
(matches brightness levels)

```
$MyNumber = $MyColor.brightness;
```
(returns 50, from above example)

Color.format()

Operator Type: Function [other Function type actions]
Operator Scope of Action: Item [operators of similar scope]
Operator Purpose: Formatting [other Formatting operators]
Operator First Added: Baseline
Operator Last Altered: As at baseline

Color.format()

Returns **Color** as a hex-string, regardless of source data is a hex value or named color.

Thus if $MyColor is "bright red":

```
$MyString = $MyColor.format();
```
gives "#ff0000"

```
$MyString = $MyColor;
```
gives "bright red"

Whilst if $MyColor is "#330099":

```
$MyColor.format();
```
gives "#330099"

```
$MyString = $MyColor;
```
gives "#330099"

Therefore for a Color-type attribute set to a named colour, to get that name string simply pass the attribute's value to a string. So if $MyColor is "bright blue", this:

```
$MyString = $MyColor;
```

…gives "bright blue" and this:

```
$MyString = $MyColor + " ("+$MyColor.format()+")";
```

…gives "bright blue (#0000ff)".

This supplements the existing format() function.

Color.green()

Operator Type: Property [other Property type actions]
Operator Scope of Action: Item [operators of similar scope]
Operator Purpose: Color [other Color operators]
Operator First Added: Baseline
Operator Last Altered: As at baseline

Syntax note: *Operators without any defined mandatory arguments may omit their empty closing parentheses*

Color.green()

Color.green

This property sets or returns the value of the green channel of an RGB colour. The value may be set with a number (0–255) or hex number (#00–#ff). The return value (if coerced to a string or number) is always a number, i.e. 255 and not #ff. By comparison the rgb() operator requires that the values for all three colour channels be set. Examples:

```
$MyColor.green = 255;
$MyColor.green = "#ff";
$MyOtherColor.red = $MyColor.green;
```
(value of $MyOtherColor red channel is that of $MyColor green channel)

```
$MyNumber = $MyColor.green;
```
(gives 255)

```
$MyString = $MyColor.green;
```
(gives "255")

Color.hue()

Operator Type: Property [other Property type actions]
Operator Scope of Action: Item [operators of similar scope]
Operator Purpose: Color [other Color operators]
Operator First Added: Baseline
Operator Last Altered: As at baseline

Syntax note: *Operators without any defined mandatory arguments may omit their empty closing parentheses*

Color.hue()

Color.hue

This property sets or returns the hue value of the associated Color-type attribute. The value is a number in the range 0-360. This operator is intended to replace the existing hue() operator. Examples:

```
$MyColor.hue = 270;
```
(sets hue of 270 degrees)

```
$MyColor2 = $MyColor.hue;
```
(matches hues)

```
$MyNumber = $MyColor.hue;
```
(returns 270, from above example)

Color.red()

Operator Type: Property [other Property type actions]
Operator Scope of Action: Item [operators of similar scope]
Operator Purpose: Color [other Color operators]
Operator First Added: Baseline
Operator Last Altered: As at baseline

Syntax note: *Operators without any defined mandatory arguments may omit their empty closing parentheses*

Color.red()

Color.red

This property sets or returns the value of the blue channel of an RGB colour. The value may be set with a number (0–255) or hex number (#00–#ff). The return value (if coerced to a string or number) is always a number, i.e. 255 and not #ff. By comparison the rgb() operator requires that the values for all three colour channels be set. Examples:

```
$MyColor.red = 255;
$MyColor.red = "#ff";
$MyOtherColor.blue = $MyColor.red;
```
(value of $MyOtherColor blue channel is that of $MyColor red channel)

```
$MyNumber = $MyColor.red;
```
(gives 255)

```
$MyString = $MyColor.red;
```
(gives "255")

Color.saturation()

Operator Type: Property [other Property type actions]
Operator Scope of Action: Item [operators of similar scope]
Operator Purpose: Color [other Color operators]
Operator First Added: Baseline
Operator Last Altered: As at baseline

Syntax note: *Operators without any defined mandatory arguments may omit their empty closing parentheses*

Color.saturation()

Color.saturation

This property sets or returns the saturation value of the associated Color-type attribute. The value is a number in the range 0-100. This operator is intended to replace the existing saturation() operator. Examples:

```
$MyColor.saturation = 50;
```
(sets 50% saturation)

```
$MyColor2.saturation = $MyColor.saturation;
```
(matches saturation levels)

```
$MyNumber = $MyColor.saturation;
```
(returns 50, from above example)

compositeFor(nameStr)

Operator Type: Function [other Function type actions]
Operator Scope of Action: Composite [operators of similar scope]
Operator Purpose: Composite [other Composite operators]
Operator First Added: Baseline
Operator Last Altered: As at baseline

compositeFor(nameStr)

returns a List-type list of paths of all notes in the composite containing note **nameStr**.

```
$MyList = compositeFor("Total");
```

By using designator this for *item*, any note in the composite can refer to its containing composite.

```
$MyList = compositeFor(this);
```

If the name of a composite is known, but not the note name(s) or any of its constituents, use compositeWithName().

compositeFor(nameStr):count

Operator Type: Property [other Property type actions]
Operator Scope of Action: Composite [operators of similar scope]
Operator Purpose: Composite [other Composite operators]
Operator First Added: Baseline
Operator Last Altered: As at baseline

compositeFor(nameStr):count

returns the Number of notes in the composite containing note **nameStr**.

This code can also use the ' my' designator to give a shorter form. These are functional equivalents:

```
$MyNumber = compositeFor(this):count;
$MyNumber = my:count;
```

compositeFor(nameStr):kind

Operator Type: Property [other Property type actions]
Operator Scope of Action: Composite [operators of similar scope]
Operator Purpose: Composite [other Composite operators]
Operator First Added: Baseline
Operator Last Altered: As at baseline

compositeFor(nameStr):kind

returns the String name of the composite from which the composite containing note **nameStr** was instantiated.

For example, if a composite was created from the built-in 'list' composite, this function will return "list". The query:

```
$MyString = compositeFor(this):kind=="book";
```

would locate all notes that participate in composites instantiated from "book" – all books.

This code can also use the ' my' designator to give a shorter form. These are functional equivalents:

```
$MyString = compositeFor(this):kind;
$MyString = my:kind;
```

compositeFor(nameStr):name

Operator Type: Property [other Property type actions]
Operator Scope of Action: Composite [operators of similar scope]
Operator Purpose: Composite [other Composite operators]
Operator First Added: Baseline
Operator Last Altered: As at baseline

---

compositeFor(nameStr):name

returns the String name of the composite containing note **nameStr**. This expression is read/write:

```
$MyString = compositeFor(this):name="example";
```

renames the composite containing the note running this code to 'example'.

This code can also use the ' my' designator to give a shorter form. These are functional equivalents:

```
$MyString = compositeFor(this):name;
$MyString = my:name;
```

compositeFor(nameStr):role(roleStr)

Operator Type: Property [other Property type actions]
Operator Scope of Action: Composite [operators of similar scope]
Operator Purpose: Composite [other Composite operators]
Operator First Added: Baseline
Operator Last Altered: As at baseline

---

compositeFor(nameStr):role(roleStr)

returns a List-type list of paths of all notes in the composite containing note **nameStr**, but including only the note or notes with the designated role **roleStr**. For example, to set the color of all notes with the role "author":

```
$MyList = $Color(compositeFor("great books"):role("author"))="red";
```

This code can also use the ' my' designator to give a shorter form. These are functional equivalents:

```
$MyList = compositeFor(this):role("some role");
$MyList = my:role("some role");
```

compositeFor(nameStr):roles

Operator Type: Property [other Property type actions]
Operator Scope of Action: Composite [operators of similar scope]
Operator Purpose: Composite [other Composite operators]
Operator First Added: Baseline
Operator Last Altered: As at baseline

---

compositeFor(nameStr):roles

returns a Set-type list of roles that appear in the composite containing note **nameStr**.

This code can also use the ' my' designator to give a shorter form. These are functional equivalents:

```
$MySet = compositeFor(this):roles;
$MySet = my:roles;
```

compositeWithName(compositeNameStr)

Operator Type: Function [other Function type actions]
Operator Scope of Action: Composite [operators of similar scope]
Operator Purpose: Composite [other Composite operators]
Operator First Added: Baseline
Operator Last Altered: As at baseline

---

compositeWithName(compositeNameStr)

returns a List-type list of paths of all notes within the composite with the name **compositeNameStr**. If several composites have the same name, data for only one of those composites is returned (first match by $OutlineOrder).

```
$MyList = compositeWithName("Lecture 16");t
```

If working in the context of a note already in a composite, compositeFor(this) will also return an appropriate list of paths.

contains(item)

Operator Type: Function [other Function type actions]
Operator Scope of Action: Item [operators of similar scope]
Operator Purpose: Query Boolean [other Query Boolean operators]
Operator First Added: Baseline
Operator Last Altered: As at baseline
Operator Uses Regular Expressions: [More on regular expressions in Tinderbox]
Operator Has Newer Dot-Operator Variant: Yes

---

contains(item)

Returns Boolean `true` if the note evaluated from a single **item** is a direct child of the current note or, put conversely, if the note is the parent of **item**. Thus it can be thought of as an "is parent of" operator and, as such, a counterpart to the "i
a child of" operator inside().

The **item** argument must be quoted unless an attribute reference. Ways to define **item**.

Examples:

```
contains("/foo")
```

This is a full path so can match *only* the root-level note 'foo'. But:

```
contains("bar")
```

could match any note with the name 'bar' of which there may be more than one. If there is more that one, Tinderbox chooses one. This is fine if one match was desired but if there are known to be multiple 'bar' notes and it is desired to locate parent container of every one, then a different approach is needed, using any(): the process is described under any().

NOTE: Checking attribute values, as opposed to object containment

There are a range of other tools to check if a String-, List- or Set-type attribute's value contains a desired search string/regex.

- For String attributes there are String.contains() and String.icontains().
- List and Set date type attributes also support .contains() and .icontains() though in this context the scope of regex matches is slightly different from that with a String-type attribute (see the linked articles).
- Single words alone can be checked using word(), which works across $Name, $Text and all String-type attributes.

Legacy issues

This operator replaces the legacy #contains query operator. The latter should not be used for new code.

cos(radiansNum)

Operator Type: Function [other Function type actions]
Operator Scope of Action: Item [operators of similar scope]
Operator Purpose: Mathematical [other Mathematical operators]
Operator First Added: Baseline
Operator Last Altered: As at baseline

---

cos(radiansNum)

cos() converts its **radiansNum**, in *radians*, to the cosine of that value.

```
$MyNumber = cos(6);
```

returns 0.9601702867 for an input of 6 radians.

count_if(scope, condition)

Operator Type: Function [other Function type actions]
Operator Scope of Action: Group [operators of similar scope]
Operator Purpose: Data manipulation [other Data manipulation operators]
Operator First Added: Baseline
Operator Last Altered: As at baseline
Operator Uses Regular Expressions: [More on regular expressions in Tinderbox]
Operator Uses Scoped Arguments: [More on scoped arguments in Action Code]
Operator Has Conditional Arguments: [More on conditional operator arguments]
Operator Has Newer Dot-Operator Variant: Yes

---

count_if(scope, condition)

Counts the number of notes in the list derived from **scope** that satisfy the evaluated expression **condition**.

scope describes the notes to be examined (defining **scope**).

condition is action code forming a valid conditional query test, i.e. it equates to `true` when matched. Some query-style operators terms may allow use of regular expressions.

This equivalent to use of sum_if(), as the next two code examples have the same result:

```
sum_if(group, condition,1)
```

but this is perhaps more easily understood as:

```
count_if(group, condition)
```

count(scope)

Operator Type: Function [other Function type actions]
Operator Scope of Action: List [operators of similar scope]
Operator Purpose: Data manipulation [other Data manipulation operators]
Operator First Added: Baseline
Operator Last Altered: As at baseline
Operator Uses Scoped Arguments: [More on scoped arguments in Action Code]
Operator Has Newer Dot-Operator Variant: Yes

---

count(scope)

The function count() counts the Number of discrete items in the specified list, **scope**, which is defined as a group of items—i.e. one or more items.

Most often, **list** may be a reference to a List or Set typed data. The **list** argument is evaluated so can use more than literal lists, including offset addresses like $Attribute(note) or more complex expressions to get data as long as the result is a list-based attribute (List or Set data types).

NOTE: where **scope** is a known List or Set type attribute reference, it is recommended and generally easier, and recommended, to use List/Set.count (or alternatively the older less intuitive List/Set.size); both the latter give the same outcom

count(scope)

For example if $DisplayAttributes for the current note is [Color;AccentColor;NameFont] then the code

```
$MyNumber = count($DisplayAttributes);
```

is effectively

```
$MyNumber = count([Color;AccentColor;NameFont]);
```

and not surprisingly returns 3. Note that the count is not all unique values for the attribute across the whole TBX, scope is restricted to 'this' note or another nominated note. Specimen usage:

```
$MyNumber = count($DisplayAttributes);
$MyNumber = count($DisplayAttributes("some other note"));
```

To use count() with a list of items that are attributes or expressions, use list():

Works: `$MyNumber = count(list(4+2,9+6));` (output: 2)

For more complex examples, where list items are action code expressions, it may be necessary to use eval() to wrap each list item expression e.g. `list(eval(expressionA),eval(expressionB))`.

Examples

The following is a trivial example (given we could use $ChildCount instead) but shows how count can be used in a more subtle way:

```
$MyNumber = count(collect(children,$Name));
```

The result of collect is List-type data, in this case a number of note titles **count(list)** will return the number of values in the list (including duplicates). To get a de-duped count, chain the `.unique` operator to the list reference, inside the `count()` operator so the .unique filter is applied before the count is taken:

```
$MyNumber = count(collect(children,$Name).unique);
```

covid([stateStr, countryStr|zipCodeStr], aDate, keywordStr)

Operator Type: Function [other Function type actions]
Operator Scope of Action: Item [operators of similar scope]
Operator Purpose: Data manipulation [other Data manipulation operators]
Operator First Added: Baseline
Operator Last Altered: As at baseline
Operator Has Optional Arguments: [More on optional operator arguments]

---

covid(zipCodeStr, dateStr, keywordStr)

covid(stateStr, countryStr, aDate, keywordStr)

The operator covid() returns information about the 2020 pandemic of the COVID-19 virus.

covid(zipCodeStr, aDate, keywordStr)

Returns the number of cases, deaths, and recoveries reported that day in the US country that contains this zip code. For example

```
$MyNumber = covid("02148", date(2020,1,4), "cases");
```

This returns the number of cases reported in Middlesex County, Massachusetts for 4 January 2020. The **keywordStr** argument may be a quoted string with any of the following values:

- cases
- deaths
- recoveries
- name (the name of the county)

covid(stateStr, countryStr, aDate, keywordStr)

A four-argument variant allows you to query results by US state:

```
$MyNumber = covid("MA","US",date(2020,1,4),"recoveries");
```

This returns the number of recoveries reported for the date 4 January 2020, from Massachusetts.

Data Source

Data are as reported by the Johns Hopkins Center For Systems Science and Engineering, and are provided by CovidNearMe.org and are provided strictly for educational and academic research purposes. *Please note*: data are copyright 202 Johns Hopkins University.

create([containerStr,]nameStr)

Operator Type:	Function [other Function type actions]
Operator Scope of Action:	Item [operators of similar scope]
Operator Purpose:	Document configuration [other Document configuration operators]
Operator First Added:	Baseline
Operator Last Altered:	As at baseline
Operator Has Optional Arguments:	[More on optional operator arguments]

---

create(name)

This creates a new note called **nameStr** at the designated location, and returns the full path to that note. If the designated note already exists, no new note is created and the operator returns the empty string.

The **create()** operator always returns the path to the new (or pre-existing) note.

This function needs no left-side expression argument, i.e. '$SomeAttribute=', to invoke it.

nameStr is typically actually a complete path:

```
create("/hardware/taps");
```

but if **nameStr** is a unique note $Name, a new note is created as a new (last) child of the current note. For example:

```
create("taps");
```

create(containerStr, nameStr)

A two-argument variant is also offered that allows the container for a new item to be specified, and the new item's $Name. This may be useful if you need to create several notes in the same container, for example is iterating a list with .eacl For example:

```
create("/hardware","taps");
```

Or, more pertinently, using a loop variable 'aPlace':

```
$SomeList.each(aPlace){
 create(aPlace,"urgentTasks");
};
```

Essentially, the two-input form allows 3 forms of variation:

- different path, different name (via two nested loops—one for paths, one for names)
- different path, same name (loop with path variants)
- same path, different name (loop with name variants)

Designators and evaluation

Paths like that below, which mix literal and computed values are not evaluated, nor are inline designators:

```
create(/Resources/Test/$MyString)
```

Instead use code like this:

```
var path="/Resources/Test/"+$MyString;
create(path);
```

Testing for pre-existing notes

Although **create()** will not re-create a note that already exists, a scenario in a big/complex document occurs where it may be necessary to run a number of action *once only* on newly created notes. How to test for a note already existing? He is one possible solution:

```
var:string vTestPath = "/foo/bar/baz";
var:string vTest = $IDString(vTestPath);
if(vTest==""){
 $Text = create(vTestPath);
 // do tasks only needed once, for new notes
};
```

A non-existent note cannot have an $IDString value. So by testing the `$IDString(path)` for the **path** of the note to be created *before* calling **create(path)**, it is possible to ensure **create()** is only called if needed and in context all the code desired to be run, *once*, at new note creation.

createAdornment([containerStr,] nameStr)

Operator Type:	Function [other Function type actions]
Operator Scope of Action:	Item [operators of similar scope]
Operator Purpose:	Document configuration [other Document configuration operators]
Operator First Added:	Baseline
Operator Last Altered:	As at baseline
Operator Has Optional Arguments:	[More on optional operator arguments]

---

createAdornment([containerStr,] nameStr)

New to v9.5.0, the operator **createAdornment()**, creates an adornment at the designated path. If an adornment already exists with that path, no new adornment is created.

createAdornment(nameStr)

This creates a new adornment called **nameStr** at the designated location, and returns the full path to that adornment. If the designated adornment already exists, no new adornment is created and the operator returns the empty string.

The createAdornment() operator always returns the path to the new (or pre-existing) adornment. Previously, if the adornment already existed, the operator returned `false`.

This function needs no left-side expression argument, i.e. '$SomeAttribute=', to invoke it.

nameStr is typically actually a complete path:

```
create("/hardware/taps");
```

but if **nameStr** is a unique note $Name, a new adornment is created as a *child* of the current note. For example:

```
createAdornment("taps");
```

createAdornment(containerStr, nameStr)

A two-argument variant is also offered that allows the container for a new item to be specified, and the new item's $Name. This may be useful if you need to create several adornments in the same container, for example is iterating a list with .each(). For example:

```
createAdornment("/hardware","taps");
```

Or, more pertinently, using a loop variable 'aPlace':

```
$SomeList.each(aPlace){
 createAdornment(aPlace,"urgentTasks");
};
```

Essentially, the two-input form allows 3 forms of variation:

- different path, different name (via two nested loops—one for paths, one for names)
- different path, same name (loop with path variants)
- same path, different name (loop with name variants)

The **createAdornment()** operator evaluates its first argument, permitting use of an expression to compute a value. Note that this means that paths should be quoted: `createAdornment("/Containers/People/Mark")`, as otherwise parts of paths may be evaluated as expressions.

Designators and evaluation

Paths like that below, which mix literal and computed values are not evaluated, nor are inline designators:

```
createAdornment(/Resources/Test/$MyString)
```

Instead use code like this:

```
var path="/Resources/Test/"+$MyString;
createAdornment(path);
```

createAgent([containerStr,]nameStr)

Operator Type:	Function [other Function type actions]
Operator Scope of Action:	Item [operators of similar scope]
Operator Purpose:	Document configuration [other Document configuration operators]
Operator First Added:	Baseline
Operator Last Altered:	As at baseline
Operator Has Optional Arguments:	[More on optional operator arguments]

---

createAgent(nameStr)

This creates a new agent called **nameStr** at the designated location, and returns the full path to that agent. If the designated agent already exists, no new agent is created and the operator returns the empty string.

The **create()** operator *always* returns the path to the new (or pre-existing) agent.

This function needs no left-side expression argument, i.e. '$SomeAttribute=', to invoke it.

nameStr is typically actually a complete path:

```
createAgent("/agents/urgentTasks");
```

but if **name** is a unique agent $Name, a new agent is created as a *child* of the current note. Be aware though that cannot be used if the current object is an agent rather than a note. For example:

```
create("urgentTasks");
```

create(containerStr, nameStr)

A two-argument variant is also offered that allows the container for a new item to be specified, and the new item's $Name. This may be useful if you need to create several agents in the same container, for example is iterating a list with .ea
For example:

```
create("/agents","urgentTasks");
```

Or, more pertinently, using a loop variable 'aPlace':

```
$SomeList.each(aPlace){
 create(aPlace,"urgentTasks");
};
```

Essentially, the two-input form allows 3 forms of variation:

- different path, different name (via two nested loops—one for paths, one for names)
- different path, same name (loop with path variants)
- same path, different name (loop with name variants)

Designators and evaluation

Paths like that below, which mix literal and computed values are not evaluated, nor are inline designators:

```
createAgent("Resources/Test/$MyString)
```

Instead use code like this:

```
var:string vPath="/Resources/Test/"+$MyString;
createAgent(vPath);
```

createAttribute(nameStr[, dataType])

Operator Type:	Function [other Function type actions]
Operator Scope of Action:	Item [operators of similar scope]
Operator Purpose:	Document configuration [other Document configuration operators]
Operator First Added:	Baseline
Operator Last Altered:	As at baseline
Operator Has Optional Arguments:	[More on optional operator arguments]

---

createAttribute(nameStr[, dataType])

The operator createAttribute(name[, type]) can create a new user attributes. If an attribute of the same **nameStr** already exists, the operator has no effect and returns `false`. Otherwise, a new user attribute is created. The nameStr argument case-sensitive and should use the capitalisation as for the desired attribute. Thus **nameStr** values of `Test` and `test` result in different attributes called 'Test' and 'test'.

Optionally, a **dataType** argument may be supplied to determine the data type of the new attribute. Recognised values for **dataType** include: `string`, `number`, `boolean`, `date`, `color`, `interval`, `file`, `list`, `set`, `url`, and `dictionary`. no **dataType** valued is supplied, Tinderbox creates a String-type attribute. The **dataType** argument is *case-sensitive*: `number` produces a number type attribute, but `Number` would result in a (default) *string* type attribute.

A basic example using just the **nameStr** argument, results in a new String-type attribute called 'MyList':

```
$MyBoolean = createAttribute("SomeList");
```

But, in the example above, the chosen name for the new attribute indicates the intended data-type should be of List data type. Therefore, use of the optional **dataType** argument would make more sense to signal user intent to Tinderbox mo clearly:

```
$MyBoolean = createAttribute("SomeList", "list");
```

Again, the action results in a new String-type attribute called 'MyList'.

Note that action code cannot alter the new attribute's name or type, nor delete the attribute. This can be done but only via the User attribute inspector.

Some other aspects of the attribute, e.g. default value) can be changed after the attribute is created, using attribute(). Here below, a number-type attribute is created, its default is set and then a value is applied:

```
createAttribute("Tester","number");
attribute("Tester")["default"] = 10;
$Tester = 1000;
```

createLink(sourceItem, destinationItem[, linkTypeStr])

Operator Type:	Function [other Function type actions]
Operator Scope of Action:	Item [operators of similar scope]
Operator Purpose:	Linking [other Linking operators]
Operator First Added:	Baseline
Operator Last Altered:	As at baseline
Operator Uses Scoped Arguments:	[More on scoped arguments in Action Code]
Operator Has Optional Arguments:	[More on optional operator arguments]

---

createLink(sourceItem,destinationItem[, linkTypeStr])

The operator **createLink()** can be used for link creation. This is useful when neither the source nor the destination of the link are this note. The arguments **sourceItem** and **destinationItem** must be defined so as to identify a single (existing) item.

Example:

```
$MyList2.each(anItem2){
 $MyList.each(anItem1){
  createLink(anItem1,anItem2);
 };
};
```

createLink() may also cope better than linkTo() and linkFrom() with the edge case when a note's $Name (or those of other notes in the note's path) contain forward slashes. Such paths are challenge as in string form the differing meaning of the use of slashes (container delimiter vs. note title) is ambiguous to the parser.

Optionally, a link type for the new link may be specified suing **linkTypeStr**. If that link type does not exist, it is created.

Date.day()

Operator Type: Property [other Property type actions]
Operator Scope of Action: Item [operators of similar scope]
Operator Purpose: Date-time [other Date-time operators]
Operator First Added: Baseline
Operator Last Altered: As at baseline

Syntax note: *Operators without any defined mandatory arguments may omit their empty closing parentheses*

Date.day()
Date.day
Returns the 1 or 2 digit day for the Date-type attribute specified. Valid range 1-31.
If $MyDate is 25 Oct 1415 16:20 then:

```
$MyNumber = $MyDate.day;
```
sets $MyNumber to 25
The operator may also be used to set the attribute's day, using a valid figure

```
$MyDate.day = 20;
```
sets the day of $MyDate to 20th of the month

Date.format(formatStr)

Operator Type: Function [other Function type actions]
Operator Scope of Action: Item [operators of similar scope]
Operator Purpose: Formatting [other Formatting operators]
Operator First Added: Baseline
Operator Last Altered: As at baseline

Date.format(formatStr)
Returns **Date** as a String, formatted as per the quoted date format string **formatStr**.
This supplements the existing format() function.
For example:

```
$MyString = $MyDate.format("L")
```
gets the note's creation date and formats it as a "long local date" such as "Sunday, 23 March, 2007" and sets that as the value of $MyString.
If **data** is a date, the format string is the same as the format used by Tinderbox's date format codes.

Date.hour()

Operator Type: Property [other Property type actions]
Operator Scope of Action: Item [operators of similar scope]
Operator Purpose: Date-time [other Date-time operators]
Operator First Added: Baseline
Operator Last Altered: As at baseline

Syntax note: *Operators without any defined mandatory arguments may omit their empty closing parentheses*

Date.hour()
Date.hour
Returns the 1 or 2 digit hour for the Date-type attribute specified. Valid range 0-23.
If $MyDate is 25 Oct 1415 16:20 then:

```
$MyNumber = $MyDate.hour;
```
returns 16
The operator may also be used to set the attribute's hour, using a valid figure

```
$MyDate.hour = 20;
```
sets the hour to 20 (8 PM)

Date.minute()

Operator Type: Property [other Property type actions]
Operator Scope of Action: Item [operators of similar scope]
Operator Purpose: Date-time [other Date-time operators]
Operator First Added: Baseline
Operator Last Altered: As at baseline

Syntax note: *Operators without any defined mandatory arguments may omit their empty closing parentheses*

Date.minute()
Date.minute
Returns the 1 or 2 digit minute for the Date-type attribute specified. Valid range 0-59.
If $MyDate is 25 Oct 1415 15:20 then:

```
$MyNumber = $MyDate.minute;
```
returns 20
The operator may also be used to set the attribute's minute, using a valid figure

```
$MyDate.minute = 36;
```
sets the minute to 36th of the hour

Date.month()

Operator Type: Property [other Property type actions]
Operator Scope of Action: Item [operators of similar scope]
Operator Purpose: Date-time [other Date-time operators]
Operator First Added: Baseline
Operator Last Altered: As at baseline

Syntax note: *Operators without any defined mandatory arguments may omit their empty closing parentheses*

Date.month()
Date.month
Returns the 1 or 2 digit month for the Date-type attribute specified. Valid range 1-12.
If $MyDate is 25 Oct 1415 15:20 then:

```
$MyNumber = $MyDate.month
```
returns 10
The operator may also be used to set the attribute's month, using a valid figure.

```
$MyDate.month = 6;
```
sets the month to 6 (June)

Date.second()

Operator Type: Property [other Property type actions]
Operator Scope of Action: Item [operators of similar scope]
Operator Purpose: Date-time [other Date-time operators]
Operator First Added: Baseline
Operator Last Altered: As at baseline

Syntax note: *Operators without any defined mandatory arguments may omit their empty closing parentheses*

Date.second()
Date.second
Returns the 1 or 2 digit seconds for the Date-type attribute specified. Valid range 0-59.
If $MyDate is 25 Oct 1415 15:20:06 then:
`$MyNumber = $MyDate.second` returns 6
The operator may also be used to set the attribute's month, using a valid figure
`$MyDate.second = 24;` sets the seconds element of the time to 24

Date.week()

Operator Type: Property [other Property type actions]
Operator Scope of Action: Item [operators of similar scope]
Operator Purpose: Date-time [other Date-time operators]
Operator First Added: Baseline
Operator Last Altered: As at baseline

Syntax note: *Operators without any defined mandatory arguments may omit their empty closing parentheses*

Date.week()
Date.week
The Date operator .week() returns the number of the week in the current year. For example, January 1 is in week 1.
Be aware that to allow for the fact the calendar year and week start are only aligned every seventh year, in some years days that the very end of the year may report as week #53.
Date.week() is read-only.
`$MyNumber = $MyDate.week();`
Thus is $MyDate is 4 January, $MyNumber would be 1.

Date.weekday()

Operator Type: Property [other Property type actions]
Operator Scope of Action: Item [operators of similar scope]
Operator Purpose: Date-time [other Date-time operators]
Operator First Added: Baseline
Operator Last Altered: As at baseline

Syntax note: *Operators without any defined mandatory arguments may omit their empty closing parentheses*

Date.weekday()
Date.weekday
Returns the day number for the Date-type attribute specified. Value range is 1-7. Days number from 1 (Monday) to 7 (Sunday). This numbering allows easy testing for weekday (#1-5) vs. weekend (#6-7).
This operator is read-only and cannot be used to change a Date attributes value.
`$MyNumber = $MyDate.weekday;`
Thus if the day of $MyDate is Tuesday, $MyNumber would be 2.

Date.year()

Operator Type: Property [other Property type actions]
Operator Scope of Action: Item [operators of similar scope]
Operator Purpose: Date-time [other Date-time operators]
Operator First Added: Baseline
Operator Last Altered: As at baseline

Syntax note: *Operators without any defined mandatory arguments may omit their empty closing parentheses*

Date.year()
Date.year
Returns the 1 to 4 digit year for the Date-type attribute specified.
If $MyDate is 25 Oct 1415 16:20 then:
`$MyNumber = $MyDate.year;` returns 1415
The operator may also be used to set the attribute's day, using a valid figure
`$MyDate.year = 1805;` sets the year to 1805

date(dateStr)

Operator Type: Function [other Function type actions]
Operator Scope of Action: Item [operators of similar scope]
Operator Purpose: Date-time [other Date-time operators]
Operator First Added: Baseline
Operator Last Altered: As at baseline

date(dateStr)
constructs a Date from a quoted string or string expression. Usually, this is not necessary; Tinderbox will coerce the **dateStr** to a date type automatically. In some contexts though, it may be more convenient or more clear to make the conversion explicit. (See format() and .format() to convert dates into strings).
This operator creates actual Date-type data, which the examples below are actually coercing back into a string Note too that the examples below were output from a system with UK setting, thus using day-month order. For reference, month-day order is actually only used in the USA and the Philippines whilst the rest of the world uses day-month order, though the delimiters used in text can vary (slash, colon, dot, etc.). The action code shown is being exported via ^value()^.
Note that seconds are ignored (from v5 onwards); if provided they are not used and are coerced to '00' instead.
Default: `$MyDate = date("24/10/2009");` sets $MyDate to "24 Oct 2009 at 13:30:06"; note how the hh:mm current at export get added. The resulting string is the same as you see for a date displayed as a Displayed Attribute. The exact format will depend on the users international settings.
Provide a time: `$MyDate = date("24/10/2009 01:30:22")` gives $MyDate a value of "24 Oct 2009 at 01:30:00"; the specified time gets used.
But, if you a date formatting string and you get a formatted *string* of the date:
`$MyString = date("24/10/2009 01:30:00").format("*");` gives $MyDate a value of "Sat, 24 Oct 2009 01:30:00 +0100
or:
`$MyString = date("24/10/2009 01:30:00").format("*");` gives $MyDate a value of "Sat, 24 Oct 2009 01:30:00 +0100"
The two methods are equivalent, note also the change of format due to use of a date format string. In the first example above note how the format() operator wraps date() and not vice versa.
An attribute can also provide part of the input:
`$EndDate = date($StartDate+"7 days");`
With care this can be extended. In the following, $MyString is "7 days" and $MyNumber is 7. The outcome is that $MyDateA/B/C are all set to the same date:

```
$MyDateA = date($MyDate+"7 days");
$MyDateB = date($MyDate+$MyString);
$MyDateC = date($MyDate+($MyNumber+" days")); note the extra parentheses are optional but suggested
```

date(yearNum, monthNum, dayNum[, hourNum, minNum])

Operator Type: Function [other Function type actions]
Operator Scope of Action: Item [operators of similar scope]
Operator Purpose: Date-time [other Date-time operators]
Operator First Added: Baseline
Operator Last Altered: As at baseline
Operator Has Optional Arguments: [More on optional operator arguments]

---

date(yearNum, monthNum, dayNum[, hourNum, minNum])

constructs a Date from individual *numeric* elements. This is useful, for example, if you need to assemble a date from separate attributes.

year is the 4-digit year

month is a number from 1–12

day is a number from 1 to 31

The time arguments are optional, and are specified in a 24-hour clock.

hour is a number from 0 to 23

minute is a number from 0 to 59

Using this operator, do not quote the whole argument list: these should be left as individual numbers.

Wrong: `$MyDate = date("2004,7,23,16,45");`

Right: `$MyDate = date(2004,7,23,16,45);`

Examples:

`$MyDate = date(2004,7,23,16,45);` …sets 23 July 2004 16:45

day(aDate[, dayNum])

Operator Type: Function [other Function type actions]
Operator Scope of Action: Item [operators of similar scope]
Operator Purpose: Date-time [other Date-time operators]
Operator First Added: Baseline
Operator Last Altered: As at baseline
Operator Has Optional Arguments: [More on optional operator arguments]

---

day(aDate[, dayNum])

Alternatively, use Date.day.

day(aDate)

returns, as a Number, the day of the month from the **aDate** expression, which may simply be a date-type attribute value.

```
$MyNumber = day($MyDate);
```

day(theDate, dayNum)

creates a new date based on the **aDate** expression, but in which the day of the month is **dayNumDate** is not changed unless theDate is an attribute and the attribute is re-setting itself:

`$MyDateA = day($MyDate,14);` $MyDate is unaltered

`$MyDate = day($MyDate,14);` $MyDate is changed

Examples. If $MyDate is July 4, 2009 then

`$MyNumber = day($MyDate);` …is 4

If $MyDate is July 4,2009, then

```
$MyDate = day($MyDate,5);
```

will change $MyDate to July 5, 2009.

days(firstDate, lastDate)

Operator Type: Function [other Function type actions]
Operator Scope of Action: Item [operators of similar scope]
Operator Purpose: Date-time [other Date-time operators]
Operator First Added: Baseline
Operator Last Altered: As at baseline

---

days(firstDate, lastDate)

returns the Number of 'days' (as defined below) that elapsed between **firstDate** and **lastDate**. If **lastDate** is earlier than **firstDate** then the result is negative.

The operator returns the number of 24-hour blocks [sic] between two dates, rounded toward zero. So if days() measures the difference between a date/time of 09:30 today and 08:30 tomorrow, the result is 0 (zero) as the difference is only 23 hours. If the times are same a whole day increment is recorded.

Thus days does not return a simplistic calendar day difference as some users might intuit it would. If times vary between **firstDate** and **lastDate**, the returned difference figure may thus be one day high or low of an expected calendar day-based value.

If $DateA is 3 January 2016 and $DateB is 9 January 2016, then:

```
$MyNumber = days($DateA,$DateB);
```

sets $MyNumber to 6.

Also see minutes(date1,date2).

degrees(radiansNum)

Operator Type: Function [other Function type actions]
Operator Scope of Action: Item [operators of similar scope]
Operator Purpose: Mathematical [other Mathematical operators]
Operator First Added: Baseline
Operator Last Altered: As at baseline

---

degrees(radiansNum)

Takes an angle argument, **radiansNum**, specified in radians and return the angle in degrees.

```
$MyNumber = degrees(6);
```

returns 343.7746771 for an input of 6 radians.

See also radians() which converts an angle in degrees to radians.

delete(scope)

Operator Type: Function [other Function type actions]
Operator Scope of Action: Group [operators of similar scope]
Operator Purpose: Data manipulation [other Data manipulation operators]
Operator First Added: Baseline
Operator Last Altered: As at baseline
Operator Uses Scoped Arguments: [More on scoped arguments in Action Code]

---

delete(scope)

Deletes the note(s) designated by **scope**; defining **scope**. If a designated note does not exist, the expression has no effect and returns `false`, i.e. nothing happens: no deletion, no message of no deletion. If the designated note exists, it will be deleted and the expression returns its former path.

```
delete("Some note");
```

However, it is strongly suggesting using a path ($Path) for **scope** and not just the note's title ($Name), thus:

```
delete("/The/path/to/Some note");
```

Avoid using this operator when possible. It can automatically delete notes you intended to create, and it can potentially saw off the branch you are standing on. In almost all circumstances, it is better to move the unwanted note to a container, and then to delete the note manually if you really need to delete it at all.

The delete() operator returns `true` if at least one note was deleted, and `false` otherwise.

descendedFrom(item)

Operator Type: Function [other Function type actions]
Operator Scope of Action: Item [operators of similar scope]
Operator Purpose: Query Boolean [other Query Boolean operators]
Operator First Added: Baseline
Operator Last Altered: As at baseline

---

descendedFrom(item)

Returns Boolean `true` if **item** is an an ancestor of the current note, i.e. it matches *all* descendants of **item** however deep the outline branch beneath it.

The **item** argument must be quoted unless an attribute reference. Ways to define **item**.

To return a Set of all items descended from container at path '/foo/bar' use the query:

```
descendedFrom(/foo/bar)
```

Or if 'bar' is a unique title in the document, $Name alone can be used:

```
descendedFrom("bar")
```

the latter being the form most usually encountered.

Unlike the paring of contains() and inside(), there is no converse operator that looks for notes that are ancestors of **item**.

Legacy issues

This operator replaces the legacy #descendedFrom query operator.

Dictionary.add(itemDict)

Operator Type: Function [other Function type actions]
Operator Scope of Action: Item [operators of similar scope]
Operator Purpose: Dictionary, Set & List operations [other Dictionary, Set & List operations operators]
Operator First Added: Baseline
Operator Last Altered: 9.5.0

Dictionary.add(itemDict)

This reads a Dictionary-type argument **itemDict** from which a **key** and a **value** are parsed.

```
$MyDictionary = $MyDictionary.add({apple:green});
```

Note that quotes are not needed around the **key** and **value**.

If **key** does not exist, that key is created with a value of **value**.

If **key** exists, **key** is given value **value**. This replaces any/all existing values for this key.

Assume $MyDictionary has no 'apple' **key**. Example:

```
$MyDictionary = $MyDictionary.add({apple:fruit});
```

The **key** 'apple' is added and now has **value** 'fruit'

```
$MyDictionary = $MyDictionary.add({apple:green});
```

The **key** 'apple' now has a new value 'green'. Now assume the 'apple' key has multiple values of 'fruit;green;red':

```
$MyDictionary = $MyDictionary.add({apple:pie});
```

Now the **value** is just 'pie' because an .add() operator *replaces* all existing **value**(s).

This operator is also equivalent to:

```
Dictionary["key"] = "value";
```

To add an *additional* value(s) to existing value(s), see Dictionary.extend().

To remove a key—and any value(s) it has—from the Dictionary, there is no operator but instead the key string is deleted using a minus operator: see '**Deleting key:value pairs**' here.

As from v9.5.2, the .add() operator accepts quoted strings. The following expressions are equivalent:

```
$MyDictionary.add({1:able})
$MyDictionary.add("{1:able}")
```

But do not use either of the following example syntax:

```
$MyDictionary.add({"1:able"})
```
WRONG!

```
$MyDictionary.add({"1":"able"})
```
WRONG!

Using offset addresses within itemDict

Either the **keyStr** or the **valueStr** may need to be calculated variable, for instance the valueStr might need to be the value of `$MyString("Some note")` or in a loop, `$MyString(loopVar)`. These cannot be resolved within the .add() operator but **itemDict** can be a variable. Thus, in a loop, rather than:

```
vDict = vDict.add({$Name(aState)+":"+$Color(aState)});
```
WRONG!

use:

```
vList.each(aState){
  // Dict.add() can't resolve attribute offset value
  // so store the complete string in a string
  // variable and pass a single var argument to .add()
  var:string vPair = $Name(aState)+":"+$Color(aState);
  vDict = vDict.add({vPair});
};
```

Legacy form (pre-v9.5.0)

Dictionary.add(keyStr, valueStr)

This sets a **keyStr** to the **valueStr**.

If **keyStr** does not exist, that key is created with a value of **valueStr**.

If **keyStr** exists, **keyStr** is given value **valueStr**. This replaces any/all existing values for this key.

Assume $MyDictionary has no 'apple' **keyStr**. Example:

```
$MyDictionary = $MyDictionary.add("apple","fruit");
```

The **keyStr** 'apple' is added and now has **valueStr** 'fruit'

```
$MyDictionary = $MyDictionary.add("apple","green");
```

The **keyStr** 'apple' now has a new value 'green'. Now assume the 'apple' key has multiple values of 'fruit;green;red':

```
$MyDictionary = $MyDictionary.add("apple","pie");
```

Now the **value** is just 'pie' because an .add() operator *replaces* all existing **valueStr**(s).

Dictionary.contains(keyStr)

Operator Type: Function [other Function type actions]
Operator Scope of Action: Item [operators of similar scope]
Operator Purpose: Query Boolean [other Query Boolean operators]
Operator First Added: Baseline
Operator Last Altered: As at baseline

Dictionary.contains(keyStr)

The expression:

```
$MyDictionary.contains(key)
```

is `true` if the dictionary contains the designated key.

Note that **keyStr** is literal string and not a regex pattern (unlike the operator's use chained with some other data types).

Dictionaries are case-sensitive, but Dictionary.icontains(*key*) is available for case-insensitive matching.

Dictionary.count()

Operator Type: Property [other Property type actions]
Operator Scope of Action: Item [operators of similar scope]
Operator Purpose: Dictionary, Set & List operations [other Dictionary, Set & List operations operators]
Operator First Added: Baseline
Operator Last Altered: As at baseline

Syntax note: *Operators without any defined mandatory arguments may omit their empty closing parentheses*

Dictionary.count()

Dictionary.count

A Number-type property. Returns the number of keys in the dictionary (by their nature all keys are discrete so there is not potential of duplication in the count). If:

```
$MyDictionary = {cat:animal; dog:animal; rock: mineral};
$MyNumber = $MyDictionary.count;
```

MyNumber is now 3.

Dictionary.count and Dictionary.size are interchangeable. Use which ever seems more intuitive.

Dictionary.empty()

Operator Type: Property [other Property type actions]
Operator Scope of Action: Item [operators of similar scope]
Operator Purpose: Dictionary, Set & List operations [other Dictionary, Set & List operations operators]
Operator First Added: Baseline
Operator Last Altered: As at baseline

Syntax note: *Operators without any defined mandatory arguments may omit their empty closing parentheses*

Dictionary.empty()

Dictionary.empty

A Boolean-type property. Returns `true` if the dictionary has no keys, and is `false` otherwise.

```
$MyDictionary = {cat:animal; dog:animal; rock: mineral};
$MyBoolean = $MyDictionary.empty;
```

MyBoolean is now `false`

```
$MyDictionary = ; (or $MyDictionary = {};)
$MyBoolean = $MyDictionary.empty;
```

MyBoolean is now `true`

Dictionary.extend(itemDict)

Operator Type: Function [other Function type actions]
Operator Scope of Action: Item [operators of similar scope]
Operator Purpose: Dictionary, Set & List operations [other Dictionary, Set & List operations operators]
Operator First Added: Baseline
Operator Last Altered: 9.5.0, 9.5.2

Dictionary.extend(itemDict)

This reads a Dictionary-type argument **itemDict** from which a **key** and a **value** are parsed. It adds the **value** string to the value(s) of a **key**.

If **key** does not exist, that key is created with a value of **value**.

If **key** exists, **value** is appended to **key**'s existing value(s).

Assume $MyDictionary has no 'pear' **key**. Example:

```
$MyDictionary = $MyDictionary.extend({pear:fruit});
```

The **key** 'pear' is added and now has **value** 'fruit'. Next:

```
$MyDictionary = $MyDictionary.extend({pear:green});
```

The **key** 'pear' now has an additional new **value** 'green', but the overall value is now a list and the key:value pair is `pear:fruit;green`.

To set the new **valueStr** so it *replaces* all existing value(s) see Dictionary.add().

Note that quotes are not used around either/both the **keyStr** and **valueStr**. As from v9.5.2, the .extend() operator accepts quoted strings. The following expressions are equivalent:

```
$MyDictionary.extend({1:able}}
$MyDictionary.extend("{1:able}"}
```

But do not use either of the following example syntax:

```
$MyDictionary.extend({"1:able"}} WRONG!
$MyDictionary.extend({"1":"able"}} WRONG!
```

Dictionary.extend(keyStr, valueStr)

This adds the **valueStr** string to the value(s) of a **keyStr**.

If **keyStr** does not exist, that key is created with a value of **valueStr**.

If **keyStr** exists, **valueStr** is appended to **keyStr**'s existing value(s).

Assume $MyDictionary has no 'pear' **keyStr**. Example:

```
$MyDictionary = $MyDictionary.extend("pear","fruit");
```

The **keyStr** 'pear' is added and now has **valueStr** 'fruit'. Next:

```
$MyDictionary = $MyDictionary.extend("pear","green");
```

The **keyStr** 'pear' now has an additional new **valueStr** 'green', but the overall value is now a list and the key:value air is `pear:fruit;green`.

To set the new **valueStr** so it *replaces* all existing value(s) see Dictionary.add().

Dictionary.extend(dictStr)

From v9.5.0, the **Dictionary.extend()** operator takes a single argument, a dictionary—using the new {} syntax of key:pair elements which will extend the current elements.

```
$MyDictionary = $MyDictionary.extend({pear:green});
```

Note that quotes are not used around either/both the **keyStr** and **valueStr**. As from v9.5.2, the .extend() operator accepts quoted strings. The following expressions are equivalent:

```
$MyDictionary.extend({1:able}}
```

`$MyDictionary.extend("{1:able}")`

But do not use either of the following example syntax:

`$MyDictionary.extend({"1:able"})` WRONG!

`$MyDictionary.extend({"1":"able"})` WRONG!

Using offset addresses within itemDict

Either the **keyStr** or the **valueStr** may need to be calculated variable, for instance the valueStr might need to be the value of `$MyString("Some note")` or in a loop, `$MyString(loopVar)`. These cannot be resolved within the .add() operator but **itemDict** can be a variable. Thus, in a loop, rather than:

~~`vDict = vDict.extend({$Name(aState)+":"+$Color(aState)});`~~ WRONG!

use:

```
vList.each(aState){
 // Dict.extend() can't resolve attribute offset value
 // so store the complete string in a string
 // variable and pass a single var argument to .extend()
 var:string vPair = $Name(aState)+":"+$Color(aState);
 vDict = vDict.extend({vPair});
};
```

Dictionary.icontains(keyStr)

Operator Type: Function [other Function type actions]
Operator Scope of Action: Item [operators of similar scope]
Operator Purpose: Query Boolean [other Query Boolean operators]
Operator First Added: Baseline
Operator Last Altered: As at baseline

Dictionary.icontains(keyStr)

The expression:

`$MyDictionary.icontains(key)`

is `true` if the dictionary contains the designated key or case variations of it.

Note that **keyStr** is literal string and not a regex pattern (unlike the operator's use chained with some other data types).

Dictionaries are case-sensitive, and Dictionary.contains(*key*) is available for case-sensitive matching.

Dictionary.keys()

Operator Type: Property [other Property type actions]
Operator Scope of Action: Item [operators of similar scope]
Operator Purpose: Dictionary, Set & List operations [other Dictionary, Set & List operations operators]
Operator First Added: Baseline
Operator Last Altered: As at baseline

Syntax note: *Operators without any defined mandatory arguments may omit their empty closing parentheses*

Dictionary.keys()

Dictionary.keys

A List-type property. Returns a list of unsorted key names (i.e. only the **keys**' strings and not those of **value**s) for this note. Unlike a List or Set, the Dictionary cannot be iterated directly using .each(). Instead, Dictionary.keys provides an iterable list. So:

```
$MyDictionary.keys.each(x){
 if($MyDictionary[x] == "Whoops"){
   ...
 }
}
```

Therefore do not try: ~~`$MyDictionary.each(x)`~~

A dictionary cannot be sorted but as above Dictionary.keys can be sorted. This is the previous example above, but with an added .sort operator inserted after .keys:

```
$MyDictionary.keys.sort.each(x){
 if($MyDictionary[x] == "Whoops"){
   ...
 }
}
```

Note that when iterating a dictionary with .each(X) the loop variable X is the String value of the key.

Depending on your needs you might prefer to pass the dictionary explicitly to a list attribute before doing further work:

`$MyList = $MyDictionary.keys; // then work with MyList`

Dictionary.size()

Operator Type: Property [other Property type actions]
Operator Scope of Action: Item [operators of similar scope]
Operator Purpose: Dictionary, Set & List operations [other Dictionary, Set & List operations operators]
Operator First Added: Baseline
Operator Last Altered: As at baseline

Syntax note: *Operators without any defined mandatory arguments may omit their empty closing parentheses*

Dictionary.size()

Dictionary.size

A Number-type property. Returns the number of keys in the dictionary (by their nature all keys are discrete so there is not potential of duplication in the count). If:

```
$MyDictionary = {cat:animal; dog:animal; rock: mineral};
$MyNumber = $MyDictionary.size;
```

MyNumber is now 3.

Dictionary.size and Dictionary.count are interchangeable. Use which ever seems more intuitive.

dictionary(dictionaryStr)

Operator Type: Function [other Function type actions]
Operator Scope of Action: Document [operators of similar scope]
Operator Purpose: Dictionary, Set & List operations [other Dictionary, Set & List operations operators]
Operator First Added: Baseline
Operator Last Altered: As at baseline

---

dictionary(dictionaryStr)

This operator constructs a new dictionary from a **dictionaryStr**. The **dictionaryStr** must contain pairs of keys and values separated by a colon; each key/value pair is separated by a semicolon. For example:

```
$MyDictionary=dictionary("cat:animal; dog:animal; rock: mineral");
```

The **key** "cat" has the **value** "animal", while the **key** "rock" has the **value** "mineral".

Normally the output will be passed to a Dictionary attribute, but if passed to an action code variable the latter should function as if a dictionary.

The operator's input string will be in the form of semi-colon delimited set of key:value pairs, i.e.

```
$MyDictionary = dictionary("cat:animal; dog:animal; rock:mineral");
```

Or

```
$MyString = "cat:animal; dog:animal; rock:mineral";
$MyDictionary = dictionary($MyString);
```

dictionary({dictionaryStr})

Using the newer {}-defined declaratory method for dictionary data:

```
$MyDictionary = dictionary({cat:animal; dog:animal; rock:mineral});
```

Or

```
$MyString = "cat:animal; dog:animal; rock:mineral";
$MyDictionary = dictionary({$MyString});
```

Here the braces (curly brackets) replace the need for quotes enclosing strings.

Direct creation from a string (deprecated)

It is possible that a dictionary can be made by passing an appropriately structured string to a Dictionary type attribute, thus:

```
$MyDictionary = "cat:animal; dog:animal; rock:mineral";
```

This latter usage is deprecated and the more explicit **dictionary()** operator should be used in its place.

Dictionary[keyStr]

Operator Type: Function [other Function type actions]
Operator Scope of Action: Item [operators of similar scope]
Operator Purpose: Dictionary, Set & List operations [other Dictionary, Set & List operations operators]
Operator First Added: Baseline
Operator Last Altered: As at baseline

Dictionary[keyStr]

returns the Dictionary's value for the submitted **keyStr** string. For example, make a dictionary:

```
$MyDictionary={cat:animal; dog:animal; rock: mineral};
```

or (now deprecated):

```
$MyDictionary="cat:animal; dog:animal; rock: mineral";
```

then test like so:

```
$MyString = $MyDictionary["dog"];
```

sets MyString to the string "animal".

It is not required to use quotes around **keyStr**.

The **keyStr** may also be a (loop) variable, or an attribute value. The latter are evaluated before use:

```
$MyList.each(anItem){$MyString = $MyString + ", " + $MyDictionary[anItem];}
$MyString = $MyDictionary[$SomeAttribute];}
```

This has the same effect as **.at( )**, but may be more convenient. Note that in both the square-bracket syntax and .at() syntax a numerical key value is interpreted as a **key** and not a list item number, unlike List/Set.at().

distance(startItem, endItem)

Operator Type: Function [other Function type actions]
Operator Scope of Action: Item [operators of similar scope]
Operator Purpose: Data manipulation [other Data manipulation operators]
Operator First Added: Baseline
Operator Last Altered: As at baseline

---

distance(startItem, endItem)

returns the map distance (in map units) between the *centres* of two notes. **startItem** and **endItem** are a name, path, attribute value or expression returning a unique reference to a single note.

To find the distance between note 'AA' and note 'BB':

```
$MyNumber = distance("AA,"BB");
```

distanceTo(startItem, endItem)

Operator Type: Function [other Function type actions]
Operator Scope of Action: Item [operators of similar scope]
Operator Purpose: Data manipulation [other Data manipulation operators]
Operator First Added: Baseline
Operator Last Altered: As at baseline

---

distanceTo(startItem, endItem)

distanceTo computes the approximate distance in kilometres between two notes for which a $Latitude and $Longitude are known. **startItem** and **endItem** are a name, path, attribute value or expression returning a unique reference to a single note.

For example, if note **startItem** 'Boston' has the latitude and longitude of Boston and note **endItem** 'Paris' has the latitude a longitude of Paris, then for

```
$MyNumber = distanceTo("Boston","Paris");
```

$MyNumber is about 5582 (km).

Long distances should use a great circle route.

Note the measurement is approximate and not intended for precise navigation.

do(macroStr[,argumentsList])

Operator Type: Function [other Function type actions]
Operator Scope of Action: Item [operators of similar scope]
Operator Purpose: Data manipulation [other Data manipulation operators]
Operator First Added: Baseline
Operator Last Altered: As at baseline
Operator Has Optional Arguments: [More on optional operator arguments]

---

do(macroStr[, argumentsList])

The **do()** operator do lets rules and actions use macros. Action code macros cannot, generally, evaluate action (or export) code within the macro itself and so might best be thought of sections of boilerplate text with configurable text inputs.

do() always returns an output (string) and so will expect a left side attribute. To do something useful with the output consider wrapping the it in an eval() call. Assume macro "Hello world" has the code: `"Hello $1!"`. Then:

```
$ReturnString = eval(do("Hello world","Nibbler"));
```

would set a $ReturnString value of "Hello Nibbler!".

Alternatively, if the macro is written so the left side is included, action() can be used. Macro "Hello world2" has the code: `$ReturnString = "Hello $1!"`. Using the macro thus:

```
action(do("Hello world2","Cubert"));
```

which would set a $ReturnString value *of the current note* to "Hello Cubert!". Notice how action() requires no left-side receiving attribute as there is no direct output. The assignment to $ReturnString occurs within the macro; the assigned attribute being whatever system/user attribute the user decides, it is not a fixed output assignment.

An offset within an action call is also possible. Macro "Hello world3":

```
$ReturnString('$2') = "Here's $1!";
```

Use the macro thus:

```
action(do("Hello world3","Calculon", "All My Circuits"));
```

which would set a $ReturnString("All My Circuits") to "Here's Calculon!".

Note that if an input is itself an expression, it may prove beneficial to evaluate it before insertion into the macro. Why? It may affect outcome if the context of the input evaluation and the output evaluation differ, e.g. where the reference 'this' might refer to a different note in each context.

do(macro[,arg1,arg2,arg3])

The first argument is the name of the macro. Subsequent **argumentsList** (comma delimited) arguments are optional and are passed to the macro, which can refer to them as $1, $2, $3, and so forth.

After the macro is evaluated, its result string is returned and is parsed again as a rule, action, or expression. For example:

```
$Name|=do(computeName,$Name,$Name(parent));
```

sets name to the result of a macro called *computeName*. This if the macro code were:

```
$2: $1
```

Then, if $Name were "mammal" and $Name(parent) were "horse", the result on the above call would be a new $Name of "mammal: horse"

```
do(Instructions);
```

simply returns whatever text is stored in a macro called *Instructions*.

Macros cannot be used in agent queries (this was supported in early versions).

Macros called in action code via do() do not evaluate any embedded export codes. This is because such behaviour is reserved for export use.

document.keys()

Operator Type: Function [other Function type actions]
Operator Scope of Action: Document [operators of similar scope]
Operator Purpose: Document configuration [other Document configuration operators]
Operator First Added: Baseline
Operator Last Altered: As at baseline

Syntax note: *Operators without any defined mandatory arguments may omit their empty closing parentheses*

document.keys()

document.keys

returns a list of the document() dictionary's keys of useful properties of the current document. The operator returns the document Dictionary object as a single string, noting that no operator trailing parentheses are needed:

```
$MyList = document.keys;
```

The returned list is in the dictionary's stored order as listed for document().

The **document.keys** operator can assist in getting a readable output of all the keys and their values

```
$Text =;
document.keys.each(aKey){
 var:list vKey = document[aKey];
 $Text += aKey + ": " + vKey.format(", ") +"\n\n";
};
```

which will list in $Text of the current note each key name and then its value with a blank line in between. For multi-item values like **user-attributes** and **link-types** list formatting applies a ', ' delimiter for legibility.

document()

Operator Type: Function [other Function type actions]
Operator Scope of Action: Document [operators of similar scope]
Operator Purpose: Document configuration [other Document configuration operators]
Operator First Added: Baseline
Operator Last Altered: 9.6.0
Operator Has Optional Arguments: [More on optional operator arguments]

Syntax note: *Operators without any defined mandatory arguments may omit their empty closing parentheses*

document()

document

document() or **document** returns a dictionary of useful properties of the current document. The properties of the document are exposed this way (in this order in the dictionary):

- **path**: the POSIX file path of the current TBX file
- **id** (from v9.6.0): the TBX file's internal UUID.
- **user-attributes**: a list of all the user attributes currently defined in the document
- **url**: the local file:// URL of the current TBX file.
- **name**: the full name of the current TBX including the file extension
- **link-types**: a list of discrete link types currently defined in the document

Importantly, The **document()** Dictionary object is *read-only*.

The operator returns the document Dictionary object as a single string:

```
$Text = document();
```

Although trailing parentheses are not needed, when calling the 'bare' operator, add parentheses—i.e. **document()**— may help the user and parse not mistake the operator **document** for the literal string "document".

As reading the Dictionary data as a single string can be hard, a neater and more easily understood method is use the document.keys() method.

It is possible to address the **document()** Dictionary directly for a specific key value using the document[keyStr] method.

however, calling `document()` with no other handling, returns the entire, unformatted Dictionary contents. To split the items onto separate lines as discrete key:value pairs:

```
$MyString = document.format("\n");
```

To get the three values (without their keys) on separate lines:

```
$MyString = document.format("\n").replace("\n[^:]+:","\n").replace("^[^:]+:","");
```

document[keyStr]

Operator Type: Function [other Function type actions]
Operator Scope of Action: Document [operators of similar scope]
Operator Purpose: Document configuration [other Document configuration operators]
Operator First Added: Baseline
Operator Last Altered: As at baseline
Operator Has Optional Arguments: [More on optional operator arguments]

Syntax note: *Operators without any defined mandatory arguments may omit their empty closing parentheses*

document[keyStr]

To address the document() Dictionary to access a particular key use a **keyStr** argument. The values for the Dictionary's available properties can be accessed individually by supplying the relevant key. Quotes around the key are optional, both forms are illustrated below:

```
$MyFile = document["path"];
$MyString = document["id"];
$MyList = document["user-attributes"];
$MyURL = document["url"];
$MyString = document["name"];
$MyList = document[link-types];
```

All property **keyStr** above return a *string* value. However, the values are more usable when passed to attributes (or variables) of the precise data types shown in the examples above.

drivingTimeTo(item)

Operator Type:	Function [other Function type actions]
Operator Scope of Action:	Item [operators of similar scope]
Operator Purpose:	Data manipulation [other Data manipulation operators]
Operator First Added:	Baseline
Operator Last Altered:	As at baseline

drivingTimeTo(item)

drivingTimeTo estimates the driving time between the location specified by the selected (current) note and the location specified by the **item** path.

Location essentially means explicit $Latitude *and* $Longitude values for *both* notes—i.e. start and destination, or data such as $GeocodedAddress that can be examined to retrieve suitable Lat/Long data.

The result is a time interval (interval data type) estimating the approximate driving time, based on traffic conditions prevailing when the action was invoked. For example to find the time to drive from the current location's

```
if ($MyInterval==0) {
   $MyInterval = drivingTimeTo(/places/faves/Swarthmore);
}
```

There is no result, i.e. estimated driving time is zero (interval of 00:00) if:

- a location for either note cannot be determined
- no route can be found between the two locations (or the locale is not mapped for driving time)
- the internet is not accessible.

Avoid excessive use

Note that this function can be slow and relies on the underlying macOS. Therefore consider using an edict rather than a rule, and avoid recalculating the driving time if you already know it. Code as in the example above should stop unneeded calls. Which such code, if there is a driving time, delete it, and the conditional guard query will allow a fresh call to drivingTimeTo().

eachLink(loopVar[,scope]){actions}

Operator Type:	Function [other Function type actions]
Operator Scope of Action:	Item [operators of similar scope]
Operator Purpose:	Linking [other Linking operators]
Operator First Added:	Baseline
Operator Last Altered:	9.5.0
Operator Uses Scoped Arguments:	[More on scoped arguments in Action Code]
Operator Has Optional Arguments:	[More on optional operator arguments]
Operator Uses Loop Variable Argument:	[More on loop variable arguments]

eachLink(loopVar){action(s)}

The **eachLink()** operator examines each link for the current note, either inbound or outbound; prototype links are excluded. The local, user-named, variable **loopVar** argument is bound to a dictionary-type object of per-link properties, and is used in the `{}`-enclosed **action** code.

From v9.6.0, **.eachLink()** includes 11 new new read-write keys in the dictionary it builds for each link. These first two can be useful when several notes in a container have identical names and the remaining new keys cover various display aspects of the link. The per-link Dictionary-type data comprises the following keys. All are editable except *item in italics*:

- type. The link's link type.
- anchor. For text links, the links' anchor text within $Text.
- comment. For ad hoc information about the link or its purpose
- source. $Path of source object.
- sourceID. $ID of the source note. v9.6.0+
- sourceIDString. $IDString of the source note. v9.6.0+
- dest. $Path of destination object. Alternate for 'destination', more consistent with ID-based versions. v9.6.0+
- destID. $ID of the destination note. v9.6.0+
- destIDString. $IDString of the destination note. v9.6.0+
- destination. $Path of destination object. Alternate for 'dest'.
- class. HTML link `class` attribute.
- title. HTML link `title` attribute.
- target. HTML link `target` attribute.
- url. For Web links only, the linked-to URL.
- (boolean visual properties, all v9.6.0+):
  - visible
  - dashed
  - dotted
  - bold
  - broad
  - linear
- *isFirst* `true` if the first (or only) listed link for this note.
- *isLast* `true` if the last listed link for this note.

Another way to understand the keys, is in terms of their general purpose:

- link type information: type, anchor comment
- Identity of linked items: source, sourceID, sourceIDString, dest, destID, destIDString, destination
- Web link configuration for HTML export: class, title, target, url. See more under the Browse Links dialog.
- Configuration of visible links (Map and Timeline views): visible, dashed, dotted, bold, broad, linear. See more under the Links Inspector.

For editing key values, see section 'Editing link properties' below.

Note that it is not possible to read a link *target*'s anchor text (i.e. for a link linking *to* a $Text selection) via **eachLink()**. Currently such data is only observable via the source XML of the document.

From v9.6.0, **eachLink()** examines each link in the sequence as seen in the listing in Browse Links…, so it examines text links is the order their anchor text appears within $Text.

Link Counts

The count of the links iterated for a given note by eachLink() is $OutboundLinkCount plus $InboundLinkCount:

```
var:number vLkCt = $OutboundLinkCount + $InboundLinkCount;
```

First and Last item tests

Although **eachLinks()** is functionally like a list iterator (i.e. List.each()), it works off a dynamically generated List but the **loopVar** is bound to each list item's Dictionary. This means the list object properties of List.count, List.first and List.last ar not available. However, the per-link Dictionary object offers two special keys:

- **loopVar["isFirst"]** is `true` for the first link in the enumeration and `false` otherwise.
- **loopVar["isLast"]** is `true` for the final link in the enumeration and `false` otherwise.

In effect these keys mimic, for **eachlink()**, the .first() and .last() tests as used by **.each()**. This the following example the first conditional if() test uses the long form test for a boolean `true`, the second the short form—both give the same result

```
eachLink(aLink){
 if(aLink["isFirst"]==true){
  //this is the first link in the listing;
 };
 if(aLink["isLast"]){
  //this is the last link in the listing;
 };
};
```

Why might this be used? If the links are being processed so as to only export certain typed links, it may be necessary to add additional text/styling before the first and after the last link.

Detecting basic vs. text vs. web links

The 3 types differ in that:

- Only web links have an **anchor** *and* **url** value.
- Only text links have an **anchor** *but no* **url** value.
- Only basic links have *neither* an **anchor** *nor* a **url** value.

These conditions can be tested for when iterating a note's links

```
eachLink(aLink){
 if(aLink["url"]!=""){
  // this is a web link
 }else{
  if(aLink["anchor"]!=""){
   // this is a text link (has anchor but no url)
  }else{
   // this is basic link (has neither anchor nor url)
  }
 }
};
```

Or as a function:

```
function fWhatSortOfLink(iAnchor:string, iURL:string){
 if(iURL!=""){
  return "web";
 }else{
  if(iAnchor!=""){
   return "text";
  }else{
   return "basic";
  }
 }
};
var:string vTypeOfLink;
eachLink(aLink){
 vTypeOfLink = fWhatSortOfLink(aLink["anchor"],aLink["url"]);
  // do something depending on the type of link
 };
```

More examples

Filtering inbound vs. outbound links (outbound links share the same $ID as the note whose links are being read):

```
function fLinkDirection(iIdNote:string, iIdLink:string){
 if(iIdNote == iIdLink){
  return "outbound";
 }else{
  return "inbound";
 }
};
// called like so
$MyString = vTypeOfLink = fLinkDirection($ID,$ID(aLink["source"]));
```

Does this note have a link of type "agree"?

```
function fIsAgreeable(){
 eachLink(aLink) {
  if(aLink["type"]=="agree"){
   return true;
  };
 };
 return false;
};
```

Or, to count the number of links from this note to tasks:

```
function fLinkedTasks(){
 var:number vCount=0;
 eachLink(aLink){
  if($Prototype(aLink["destination"])=="Task"){
   vCount += 1;
  };
 };
 return count;
};
```

The examples use line breaks for clarity, and the last example may equally well be stored and used without line breaks, though still taking care to delimit/terminate discrete expressions with semicolons

The latter might make sense if trying to use a function within something like a rule, but if the code is defined in a Library note within Hints, then there is little gain in a one-line approach as it can be hard to read.

eachLink(loopVar,scope){action(s)}

From v9.5.0, an optional second argument for **eachLink()** allows designating the note whose links are to be examined, using a path or designator as the **scope** argument. Previously, **eachLink()** was explicitly bound to the current note.

For example,:

```
eachLink(aLink, parent){
 … // action code here
};
```

performs an action on each of the links to and from the *parent* of the current note (i.e. *this* note).

Note that **scope** is a single item; if needing to work a list of notes, use a nesting **List.each()** with the outer loop passing a single **scope** item to eachLink(). Consider the following:

```
$MyList.collect(children,$ID);
$MyList.each(anID){
   eachLink(aLink,anID){
      // do stuff on the links of the note defined by current anID's $ID
   }
};
```

The latter might as easily be done with a function, that takes $ID as an input and wraps the eachLink() loop using the passed-in ID as the eachLink() second argument.

Where the designator argument would be *this*, it may be omitted as the original usage `eachLink(aLink){…}` automatically assume the context of the action performed on each link to and from *this* note.

Editing link properties

In **eachLink()** loops, most properties of the link are editable (at least, from v9.6.0). For example, to change any of the current note's 'untitled' links to link type 'reference':

```
eachLink(aLink){
 if(aLink["type"] == "*untitled"){
  aLink["type"]  = "reference";
 }
};
```

If the link type being set does not already exist, it is created and applied. But, what if the note to be checked is not the current note? Consider working via an agent, where the current note is an alias. In that case, it is necessary to use the optional **scope** argument and additionally rather than specify a note name or path to use the original designator. The same task as above now becomes:

```
eachLink(aLink,original){
 if(aLink["type"] == "*untitled"){
  aLink["type"]  = "reference";
 }
};
```

the only change being the extra use of the **scope** argument to define the correct context for evaluating the links.

To change the link type of any link of type 'disagree' to 'agree', within the **eachLink()** loop use:

```
    if(aLink["type"]=="agree"){aLink["type"]="disagree";};
```

escapeHTML(dataStr)

Operator Type: Function [other Function type actions]
Operator Scope of Action: Item [operators of similar scope]
Operator Purpose: Formatting [other Formatting operators]
Operator First Added: Baseline
Operator Last Altered: As at baseline

escapeHTML(dataStr)

escapeHTML converts HTML and XML special characters in string **dataStr** to HTML/XML entities. For example, '<' is replaced by '<' and '&' is replaced by '&'.

For example, if the source **dataStr** is "this & that":

`$MyString = escapeHTML("this & that")` outputs `"this & that"`.

See also attributeEncode(), especially if working with XML strings.

eval([item], expressionStr)

Operator Type: Function [other Function type actions]
Operator Scope of Action: Item [operators of similar scope]
Operator Purpose: Data manipulation [other Data manipulation operators]
Operator First Added: Baseline
Operator Last Altered: As at baseline
Operator Has Optional Arguments: [More on optional operator arguments]

eval([item,] expressionStr)

The **item** argument must be quoted unless an attribute reference. Ways to define **item**.

The eval() function takes two quoted evaluated (as string) arguments. The first argument, **item**, is optional. The second argument , expressionStr, holds string holding an action code expression to be evaluated. Thus, in basic form:

```
if($MyBool){$AttribX = "$Name(parent)+' XYZ'";}else{$AttribX = "$Name(grandparent) + 'ABC'";}; $AttribY = eval($AttribX);
```

will set AttribY to the name of the current note's parent's title plus string "XYZ" or that of its grandparent plus string "ABC", depending on the value of attribute MyBool by evaluating attribute AttribX. Thus eval() gives the action code result of an attribute rather than its literal value. As such eval is normally applied to string-based attributes.

If trying to construct an attribute reference from "$" and a literal string or variable, use action() instead.

Further examples

The eval() function can be useful for inserting a local attribute value in an expression that will be evaluated in a different context. Consider this find() in an a rule:

```
$ArtistCount = find($ArtistName==$Name & !$IsAlias).size;
```

For each note evaluated, the note's $Name is compared to ArtistName. However, what if the intent was to use this in a note with an artist's name where it is necessary to come the calling note's $Name, not that of the in-scope note. There is no designator for that relationship, but eval() offers a way around:

```
$ArtistCount = eval('(find($ArtistName=="'+$Name+'" & !$IsAlias).size)');
```

This first of all makes a string using the correct find code but also inserting the value of the calling note's $Name, and then the whole is evaluated as if it were the original verbatim find() call above.

The process can be made clearer by splitting it into two steps by first using a placeholder $TempString attribute:

```
$TempString = '(find($ArtistName=="'+$Name+'" & !$IsAlias).size)';
```

The outer parentheses are simply to ensure the .size() call chained on the end of the find is evaluated with the find(). For a note called "Jacques Brel", $TempString would be the string: `(find($ArtistName=="Jacques Brel" & !$IsAlias).size)`. Now, to use eval():

```
$ArtistCount = eval($TempString);
```

Side note: the actual scenario above can also be solved using the ' that' designator, although the example holds true as an exploration of how eval() works.

A more complex example, using nested eval() calls is described under ' using long sections of code'.

Macros: eval() can be combined with do(macro). As macros take input arguments, an evaluated macro can work a bit like a code function, taking inputs and returning output that once passed through eval() gives an evaluated result. See the do() operator.

The eval() operator also allows access to two Tinderbox properties that are not available via action syntax or attribute value. There are the current TBX document's filename (sans extension) and the app version of the currently used Tinderbox on the user's Mac:

`eval(^docTitle^)` gives a value of "aTbRef-95" (note no '.tbx' extension)

`eval(^version^)` gives "9.6.2" (note that you might want to prefix the return string with 'v' or 'v.' thus: "v.9.6.2").

In full syntax form, an additional first argument is added that is an expression string evaluating as a note name, path or note name. Where specified, this indicates the note from which attribute values in the second argument should be drawn.

To create an attribute reference ($-prefixed as in '$Name' not 'Name') use action() instead of eval().

Example using both inputs

A different example, using the optional path argument. Take two root-level notes, AA and BB.

$Text("AA") is: `1+1`

$Text("BB") is: ""(i.e. nothing)

$Rule("BB") is: `$Text=eval("AA",$Text)`

…after a brief delay, the text of note BB becomes 2, i.e. the sum of the expression stored in aa's text. The **item** argument "AA" simply indicates that the source of the **expressionStr** is the note AA.

Now change AA's text:

$Text("AA") is: `if(1){42}else{1000}`

…after a brief delay, the text of note BB becomes 42. How? Any if(condition) is a Boolean test trying to get a true/false result. If the condition does not result in an actual Boolean value (e.g. testing the value of a Boolean attribute) a Boolean is coerced from the result. Thus an empty string or set or the number zero equate to `false`, all other values to `true`. Thus in AA's text above, the condition (1) equates to `true` so the result is 42.

Again, replace the $Text of note 'AA' with

$Text("AA"): `$Color=="red"`

This is less obvious. When the expression is evaluated, the result will be `true` if AA is red, and `false` otherwise. So, $Text("BB") will be empty unless AA is red; if AA's $Color is red,

$Text("BB"): `true`

every(scope, condition)

Operator Type: Function [other Function type actions]
Operator Scope of Action: Group [operators of similar scope]
Operator Purpose: Non-query Boolean [other Non-query Boolean operators]
Operator First Added: Baseline
Operator Last Altered: As at baseline
Operator Uses Regular Expressions: [More on regular expressions in Tinderbox]
Operator Uses Scoped Arguments: [More on scoped arguments in Action Code]
Operator Has Conditional Arguments: [More on conditional operator arguments]

every(scope, condition)

This Boolean operator examines notes at **scope** and determines whether any note (i.e. at least one) in the group evaluates condition as `true`.

scope is the notes to be examined and may be any group designator including a find() query. If group evaluates to contain no items, see the edge case described below.

condition may be any valid expression, but will usually be a reference to an attribute; short form Boolean attribute expressions are acceptable.

condition must not be enclosed in quotes.

For example:

```
every(child,$Status=="Important")
every(children,$Overdue==true)
every(children,$Overdue) (using short from test)
every(children,$Overdue==false)
every(children,!$Overdue) (using short from test)
```

Be aware that a counter-intuitive edge case occurs if the **scope** is empty. Thus, every() is `false` if **condition** is `false` for any designated note. If there are no designated notes at all, i.e. **scope** is empty, every() is therefore `true`, whereas th user might have assumed no outcome and thus no `true` result. If problematic to the user's intent, this condition can be worked around by first ensuring that **scope** contains something. In the above example, that uses the **scope** 'children', a modification can be used, like this:

```
$ChildCount>0 & every(children,$Overdue==true)
```

See also: any().

exp(powerNum)

Operator Type: Function [other Function type actions]
Operator Scope of Action: Item [operators of similar scope]
Operator Purpose: Mathematical [other Mathematical operators]
Operator First Added: Baseline
Operator Last Altered: As at baseline

---

exp(powerNum)

Returns the exponential value of **powerNum** as *e* to the power of **powerNum**.

```
$MyNumber = exp(3);
```

returns 20.08553692 for an input of 3.

exportedString(item[, templateStr])

Operator Type: Function [other Function type actions]
Operator Scope of Action: Item [operators of similar scope]
Operator Purpose: Formatting [other Formatting operators]
Operator First Added: Baseline
Operator Last Altered: As at baseline

---

exportedString(item[, templateStr])

The operator **exportedString()** applies an export template to a note, returning the result as a String.

item designates the note to be exported; it is commonly "this". It is a note name, path or item designator. An attribute may not be used as a placeholder for such information. The exportedString() operator binds 'current' to 'this' note. Formerly, 'current 'was only bound in export templates. In short, this short improve the operator exporting the correct content.

The second argument **templateStr** has 3 forms.

- an HTML export template name; this argument is evaluated allowing use of attribute values and expressions. Full template paths should not be quotes, but enclose template names ($Name) in double-quotes.
- the name of an attribute or variable whose value is a string of actual template code (i.e. what would normally be the contents of a template), which is then applied to the referenced note. The attribute reference cannot use a path extension to refer to a different note, $ExportCode is OK but an offset address like $ExportCode(Another note) will fail.
- a literal string of export code. The most usual use of this is `"^text(exportedString(item[, templateStr]),PDF-source-item):: text tries to include itself "`. This form is not suggested for other than *very sho* code strings.

The exportedString() operator is especially useful in conjunction with the runCommand() operator. You can use exportedString() to assemble the input an external program will require, and then pass that input to the external program.

If you simply wish to transform a string or attribute value (e.g. $Name) into a 'safe' value for use as an HTML/XML element 'id' value, use idEncode().

Examples

exportedString(item)

Use of exportedString with just the target **item**:

```
$MyString = exportedString("Some Note");
```

exportedString(item, templateStr)

The first form using **templateStr** takes a reference to a note and the name of a template note and returns the result of applying the template to the note. The first example used just the (unique) template note $Name:, which should be quoted

```
$MyString = exportedString(this,"tSome_Template");
```

if item is a path or title, the value requires quotes:

```
$MyString = exportedString("Some note","tSome_Template");
```

or use the unquoted full $Path for the template:

```
$MyString = exportedString(this,/Templates/tSome_Template);
```

The second form using **templateStr** requires as its second argument is the template string itself which may be literal code or an attribute reference.

```
$MyString = exportedString(this,$MyTemplateCode);
$MyString = exportedString(this,vTemplate);
```

The third form using **templateStr** requires as its second argument some literal code:

```
$MyString = exportedString("/Path/To/Some note","^value($Name(parent))^");
```

fail()

Operator Type: Function [other Function type actions]
Operator Scope of Action: Item [operators of similar scope]
Operator Purpose: Stream parsing [other Stream parsing operators]
Operator First Added: Baseline
Operator Last Altered: As at baseline

Syntax note: *Operators without any defined mandatory arguments may omit their empty closing parentheses*

---

fail()

fail

The **fail** operator makes an explicit declaration of a failure, and if invoked only ever returns `false`.

Often, in stream processing a failure occurs implicitly, for example because the end of the processed string is reached without finding the desired data. However, sometimes, there is a definite need to signal failure; for example, because the parsed data are invalid.

For example:

```
$MyString.try{
   $MyString.skipTo(":").captureWord("TheWord");
   if($TheWord.contains("None")) {
      fail();
   }
}
```

The above skips through $MyString to the first colon and then captures the next discrete word after the colon into $TheWord. The action fails if:

- if there is no colon
- if there is no word after the colon
- if $TheWord contains "None" as a case-sensitive exact full match or substring match.

fetch(urlStr,headersDict,attrNameStr,cmdStr[,httpMethod])

Operator Type: Function [other Function type actions]
Operator Scope of Action: Item [operators of similar scope]
Operator Purpose: Document configuration [other Document configuration operators]
Operator First Added: 9.6.0
Operator Last Altered: As at baseline

---

fetch(urlStr,headersDict,attrNameStr,cmdStr[,httpMethod])

In v9.6.0, the **fetch()** operator enables collecting data from a server when greater flexibility than when using AutoFetch. For most purposes, AutoFetch should be the primary choice;> If uncertain use AutoFetch unless it proves not to work; **fetch()** is an advanced operator for the few who need it.

urlStr is the URL source to be called into Tinderbox.

headersDict is a dictionary of HTTP(S) headers. This must be an empty dictionary `{}` (no enclosing quotes) if you do not need custom headers.

attrNameStr is the name of (*not* a reference to) an attribute or local variable in which the output of the fetch process is stored. Note the difference from AutoFetch which places the output directly into $Text. Here, the data from the remote server will be fetched asynchronously, and stored in this attribute or local variable. Note that this parameter is not evaluated, i.e. `"MyString"` *not* `$MyString`, unless attribute MyString hold the name of an attribute. This attribute cannot be left empty and must be a literal quoted string, an attribute reference or a variable.

cmdStr is a Tinderbox action expression (or semi-colon delimited sequence of actions) to be performed after the data is fetched; i.e. the value is action code. Be aware that the latter *may* be some time after the the fetch operator returns; do not assume full processing happens immediately. It may be an empty string `""` if no transform is needed on the recovered data (this in unlikely to be the case).

httpMethod is an optional argument, indicating the HTTP request method to be used. HTTP 1.1. defined methods (per RFC 9110) are `GET`, `HEAD`, `POST` , `PUT` (including `PATCH`), `DELETE`, `CONNECT`, `OPTIONS`, and `TRACE`. If **httpMethod** is n specified, **fetch()** defaults to using a GET request. The methods allowed are as per the HTP protocol in use at the server, rather than a restriction by Tinderbox. Also be aware that support for a given method may also be a constraint at the *server* end of the request.

Note that the server might take several seconds or more before supplying a response. **fetch()** returns immediately without waiting for the server, and arranges for the command to be evaluated when the data arrive; Tinderbox will continue to try for c.30 seconds before giving up. Why? Because Tinderbox has no control over the time the source at **urlStr** takes to return the requested data.

Given this possible delay between calling **fetch()** and the completion of the full process, it may be necessary to take steps to avoid repeating the **fetch()** call *before* the previous operation has completed. For example, this can be done by setting a $MyBoolean(/In Progress) to `true` immediately before starting the fetch, and setting it to `false` when all is complete. Note: the choice of an "In Progress" note to hold the boolean is just one approach—where the boolean value is stored and in which attribute are choices for the user to make.

fetch() processes its command on a serial queue. So, if several fetch() requests are outstanding, they will not interfere: that queue is the sequence in which they will be evaluated.

The show() operator can be used as part of the **cmdStr**, as a way of giving feedback.

Examples

Suppose there is a need to interact with an API. Typically, that means:

- Passing a GET or POST message to a designated URL.
    - Possibly doing this using some custom headers for authentication or other reasons.
- Wait for the server to respond.
- When the server does respond, take what is returned and process it.

Note how this is more complex than previous AutoFetch. But, custom headers may be needed or it may be necessary to process the results right away. For example, the result might say: "Here are the first ten results; and here is a magic token to send to get the next ten." Such tokens are typically good for a short time only. AutoFetch was not designed for such complications.

In the following example, notice the **fetch()** argument #2 uses the newer `{}` style of dictionary declaration:

```
var string aServerULR = "https://yourserver.com";
var string myPayload = "?status=publish";

fetch(aServerULR + myPayload,{Content-Type: application/json},"aTempVariable","doProcessCallback(aTempVariable);",POST)

function doProcessCallback(returnValue:string) {
    show(returnValue);
};
```

Here is a fuller example of a Readwise data importer function as an example. Be aware that the code refers to user fuctions, e.g. '`rwHeaders()`' that are *not included* in the example code:

```
function getHighlights(auth:string,page:string) {
 rwSetup();
 if(rwIsInProgress()) {
  show("Readwise already in progress");
  return;
  }

 show("Readwise starting import");
 rwMarkInProgress();

 var:string theLog=create("/log");
 fetch(rwURL(page),rwHeaders(auth),"temp","rwParse(temp,/log,0)");

 }
```

Note: everything in the above example with an 'rw' prefix is a function.

find(scope)

Operator Type:	Function [other Function type actions]
Operator Scope of Action:	Query [operators of similar scope]
Operator Purpose:	Dictionary, Set & List operations [other Dictionary, Set & List operations operators]
Operator First Added:	Baseline
Operator Last Altered:	As at baseline
Operator Uses Scoped Arguments:	[More on scoped arguments in Action Code]

---

find(query)

The find() operator returns List-type data of the $Path of all items matching the query argument **scope** (defining **scope**). Note that find(), unlike agent queries, *does not de-duplicate* its matches so any aliases in scope with be matched: to av the latter see section 'Filtering out aliases' below.

To return a list of paths for notes with prototype of 'pRef' and fro which boolean $HasRef is `false` use:

```
find($Prototype=="pRef"&$HasRef==false)
```

Conceptually find() is intended for where no item or group designator exists for use inline in action code (note: such designators were the only scoping terms in early version of Tinderbox). In most cases an agent is the likely and better alternative, noting that agents can also use the results of other agents. find() can also be thought of as a short way of writing:

```
$MyList = collect_if(all, expression, $Path);
```

where the expression is **scope**.

The find() operator omits notes for which $Searchable is `false`.

Filtering out aliases

The find() operator does not de-dupe results in the way an agent does. As find() collects $Path data, aliases both inside and outside agents may also match the query. Adding `!$IsAlias` as a query term will scrub any aliases from matching search.

Thus if getting unexpected results via find(), consider whether it is because some de-duping was (incorrectly) assumed and be prepared to sharpen the query terms accordingly.

In the above example, to additionally filter out possible alias matches add an extra query term:

```
find($Prototype=="pRef"&$HasRef==false&$IsAlias==false)
```

first(item[, childrenNum])

Operator Type:	Function [other Function type actions]
Operator Scope of Action:	Item [operators of similar scope]
Operator Purpose:	Query Boolean [other Query Boolean operators]
Operator First Added:	Baseline
Operator Last Altered:	As at baseline
Operator Has Optional Arguments:	[More on optional operator arguments]
Operator Has Newer Dot-Operator Variant:	Yes

---

first(item[, childrenNum])

Returns Boolean `true` if the current note is among the first **itemsNum** of children of **item**. If **itemsNum** is missing, a value of 1 is assumed. The **item** argument must be quoted unless an attribute reference. Ways to define **item**.

Both arguments are evaluated and can be a literal string/number, an attribute value or an action code expression evaluating to that same.

If the current note has a $SiblingOrder value of 2, then if first() is run on its parent container:

`first("Note A", 5)` returns `true`

but if 'it has a $SiblingOrder value of 7:

`first("Note A", 5)` returns `false`

first() also has a logical opposite in last().

Legacy issues

This operator replaces the legacy #first query operator.

firstWord(dataStr)

Operator Type: Function [other Function type actions]
Operator Scope of Action: Item [operators of similar scope]
Operator Purpose: Data manipulation [other Data manipulation operators]
Operator First Added: Baseline
Operator Last Altered: As at baseline

---

firstWord(dataStr)

The **firstWord()** operator has one argument, **dataStr** (a quoted string), and it returns the first word of **dataStr**. The delimiter used to define words is one or more spaces (possibly also line break(s)?).

The **dataStr** argument is evaluated so could be an expression. For example, if the note 'First Line' has the body text "Winter is coming.", then

```
$MyString = firstWord($Text("First Line"));
```

should give a result of "`Winter`".

floor(sourceNum)

Operator Type: Function [other Function type actions]
Operator Scope of Action: Item [operators of similar scope]
Operator Purpose: Mathematical [other Mathematical operators]
Operator First Added: Baseline
Operator Last Altered: As at baseline
Operator Has Newer Dot-Operator Variant: Yes

---

floor(sourceNum)

rounds the number value of **sourceNum** down to next whole integer.

See also ceil() and round().

If $MyNumber is 3.9 then:

```
$MyNumber = ceil($MyNumber);
```

sets $MyNumber to 3. Note unlike normal rounding the value is set downwards to the next integer (i.e. whole number).

format(dataStr, formatStr[, additionalArguments])

Operator Type: Function [other Function type actions]
Operator Scope of Action: Item [operators of similar scope]
Operator Purpose: Formatting [other Formatting operators]
Operator First Added: Baseline
Operator Last Altered: As at baseline
Operator Has Optional Arguments: [More on optional operator arguments]
Operator Has Newer Dot-Operator Variant: Yes

---

format(dataStr, formatStr[,additionalArguments])

*Note: in most cases it is better to use newer **.format()** dot-operator for this task.*

The operator **format()** converts various Tinderbox objects to strings. In quoted (") string arguments a `\"` is converted into a quotation mark (a.k.a. double quote), `\n` to a carriage return and `\t` to a tab.

The argument **dataStr** is evaluated, and is usually an attribute reference or expression. An attribute reference can take both short and long forms:

`$MySet` (i.e. the current note's $MySet)
`$MySet(Test)` (i.e. set $MySet data from note 'Test')

The meaning of **formatStr** depends on the type of object represented by what. Additionally, number of **additionalArguments** may be supplied depending on the **dataStr** data type. Tinderbox data types Date, Set and Number are handled using different sets of arguments as described below.

This function is being supplemented by per-data-type .format() dot operators which are usually more flexible when writing non-trivial code: links to per-data-type dot operators are added below.

DATE-Type Data

If **dataStr** is a date, the format string is the same as the format used by Tinderbox's date export codes.

format(dataStr, formatStr)

For example:

```
$MyString = format($Created,"L");
```

gets the note's creation date and formats it as a "long local date" such as "Sunday, 23 March, 2007".

Also use Date.format("formatString").

SET/LIST-Type Data

If **data** is a Set or a List, the format string is the delimiter used to separate set elements:

format(data, formatStr)

The process preserves original data; duplicate values in lists are maintained. For example

```
$MyString = format($DisplayedAttributes,", ");
```

converts Displayed Attributes to a comma+space-separated list. To put each item on a separate line use this:

```
$MyString = format($DisplayedAttributes,"\n");
```

Thus $Text may be created from concatenation of other texts:

```
$Text = format(collect(children, $Text),"\n");
```

To strip duplicates from a List, do not use format(). Instead, simply set a Set attribute to the contents of the List attribute (or any function/expression returning List-type data).

Also use List.format("formatString").

Optionally, you may supply **formatStr** as four discrete arguments to format a list, for example as an HTML list:

format(data,list-prefix,item-prefix,item-suffix,list-suffix)

For example

```
$MyString = format($Classes,"<ul>","<li>","</li>","</ul>");
```

will return HTML code for a bulleted list with each set member marked up as a list item. Note that the tags must be in double quotes. To have each element on a separate line and indent the items add tabs and line breaks:

```
$MyString = format($Classes,"<ul>\n","\t<li>","</li>\n","</ul>\n");
```

To make this easier to use in a code export context, you might pass the output of format into another attribute and call the latter within the template with a ^value()^ code. By a repeat call to format it is possible to export lists of links.

Also use List.format("listPrefix","itemPrefix","itemSuffix","listSuffix")

NUMBER-Type Data

Use of format() with number data is deprecated in favour of either the Number.precision() or Number.format() operators.

If **dataStr** is a number, then the arguments are numeric and interpreted as follows:

format(dataStr,precisionNum[, widthNum][,padStr])

The **precisionNum** argument controls the number of decimal places returned. The optional **widthNum** argument allows the returned value to be a string left padded with spaces, e.g. to return a string with the same number of characters as submitted.

For example, if $MyNumber is 3.1415927, then

`$MyString = format($MyNumber,2,7);` is " 3.14" (3 spaces + number + period + 3 numbers = 7)
`$MyString = format($MyNumber,2);` is 3.14
`$MyString = format($MyNumber,0);` is 3
`$MyString = format($MyNumber,0,2);` is ' 3' (two left-padding spaces)
`$MyString = format(5.1415927,2);` is 5.14

Note that with **widthNum**, decimal character is not counted as part of the number.

If the optional **padStr** is given, this specifies the character used for padding. The default is a space:

`$MyString = format(7,0,3);` gives " 7"

`$MyString = 7.format(7,0,3,"0");` gives "007"

`$MyString = format(7,0,3,"#");` gives "##7"

An alternate number format is offered using a quoted format string:

Number.format(formatStr)

Currently only one such string is supported "L". This will return a string of the number formatted with (OS) locale-dependent group & decimal delimiters. For example, for the US locale these are a comma and a period; in other locales they may vary.

Also use Number.format(decimalsN,widthN).

COLOR-Type Data

If **dataStr** is color type, format strings are ignored:

format(colorStr)

The operator returns the colour **colorStr** in hex form, e.g. "#ff00ff", regardless of whether the stored value is hex or a named Tinderbox colour.

```
$MyString = format($MyColor)
```

is "#ff00ff" if $MyColor is "purple"

Also use Color.format().

INTERVAL-Type Data

If data is interval-type, format strings can use date-type format strings. An interval of 00:00 (hours:minutes) is always displayed as an empty string.

Also use Interval.format().

function

Operator Type:	Statement [other Statement type actions]
Operator Scope of Action:	Document [operators of similar scope]
Operator Purpose:	Data manipulation [other Data manipulation operators]
Operator First Added:	Baseline
Operator Last Altered:	9.5.0
Operator Uses Loop Variable Argument:	[More on loop variable arguments]

---

function

The **function** statement defines a user-defined function. Unusually as an operator **function** is used as a statement which is then followed by the definition of a user-named function **fName**. Thus:

function fName([argumentsList]){code}

Once defined a function is called in action code using the user-defined function name (here **fName**), as described here. Any function may several arguments or none at all. These are defined via a comma-delimited **argumentsList**. All arguments defined in the list are mandatory, unlike some action code operators which define some arguments as optional. In use, all defined arguments must be supplied, even if only as a default value, e.g. `0` for a number, etc.

All input arguments (the **argumentsList**) are available to use in code within the function. Thus by using the argument name as defined in the **argumentsList**, each argument's supplied value is retrievable anywhere within the function. Just as with a loop variable, the defined argument name is the code used to retrieve that argument's value.

As functions are more complex than other operators, their complete syntax and use is described in detail in the article on Functions and its sub-articles.

From v9.5.0, function declarations may optionally specify the date type of their input arguments, as passed via **argumentsList**.

hasLocalValue(attributeNameStr[, item])

Operator Type:	Function [other Function type actions]
Operator Scope of Action:	Item [operators of similar scope]
Operator Purpose:	Data manipulation [other Data manipulation operators]
Operator First Added:	Baseline
Operator Last Altered:	As at baseline
Operator Has Optional Arguments:	[More on optional operator arguments]

---

hasLocalValue(attributeNameStr[,item])

For the current note this tests if an attribute name **attributeNameStr** has a locally set value (returning a Boolean `true`), or whether that value is inherited from a prototype or document default (`false`). The main input is a quote-enclosed system or currently defined user attribute (without a $ prefix):

```
$MyBoolean = hasLocalValue("Rule")
```

The arguments are evaluated, so

```
$MyBoolean = hasLocalValue($MyString)
```

returns information about the attribute whose name is currently saved as the current note's value of $MyString.

The note evaluated is the current note. The '**item**' input allows for offset addressing. For example:

```
$MyBoolean = hasLocalValue("Rule","Some Note")
```

checks the local value status of $Rule not for the current note but for the note called "Some Note"

Most users will not have need of this but it is useful in very large document to find the odd note with a local setting, when visual review would take too long.

hour(aDate[, hoursNum])

Operator Type:	Function [other Function type actions]
Operator Scope of Action:	Item [operators of similar scope]
Operator Purpose:	Date-time [other Date-time operators]
Operator First Added:	Baseline
Operator Last Altered:	As at baseline
Operator Has Optional Arguments:	[More on optional operator arguments]
Operator Has Newer Dot-Operator Variant:	Yes

---

hour(aDate[,hoursNum])

Alternatively, use Date.hour.

hour(aDate)

returns the hour element from the **aDate** date/time expression, which may simply be a date-type attribute value.

hour(theDate, hoursNum)

creates a new date based on the **aDate** expression, but in which the hour is **hoursNumDate** is not changed unless theDate is an attribute and the attribute is re-setting itself:

```
$MyDateA = hour($MyDate,14);
```
$MyDate is unaltered

```
$MyDate = hour($MyDate,14);
```
$MyDate is changed

Examples. If $MyDate is July 4,2009 09:30, then

```
$MyDate=hour($MyDate,19);
```

will change $MyDate to July 4, 2009 19:30.

hours(startDate, endDate)

Operator Type:	Function [other Function type actions]
Operator Scope of Action:	Item [operators of similar scope]
Operator Purpose:	Date-time [other Date-time operators]
Operator First Added:	Baseline
Operator Last Altered:	As at baseline

---

hours(startDate, endDate)

returns the Number of whole hours that elapsed between **startDate** and **endDate**. If **endDate** is earlier than **startDate** then the result is negative.

If $DateA has time 12:30 and $DateB has time 15:00, then:

```
$MyNumber = hours($DateA,$DateB);
```

sets $MyNumber to 2.

idEncode(dataStr)

Operator Type: Function [other Function type actions]
Operator Scope of Action: Item [operators of similar scope]
Operator Purpose: Formatting [other Formatting operators]
Operator First Added: Baseline
Operator Last Altered: As at baseline

---

idEncode(data)

Returns input argument string **data** in a form suitable for use an id attribute value in HTML and XML. This addresses the problem of using general text strings such as note names as 'id' attributes in output for marked up languages like HTML and XML.

idEncode("stringValue")

idEncode($AttributeName)

Using idEncode() ensures the result begins with a letter or underscore, and contains only letters, digits, and the underscore character. Multiple underscores are collapsed to a single character. For example:

idEncode(string) → result string

frogs → frogs

frogs dogs → frogs_dogs

War And Peace →War_And_Peace

3 blind mice → _3_blind_mice

Wow!!! Look → Wow_Look

Notice that leading numbers (not legal for 'id' initial characters) are preceded by underscores to maintain legibility vs. the source string.

Note too (see syntax examples at top) that string literal values are "quoted" whereas attribute names are not, the latter's status being indicated by their $ prefix.

In code, the last example above can be expressed:

```
$MyString = idEncode("Wow!!! Look")
```

which gives a value `"Wow_Look"`.

if(condition){actions}[else{actions}]

Operator Type: Function [other Function type actions]
Operator Scope of Action: Conditional Group [operators of similar scope]
Operator Purpose: Data manipulation [other Data manipulation operators]
Operator First Added: Baseline
Operator Last Altered: As at baseline
Operator Uses Regular Expressions: [More on regular expressions in Tinderbox]
Operator Has Optional Arguments: [More on optional operator arguments]
Operator Has Conditional Arguments: [More on conditional operator arguments]

---

if(condition){actions}[else{actions}]

In rules and (agent) actions, Tinderbox uses this conditional action syntax:

if(*condition*){ action(s) }

if (*condition*){ action(s) }else{ action(s) }

Where:

- **condition** is any action code involving an evaluation to a conditional test that resolves to a Boolean `true` (if the condition is as expected). If the expression is in the form of an $Attribute.[i]contains(regex) query, regex back-references can be used in in the action code of the **actions**. A conditional expression can also be a query, such as find(), where matching zero results is `false` and otherwise `true` is returned.)
- **actions** is a list of one or more action code expressions, separated by semicolons.

Example:

```
if($ChildCount > 5){$Color = "red";$Width = 3.5;}else{$Color = "blue";}
```

A condition can have multiple tests. And (&) or Or (|) joins are allowed as is parenthesised nesting of expressions:

```
if($ChildCount > 5 | $Badge = "ok"){$Color="red";$Width=3.5;}
if(($ChildCount > 5 & $Badge = "ok") | $WordCount >= 450){$Color = "red";$Width = 3.5;}
```

If the expression is a complex set of condition clauses it may make sense to calculate the conditions, store the value in a attribute and test that in the if(0. This form of re-writing the test is a normal part of incremental formalisation such as may occur as a document gets bigger and more complex.

Negative tests

The testing for negatives, the absence of a value (or being in default condition), may be done in several ways. For an attribute value test, the attribute name may simply be prefixed with an exclamation mark, using the short form Boolean tes

```
if(!$MyString){…etc.
```

Otherwise the != operator is used:

```
if($MyString != "some value"){…etc.
```

Closing statements

It is not necessary to close the (last) action code statement in an action list, i.e. put a semi-colon before the closing '}'. However, a semi-colon closure is required *after* the last closing '}' if other action code follows within the same rule or action:

```
if($ChildCount > 5){$Color = "red";$Width = 3.5;}else{$Color = "blue";}; $Badge = "ok";
```

```
if(!$MyString){
 $Color = "red";
}else{
 $Color = "blue";
};
$WordCount(parent) = $WordCount(parent)+$WordCount;
```

Testing multiple conditions

Whilst a single if() test can use a complex expression to form its conditional test, it may be necessary to run a set of linked tests. However, there is no 'else if' construct as found in many programming languages. To handle more than two branches to a condition test, nest an additional if() in either action-list of the first test. Thus:

```
if(!$MyString){
 …
}else{
 if($MyBoolean){
  …
 }else{
  if($MyNumber > 2){
   …
  }else{
   …
  }
 }
}
```

In the example above there are two nested if tests within the original if(). Note how nesting is achieved. It is probably more normal that the additional if goes in the 'else' branch of the preceding condition but that is not a requirement. It can be in either branch according to the needs of the scenario. Indeed, both branches of an if could in theory hold another if().

However, this form of coding can quickly get complex and care should be taken with placing appropriate '}' closures. In complex branching, consider using additional queries to hold some initial levels of test, rather than do everything in a complex nested call.

Back-references

Back-references found by regular expression matches in operators like .contains() are available in if() clauses. For example, the rule:

```
if($Name.contains("a(..)") ){$MyString = $1;}
```

will set $MyString to "pp" if $Name is "apple", or to "rs" if $Name is "pears". Note that the $0 back-reference contains the full matched expression.

Within an overall action, any existing back-references generated by at if() persist until another if(0 statement is met. Note:

- back-references created by an if() statement persist beyond its {}-enclosed code. They last to the end of the action or a new if() statement.
- an if() statement re-sets *all* previous back-references even if it does not populate them. Thus if the first if() set a $1 and $2, and a second if sets only $1, $2 is now empty—the previous $2 value does not persist.
- even a nested if() resets *all* back-references for all the rest of the action.
- back-references are not accessible within an 'else' branch. An if() nested under the else branch is only evaluated (and thus sets back-references) if the original if() test fails. In other words, if the opening if is `true`, the code in the else `{ ... }` scope is never evaluated. But if the opening test fails, only the else scope is evaluated.

Inline use of if() statements

Assignment to an inline if() expression is permitted. For example:

```
$Color = if($MyBoolean){"red"}else{"blue"};
```

However, when practical, the more conventional and idiomatic style is preferred:

```
if($MyBoolean){$Color = "red";}else{$Color = "blue";};
```

Both the above are functionally equivalent.

indented(depthNum[, item])

Operator Type:	Function [other Function type actions]
Operator Scope of Action:	Item [operators of similar scope]
Operator Purpose:	Query Boolean [other Query Boolean operators]
Operator First Added:	Baseline
Operator Last Altered:	As at baseline
Operator Has Optional Arguments:	[More on optional operator arguments]

---

indented(depthNum[, item])

Returns Boolean `true` if the current note is **depthNum** levels below the root. Where **item** is supplied, **depthNum** is instead tested against the number of outline levels between the current note and **item** (as opposed to root). This is essentia a way to test the $OutlineDepth of a note.

The **item** argument must be quoted unless an attribute reference. Ways to define **item**.

Legacy issues

This operator replaces the legacy #indented query operator.

inheritsFrom([item,]prototypeStr)

Operator Type:	Function [other Function type actions]
Operator Scope of Action:	Item [operators of similar scope]
Operator Purpose:	Query Boolean [other Query Boolean operators]
Operator First Added:	Baseline
Operator Last Altered:	As at baseline
Operator Has Optional Arguments:	[More on optional operator arguments]

---

inheritsFrom([item,]prototypeStr)

inheritsFrom(), is Boolean `true` if a specific **prototypeStr** (note) is used by a note, either directly or through other prototypes. For a direct inheritance, the value of $Prototype can be checked but **inheritsFrom()** allows checking of inheritanc via multiple prototypes. Where a note X inherits from a prototype A but the latter itself uses a prototype B, **inheritsFrom()** allows testing if X inherits indirectly from B. As such inheritance occurs via A, it is otherwise difficult to test such inheritance. This is a specialist operator unlikely to be used except where prototypes themselves use other prototypes.

In a query, or find(), all notes are tested. In an action context (rule, edict, agent action, OnAdd, expression, etc.) only the current note (this) is evaluated. To query the document for notes inheriting from prototype 'pEvent', use the query term:

```
inheritsFrom("pEvent")
```

It is possible to test, via an action, the inheritance of a different note using the optional second scoping argument **item**. Thus to test if 'Note B' inherits from prototype 'pTask':

```
inheritsFrom("Note B","pTask")
```

For example, suppose prototype 'pFlower' has the prototype 'pPlant', and note 'Rose' uses the prototype 'pFlower'. Then

```
inheritsFrom("pPlant")
```

is `true` for both pFlower and pPlant.

```
inheritsFrom("pFlower")
```

is `true` for Rose, but `false` for pPlant.

N.B. note that the stated prototype is also included in the items testing `true`. This makes sense if actual notes in the document are prototypes, as opposed to using only deliberate 'non-content' prototypes. To filter the latter, and exclude all prototype notes, use a query like:

```
inheritsFrom("pFlower") & $IsPrototype==false
```

You can also write an offset test so note 'Rose' can test if note 'Camelia' uses prototype 'pFlower':

```
inheritsFrom("Camelia","pFlower")
```

This returns `true` if 'Camelia' inherits from that prototype. Or, 'Rose' might want to check if its parent uses that prototype:

```
inheritsFrom("parent","pFlower")
```

inside(item)

Operator Type:	Function [other Function type actions]
Operator Scope of Action:	Item [operators of similar scope]
Operator Purpose:	Query Boolean [other Query Boolean operators]
Operator First Added:	Baseline
Operator Last Altered:	As at baseline

---

inside(item)

Returns Boolean `true` if the current note is a direct child of **item** or, put conversely, if **item** is the parent of the note. Thus it can be thought of as an "is a child of" operator and as such a counterpart to the "is parent of" operator contains(). Note that aliases can be matched by inside(), which may appear counter-intuitive at first encounter: if only wanting to match original notes, see section 'Filtering out aliases' below.

The **item** argument must be quoted unless an attribute reference. Ways to define **item**.

Pertinent to map view, inside() and adornments:

- inside(item) is true for a note that is a child of the map's container **item**.
- inside(item) is true for a map note that is on top of (within or overlapping) the adornment **item**.
- inside(item) for a map container **item** does not match any child adornments.

In a more general context `inside("X")` is true for note A if any of the following are `true`:

- original A is inside X
- an alias of A is inside X
- A is an alias elsewhere, but its original is inside X

The last of these, if overlooked, can give unexpected results. The more expansive matching above is necessary to do things like looking inside agents.

If more than one container matching item is present, inside() returns items matching the first such container as listed in $OutlineOrder. If deliberately wanting to match notes inside any of several same-named containers, e.g. "exploded notes", then use `$Name(parent)=="container name"` instead.

Filtering out aliases

To match *only* originals, i.e. just the first of the three conditions in the previous list above, use `inside("X") & $IsAlias==false`.

Legacy issues

This operator replaces the legacy #inside query operator.

Interval.day()

Operator Type: Property [other Property type actions]
Operator Scope of Action: Item [operators of similar scope]
Operator Purpose: Date-time [other Date-time operators]
Operator First Added: Baseline
Operator Last Altered: As at baseline

Syntax note: *Operators without any defined mandatory arguments may omit their empty closing parentheses*

Interval.day()
Interval.day
Returns the Interval expressed as a Number of whole days. Thus, if $MyInterval is 1day 12 hours 30 minutes then:

```
$MyNumber = $MyInterval.day;
```

returns 1

Interval.format(formatStr)

Operator Type: Function [other Function type actions]
Operator Scope of Action: Item [operators of similar scope]
Operator Purpose: Formatting [other Formatting operators]
Operator First Added: Baseline
Operator Last Altered: As at baseline

Interval.format(formatStr)
For this data type **formatStr** strings can use date-type format strings. The function returns **Interval** as a String formatted as per the quoted date/interval format string **formatStr**.
This supplements the existing format() function.
When used with Interval data, **.format()** expects the interval to be less than one hours and thus comprising only minutes and seconds data. Further more, only two date format strings are accepted: `Interval.format ("l")` and `Interval.format ("L")` which format the mm:ss interval according to the current locale. The lower-case "l" format uses the locale's abbreviated form, while "L" spells out the interval in a phrase customised to local usage.
If $MyInterval is "12:55:23:"

```
$MyString = $MyInterval.format("l");
```

gives "12:55"

```
$MyString = $MyInterval.format("L");
```

gives "12 hours, 55 minutes"
An Interval value of "00:00" (minutes:seconds)—i.e. that data type's default—always returns an empty string. This if $MyInterval the default `00:00` then:

```
$MyString = $MyInterval.format("l");
```

results in "", i.e. no value is set in $MyString.
Formatting Intervals of over one hour
The expectation of input as being only mm:ss only can cause confusion when using **.format()**. If the value of $MyInterval is "12:55:40" (12 hours, 55 minutes and 14 seconds):

```
$MyString = $MyInterval.format("l")
```

The resulting $MyString value is "12:56", i.e. the seconds are rounded (up or down accordingly) and an hours:minuntes string is returned.
As Interval maximum scope is days/hours/minutes/seconds, if the aim is to get the entire interval duration, simply pass the Interval data direct to a string. This if $MyInterval is "1 day 12:55:23":

```
$MyString = $MyInterval;
```

sets $MyString to "1 day 12:55:23". If only part of the source is needed, a different approach is needed. Thus, if $MyInterval is "1 day 12:55:23" and only the number of whole minutes is needed, some string manipulation is required:

```
$MyString = $MyInterval.extract("(\d{2}):\d{2}$");
```

setting $MyString to "55". Note that regular expressions are very specific. If the closing '$' is omitted from the regex pattern, this results in a $MyString value of "12" instead of "55". Why? Because the latter pattern returns the first pair of digits followed explicitly by a colon and two more digits. Adding the '$' tells the regex that the literal sequence must come at the end of the source string, resulting in "55" being extracted.

Interval.hour()

Operator Type: Property [other Property type actions]
Operator Scope of Action: Item [operators of similar scope]
Operator Purpose: Date-time [other Date-time operators]
Operator First Added: Baseline
Operator Last Altered: As at baseline

Syntax note: *Operators without any defined mandatory arguments may omit their empty closing parentheses*

Interval.hour()
Interval.hour
Returns the Interval expressed as a Number of whole or partial hours within the current day (ignoring whole days). Thus, if $MyInterval is 12 hours 30 minutes then:

```
$MyNumber = $MyInterval.hour;
```

returns `12`

Interval.minute()

Operator Type: Property [other Property type actions]
Operator Scope of Action: Item [operators of similar scope]
Operator Purpose: Date-time [other Date-time operators]
Operator First Added: Baseline
Operator Last Altered: As at baseline

Syntax note: *Operators without any defined mandatory arguments may omit their empty closing parentheses*

Interval.minute()
Interval.minute
Returns the Interval expressed as a Number of whole or partial minutes within the current hour (ignoring whole hours). Thus, if $MyInterval is 1 hour, 30 minutes and 15 seconds then:

```
$MyNumber = $MyInterval.minute;
```

returns `30`
If $MyInterval is 1 hour, 30 minutes and 15 seconds then:

```
$MyNumber = $MyInterval.minute
```

returns `90.25`
To get an integer value, i.e. whole minutes, use either Number.precision() or floor(), thus:

```
$MyNumber = $MyInterval.minute.precision(0);
```

returns `90`

```
$MyNumber = floor($MyInterval.minute);
```

returns `90`

Interval.second()

Operator Type: Property [other Property type actions]
Operator Scope of Action: Item [operators of similar scope]
Operator Purpose: Date-time [other Date-time operators]
Operator First Added: Baseline
Operator Last Altered: As at baseline

Syntax note: *Operators without any defined mandatory arguments may omit their empty closing parentheses*

Interval.second()

Interval.second

Returns the Interval expressed as a Number of whole seconds within the current minute (ignoring whole minutes). Thus, if $MyInterval is 1 hour, 30 minutes and 15 seconds then:

```
$MyNumber = $MyInterval.second;
```

returns 15

Although date-time units are stored as milliseconds under the hood, Tinderbox does not return increments smaller than one second.

interval(dataStr)

Operator Type: Function [other Function type actions]
Operator Scope of Action: Item [operators of similar scope]
Operator Purpose: Date-time [other Date-time operators]
Operator First Added: Baseline
Operator Last Altered: As at baseline

interval(dataStr)

From v9.5.0, the operator interval(s) now converts a string to an interval. For example:

```
$MyInterval = interval("30:00")
```

is an interval of thirty minutes. The two-argument version, interval(start,end), continues to return the interval between two dates.

interval(startDate, endDate)

Operator Type: Function [other Function type actions]
Operator Scope of Action: Item [operators of similar scope]
Operator Purpose: Date-time [other Date-time operators]
Operator First Added: Baseline
Operator Last Altered: As at baseline

interval(startDate, endDate)

The function **interval(startDate,endDate)** returns, in Interval data-type form, the time interval between two Date-type dates. For example, to test the interval between a note's initial creation and its most recent modification and to then store it in an Interval-type attribute:

```
$MyInterval = interval($Created,$Modified);
```

The interval() function can be used to test the exact equivalence of two Date-type attribute values, using the full Date and time. Thus:

```
if(interval($DateA,$DateB)=="00:00"){…
```

The latter gets around the fact that `==` and `!=` operators match Date-type attributes only at day scope rather than the actual date-time values.

isbn10(dataStr)

Operator Type: Function [other Function type actions]
Operator Scope of Action: Item [operators of similar scope]
Operator Purpose: Data manipulation [other Data manipulation operators]
Operator First Added: Baseline
Operator Last Altered: As at baseline

isbn10(dataStr)

takes quoted an ISBN-13 format code as the **dataStr** input and returns is in ISBN-10 format. Any dashes or other punctuation will be ignored. If the argument is not a valid ISBN code, the function returns an empty string. If such characters as in the source string and quotes are omitted, hyphen may be mis-parsed as minus signs giving a wrong result.

Consider a book with the ISBN-10 '1472268997' and the ISBN-13 '978-1472268990':

`$MyString = isbn10("978-1472268990")` gives the correct '1472268997' but note, without any hyphens.

be aware that if **dataStr** is not in quotes, the input is treated as an arithmetical expression:

`$MyString = isbn10("978-1472268990")` gives the *incorrect* value '1472268012'.

isbn13(dataStr)

Operator Type: Function [other Function type actions]
Operator Scope of Action: Item [operators of similar scope]
Operator Purpose: Data manipulation [other Data manipulation operators]
Operator First Added: Baseline
Operator Last Altered: As at baseline

isbn13(data)

takes an ISBN-10 format code as the quoted string **dataStr** and returns it in ISBN-13 format. Any dashes or other punctuation will be ignored. If the argument is not a valid ISBN code, the function returns an empty string. If such characters as in the source string and quotes are omitted, hyphen may be mis-parsed as minus signs giving a wrong result.

Consider a book with the ISBN-10 '1472268997' and the ISBN-13 '978-1472268990':

`$MyString = isbn13("1472268997")` gives the correct '9781472268990' but note, without any hyphens.

be aware that if **dataStr** is not in quotes, result is the same.

isDuplicateName(item)

Operator Type: Property [other Property type actions]
Operator Scope of Action: Item [operators of similar scope]
Operator Purpose: Non-query Boolean [other Non-query Boolean operators]
Operator First Added: Baseline
Operator Last Altered: 9.5.0

isDuplicateName(item)

This is `true` if another note in the document has the same $Name as this note.

This can be useful if trying to find/resolve notes with duplicate names within a document. If expecting to use action code to work with notes, it helps if a note's title ($Name) is unique.

From v9.5.0, **isDuplicateName()** ignores all aliases. Previously, testing aliases resulted in false reports of duplicate names.

JSON.each([pathStr]){actions}

Operator Type: Function [other Function type actions]
Operator Scope of Action: List [operators of similar scope]
Operator Purpose: Stream parsing [other Stream parsing operators]
Operator First Added: Baseline
Operator Last Altered: 9.6.0

JSON.each([pathStr]){action(s)}

JSON.each{action(s)}

If the top-level element is an array, rebinds the JSON object in turn to each array element. After calling the action block for each element, the JSON object is restored.
For example, if the value of $MyString is `[{"price":1}, {"price",2}]`, then

```
$MyList=[];
$MyString.json.each{$MyList += json["price"]*3;}
```

would set $MyList to "3;6".

JSON.each(pathStr){action(s)}

In **.json.each(){...}**, an optional path argument, **pathStr**, supplies a path to the array to be iterated. For example, if $Text is:

```
{
   "person": { "firstName": "Thomas",  lastName: "Roe"},
   "coordinates" : [-90,41]
}
```

then `$Text.captureJSON().json.each(coordinates){…}` would iterate through the array of coordinates.
If String.json.each begins a statement,

```
    $Text.json.each(coordinates){...}
```

.json.each(){} reuses the current JSON object. This can be much faster than repeatedly re-parsing a complex json package.
As a result, the older syntax `$Text.json[coordinates].each(x){…}`, that chained of JSON.json[keyStr] is no longer supported.

Loading a dictionary of dictionaries from JSON and looking up items

Let $Text be:

```
{ "French":{ "child":"enfant"; "cat":"chat"}; "Swedish":{"child":"barn";"cat":"katt"} }
```

Now:

`$Text.json[French][cat]` is "chat"
`$Text.json["Swedish"]["cat"]` is "katt"
`$Text.json.keys` is "French; Swedish"

```
$Text.json.keys.each(x){
 $MyList=$MyList+$Text.json[x][cat];
};
```

$MyList is `[chat;katt]`.

Loading a list of dictionaries from JSON a and looking up items

Let $Text be:

```
[ { "child":"enfant"; "cat":"chat"} , {"child":"barn";"cat":"katt"} ]
```

Now:

`$Text.json[0][cat]` is chat
`$Text.json[0]['cat']` is katt
`$Text.json.count` is 2

```
$Text.json.each(x){
 $MyList += x[child];
}
```

$MyList is now `[enfant;barn]`.

JSON.json[itemNum]

Operator Type:	Function [other Function type actions]
Operator Scope of Action:	Item [operators of similar scope]
Operator Purpose:	Stream parsing [other Stream parsing operators]
Operator First Added:	Baseline
Operator Last Altered:	As at baseline

JSON.json[itemNum]

If there is no current JSON object, attempts to parse the string as JSON and fails if unsuccessful. If there is a current JSON object, that object will be reused.
If the top-level element is an array, Stream.json[N] returns the **itemNum** object. If the top-level element is an object, see JSON.json[keyStr].
For example, if the $Text is:

```
{ [1,4,9,16,25] }
```

Then $Text.json[1] is 4.
This usage follows the existing List/Set[N] convention.

JSON.json[keyStr]

Operator Type:	Function [other Function type actions]
Operator Scope of Action:	Item [operators of similar scope]
Operator Purpose:	Stream parsing [other Stream parsing operators]
Operator First Added:	Baseline
Operator Last Altered:	9.6.0

JSON.json[keyStr]

If there is no current JSON object, attempts to parse the string as JSON and fails if unsuccessful. If there is a current JSON object, that object will be reused. **keyStr** is a quoted key name.
If the top-level element is an object, Stream.json[key] returns a dictionary for that object. If the top-level element is an array, see JSON.json[itemNum].
For example if $Text is:

```
{ "title":"Becket", "price": 9.95 }
```

and $Subtitle is set to "title", then:

`$Text.json["title"];` is "Becket".
`$Text.json[title];` is "Becket".
`$Text.json["price"];` is "9.95".
`$Text.json['$Subtitle'];` no such field.
`$Text.json[$Subtitle];` is "Becket".
Though from v9.6.0 multiple bracketed arguments can be used to address a JSON path, consider use of .jsonValue(pathStr) instead. For example:

```
Text:  {
  "person": { "firstName": "Thomas", "lastName": "Roe" },
  "coordinates": [-90,41]
}
MyString: "person.lastName"
```

Then:

`$Text.json['person']['lastName']` gives 'Roe'
But, easier:
`$Text.json.jsonValue(person.lastName)` gives 'Roe'
`$Text.json.jsonValue($MyString)` gives 'Roe' (from v9.6.0)
Again, using the same JSON as above:

```
$MyString="coordinates";
```

Now
`$Text.json['coordinates'][0]` gives -90 (from v9.6.0)
`$Text.json[$MyString][1]` gives 41 (from v9.6.0)

JSON.jsonValue(pathStr)

Operator Type: Function [other Function type actions]
Operator Scope of Action: Item [operators of similar scope]
Operator Purpose: Stream parsing [other Stream parsing operators]
Operator First Added: 9.6.0
Operator Last Altered: As at baseline

JSON.jsonValue(pathStr)

New to v9.6.0, .jsonValue() allows a json object to be addressed by its JSON path. By comparison .json[keyStr] must use a key name. For example:

```
Text:  {
  "person": { "firstName": "Thomas", "lastName": "Roe" },
  "coordinates": [-90,41]
}
MyString: "person.lastName"
```

Then:

`$Text.json['person']['lastName']` gives 'Roe'

But, easier:

`$Text.json.jsonValue(person.lastName)` gives 'Roe'

`$Text.json.jsonValue($MyString)` gives 'Roe' (in v9.6.0+)

jsonEncode(dataStr)

Operator Type: Function [other Function type actions]
Operator Scope of Action: Item [operators of similar scope]
Operator Purpose: Formatting [other Formatting operators]
Operator First Added: Baseline
Operator Last Altered: As at baseline
Operator Has Newer Dot-Operator Variant: Yes

---

jsonEncode(dataStr)

This operator returns a JSON-encoded UTF-8 version of the **dataStr** string. The apostrophe (straight single quote), straight double quote, solidus (forward slash) and backslash characters are all escaped by a preceding backslash character. The control characters backspace, form feed, new line, carriage return, horizontal tab are encoded as standard JavaScript escapes (\b, \f, \n, \r\, \t). Unicode addresses are also escaped: \u2345.

Note: the single apostrophe (straight quote) is not escaped as this can cause some Ajax functions to fail.

For string-type data, see also the '.json' operator.

jsonEncode() does not escape single straight quotation marks.

last(item[, childrenNum])

Operator Type: Function [other Function type actions]
Operator Scope of Action: Item [operators of similar scope]
Operator Purpose: Query Boolean [other Query Boolean operators]
Operator First Added: Baseline
Operator Last Altered: As at baseline
Operator Has Optional Arguments: [More on optional operator arguments]
Operator Has Newer Dot-Operator Variant: Yes

---

last(item[item[, childrenNum])

Returns Boolean `true` if the current note is among the last **childrenNum** of children of **item**. If **childrenNum** is missing, a value of 1 is assumed.

The **item** argument must be quoted unless an attribute reference. Ways to define **item**.

Both arguments are evaluated and can be a literal string/number, an attribute value or an action code expression evaluating to that same.

If the current note has a $SiblingOrder value of 7, then if first() is run on its parent container with 10 children:

`first("Note A", 5)` returns `true`

but if it has a $SiblingOrder value of 2:

`first("Note A", 5)` returns `false`

last() also has a logical opposite in first().

Legacy issues

This operator replaces the legacy #last query operator.

lastWord(dataStr)

Operator Type: Function [other Function type actions]
Operator Scope of Action: Item [operators of similar scope]
Operator Purpose: Data manipulation [other Data manipulation operators]
Operator First Added: Baseline
Operator Last Altered: As at baseline

---

lastword("data")

The **lastWord()** operator has one argument, **dataStr** (a quoted a string), and it returns the final word of the data string. The delimiter used to define words is one or more spaces (possibly also line break(s)?).

The **dataStr** argument is evaluated so could be an expression. For example, if the note 'First Line' has the body text "Winter is coming.", then

```
$MyString = lastWord($Text("First Line"));
```

should give a result of "`coming`".

linkedFrom(scope[, linkTypeStr])

Operator Type: Function [other Function type actions]
Operator Scope of Action: Item [operators of similar scope]
Operator Purpose: Query Boolean [other Query Boolean operators]
Operator First Added: Baseline
Operator Last Altered: As at baseline
Operator Uses Scoped Arguments: [More on scoped arguments in Action Code]
Operator Has Optional Arguments: [More on optional operator arguments]

---

linkedFrom(scope[, linkTypeStr])

A test for inbound links. This boolean test at returns `true` if the current note has a least one link *from any of* the note(s) defined by **scope**; this is optionally filtered to only links of type **linkTypeStr**. Put another way:

- "Does an *inbound link* exist to the current note from any of **scope**'s item(s)?".

or

- "Do any of **scope**'s item(s) have an *outbound link* to the current note?".

As this is effectively only a query term, (with a Boolean result), if trying to collect data about the linked note(s), use links() instead.

scope defines a group of items **in a number of ways**. An additional option to normal group descriptors is a wildcard designator * that matches *all* notes in the document and replaces the normal "all" group designator.

For **linkTypeStr**, links of type 'prototype' are ignored. Used in an agent, 'this' note is the alias in the agent and not its original, making this action unsuitable for testing in an agent action. If using **linkTypeStr**, you must use the value `"*untitled"` to match an 'untitled' type link (rather than an empty string, "", or "untitled").

Thus, to test if any note using the 'Event' prototype has an inbound link of the 'untitled' link type the agent query would be:

```
$Prototype=="Event" & linkedFrom("*", "*untitled")
```

The logical opposite of this test is linkedTo().

This function can match a link from an alias as opposed to an original (if the logical choice).

Working with aliases

If testing links using aliases (e.g. those created by an agent query) be aware that, for basic type links only, the original and alias can and may have differing numbers of basic links. Thus if wishing to check, unambiguously, the original's links from the context of one of its aliases, use originalLinkedFrom() instead.

Legacy issues

This replaces the legacy #linkedFrom query operator (deprecated since v4.6).

linkedTo(scope[, linkTypeStr])

Operator Type:	Function [other Function type actions]
Operator Scope of Action:	Item [operators of similar scope]
Operator Purpose:	Query Boolean [other Query Boolean operators]
Operator First Added:	Baseline
Operator Last Altered:	As at baseline
Operator Uses Scoped Arguments:	[More on scoped arguments in Action Code]
Operator Has Optional Arguments:	[More on optional operator arguments]

---

linkedTo(scope[, linkTypeStr])

A test for outbound links. This boolean test at returns `true` if the current note has a least one link *to any of* the note(s) defined by **scope**; this is optionally filtered to only links of type **linkTypeStr**. Put another way:

- "Does an *outbound link* exist from the current note to any of **scope**'s item(s)?".

or

- "Do any of **scope**'s item(s) have an *inbound link* from the current note?".

As this is effectively only a query term, (with a Boolean result), if trying to collect data about the linked note(s), use links() instead.

scope defines a group of items **in a number of ways**. An additional option to normal group descriptors wildcard designator * that matches *all* notes in the document and replaces the normal "all" group designator.

For **linkTypeStr**, links of type 'prototype' are ignored. Used in an agent, 'this' note is the alias in the agent and not its original, making this action unsuitable for testing in an agent action. If using **linkTypeStr**, you must use the value `"*untitled"` to match an 'untitled' type link (rather than an empty string, "", or "untitled").

Thus, to test if any note using the 'Event' prototype has an outbound link of the 'untitled' link type the agent query would be:

```
$Prototype=="Event" & linkedTo("*", "*untitled");
```

The logical opposite of this test is linkedFrom().

This function can match a link from an alias as opposed to an original (if the logical choice).

Working with aliases

If testing links using aliases (e.g. those created by an agent query) be aware that, for basic type links only, the original and alias can and may have differing numbers of basic links. Thus if wishing to check, unambiguously, the original's links from the context of one of its aliases, use originalLinkedTo() instead.

Legacy issues

This replaces the legacy #linkedTo query operator (deprecated since v4.6).

linkFrom(scope[, linkTypeStr])

Operator Type:	Function [other Function type actions]
Operator Scope of Action:	Group [operators of similar scope]
Operator Purpose:	Linking [other Linking operators]
Operator First Added:	Baseline
Operator Last Altered:	As at baseline
Operator Uses Scoped Arguments:	[More on scoped arguments in Action Code]
Operator Has Optional Arguments:	[More on optional operator arguments]

---

linkFrom("item|group"[,"linkType"])

This creates an untitled type basic link from **scope** to the current note (i.e. an inbound link).

The **scope** argument must be quoted unless an attribute reference, e.g. `"Some note"` vs. `$MyString`. Ways to define **scope**.

The **scope** can be **group** scoped including use of group designators and operators like find,() collect() and links().

linkTypeStr (string). Optionally, instead of an 'untitled' link the link can be of **linkTypeStr** type. An untitled type link can be explicitly specified using the string "*untitled". Unlike unlinking, this argument may only contain a single link type value

Both arguments are evaluated. This operator does not require a left-side argument, simply calling effects a result. A new link *will not be created* if a link of the stated type already exists.

Examples

Linking to a note "Some note":

No link type: `linkFrom("Some note")`

Link type 'agree': `linkFrom("Some note","agree");`

Linking to the first child (via a designator):

```
linkFrom(child);
linkFrom(child,"agree");
```

Relevant similar operators: linkTo, unlinkTo, unlinkFrom.

Use of this action does not shift note focus; in addition if **scope** contains operators (brackets, plus, minus, etc.) Tinderbox will first look for a match to the literal **item** string and only if there is no match will the app try evaluating to operators and testing the resulting string. For example:

```
linkFrom("Example 1 (a test)");
```

will link from the note named 'Example 1 (a test)'. If no note matches this string, Tinderbox will attempt to evaluate the string. Thus for:

```
linkFrom("2+2")
```

will link from the note named '2+2' but if no match will look for a note named '4'.

This function can link from an alias as opposed to an original (if the logical choice) and can accept a group scope.

Use in agents

Beware that the action is working on an *alias* of the current note and note the current note itself. As originals and aliases support discrete basic links this function should not generally be used in an agent. The best way to use the function is by using a prototype and apply a $Rule to it thus running the code in all notes using the prototype.

An alternative linkFromOriginal() code will ensure any link created is between two original notes regardless of whether an alias is the context of execution of the code.

linkFromOriginal(scope[, linkTypeStr])

Operator Type:	Function [other Function type actions]
Operator Scope of Action:	Group [operators of similar scope]
Operator Purpose:	Linking [other Linking operators]
Operator First Added:	Baseline
Operator Last Altered:	As at baseline
Operator Uses Scoped Arguments:	[More on scoped arguments in Action Code]
Operator Has Optional Arguments:	[More on optional operator arguments]

---

linkFromOriginal(scope[, linkTypeStr])

This function works exactly as the same as linkFrom(), *except* for one important difference that the link created is *always between two originals* even if either/both the evaluated source or destination are an alias.

For more detail of use, see linkFrom().

See also linkToOriginal(), unlinkFromOriginal(), unlinkToOriginal().

linkPath(pathNameStr[, startStr, endStr])

Operator Type: Function [other Function type actions]
Operator Scope of Action: Item [operators of similar scope]
Operator Purpose: Linking [other Linking operators]
Operator First Added: Baseline
Operator Last Altered: As at baseline
Operator Has Optional Arguments: [More on optional operator arguments]

---

linkPath(pathNameStr)

linkPath(pathNameStr[, startStr, endStr])

These functions return a list of notes that are on a designated path. The **pathNameStr** input must be supplied but can use a value "*" designates any path, regardless of path name—i.e. all notes that have at least one inbound or outbound link. This operator in many ways mirrors the visual function of the Hyperbolic view.

If **startStr** is provided, that path starts at the designated note and ends on reaching the **endStr** note or when all links on the path reachable from start are exhausted. If only the pathName is provided, all notes on the path are listed, whether c not they all form one contiguous network.

For example, to collect all notes on the path "example":

```
$MyList = linkPath("example");
```

Or, to return all notes connected by links of any type:

```
$MyList = linkPath("*");
```

To find notes linked by the link type "Project A", starting at "RFC" :

```
$MyList = linkPath("Project A", "RFC");
```

To find notes linked by the link type "Project A", starting at "RFC" and ending at "Archive":

```
$MyList = linkPath("Project A", "RFC", "Archive");
```

If several possible paths exist from **startStr** to **endStr**, Tinderbox will return the shortest path, or at least a path which is not longer than any other path. Bear in mind that in richly interlinked documents, there may be no 'obvious' single path between the specified notes.

If only **endStr** is supplied, still include the argument delimiter for the unused **startStr**:

```
$MyList = linkPath("Project A", , "Archive");
```

The underlying graph may contain cycles (i.e. have looping paths).

links[(scope)].[directionStr].[linkTypeRegex].attributeNameRefStr

---

Operator Type: Function [other Function type actions]
Operator Scope of Action: Conditional Group [operators of similar scope]
Operator Purpose: Dictionary, Set & List operations [other Dictionary, Set & List operations operators]
Operator First Added: Baseline
Operator Last Altered: As at baseline
Operator Uses Regular Expressions: [More on regular expressions in Tinderbox]
Operator Uses Scoped Arguments: [More on scoped arguments in Action Code]
Operator Has Optional Arguments: [More on optional operator arguments]

---

links[(scope)].[directionStr].[linkTypeRegex].attributeNameStr

The **links()** operator builds a List from a collection of links. It selects the note(s) whose links should be inspected. The **scope** is one or more items (defining **scope**), but note the **scope** argument is not fully evaluated, so only use simple expressions. **scope** never matches aliases. When using links() in the context of an agent's action, remember that aliases can have different basic links to their originals. Therefore, it is likely that act action using 'this' will want to replace it with 'original' as the **scope** when re-used in an agent action.

The **directionStr** argument filters the directionality of the links collected. It is mandatory, and should *not* be quoted. It can only be is one of these values:

- inbound
- outbound

The optional **linkTypeRegex** argument is evaluated as a regular expression. Although the most normal usage will be as a literal string of a single link type name. Regex use allows for It collects only links of a specified link type, or such type(s) as match the **linkTypeRegex** regex amongst the link type names defined in the current TBX. Regular expression wild-card characters are permitted and retain their special meanings. If the **linkTypeStr** value contains white space or periods, must be enclosed in double quotes:

```
links.outbound."responds to".$Name
```

If **linkTypeRegex** is left empty, links of all types are collected except prototype links. *Prototype links are always omitted*. Single quotes can be used to enclose **linkTypeRegex** but if the quoted string includes a single quote this must be backslash-escaped or double quote used instead:

```
links.outbound."Peter's place".$Name OK
links.outbound.'Peter's place'.$Name wrong
links.outbound.'Peter\'s place'.$Name OK
```

The is evaluated for regex.

The **attributeNameRefStr** argument is the $-prefixed reference to the attribute whose values are to be collected in the result. An attribute reference, e.g. $Name("nextSibling") is invalid, the command work but the reference is ignored and the stated attribute for the linked note is used, i.e. **attributeNameRefStr** is a literal value and cannot be an expression.

If simply wishing to test the state of links between two items, consider the Boolean queries linkedFrom() and linkedTo().

Examples

```
$MyList=links(/config).outbound.supports.$Name;
```

constructs a set of all the titles (from Name attribute) of notes that are linked to the top-level note named 'config' via links with the link type 'supports'. This does the same but for all link types;

```
$MyList=links(/config).outbound..$Name;
```

For multi-word link type names use quotes (or if using regex characters):

```
$MyList=links(/config).outbound."agrees with".$Name;
```

Whilst it is likely that 'Name' will be the most usual value for **attribute**, it can be *any* currently defined attribute:

```
$MyList=links.inbound."went to".$SchoolName;
```

collects a list of values of the attribute 'SchoolName' for notes that have an inbound link to the current note of link type "went to".

Beware when using a TBX that has notes with duplicate (same) $Name values. As a set contains unique values, if several notes have identical names, then

```
$MySet=links.inbound..$Name;
```

will list the distinct Names only once in MySet, and so the latter will have fewer values than the actual number of matching links. In the same scenario:

```
$MyList=links.inbound..$Name;
```

will create a list containing duplicates.

The format() or List.format() operators can help make more use of links() data during export, e.g. as lists or lists of links. Internally, if analysing links and there is no real need to keep set-type data, using agents employing linkedTo() and linkedFrom() will find most of the same data as links() can provide.

The links() function can be chained by dot operators pertinent to use with List type data. But, as links() uses dot-chained *arguments*, it is necessary to use parentheses to chain other dot operators. Thus, to get a count of the number of linke items found:

```
(links(children).inbound..$StartDate).size
```

In addition, the use of parentheses helps make sense of the intended order of execution of the tasks chained to **links()**:

```
$MyList=(links.inbound."colleague of".$Name).sort("$StartDate");
```

More examples of links() syntax:

```
links.outbound.agree.$Name
links(this).outbound.agree.$Name
links(children).inbound..$StartDate
links("A note").outbound."example|agree".$Name
links("A note;A different note").inbound."*untitled".$Path
links($MyString).outbound..$TutorName
links(find(descendedFrom("Some note")).inbound.my_link.$SomeAttribute
links(find(descendedFrom("Some note")&$MyDate==$StartDate).outbound..$Width
links($MySet).outbound.example.$Name
links($MyList).inbound.note.$WordCount
```

Working with links

If needing to return multiple items from linked items or to do more complex link-based work, see the eachLink() operator which offers a wider range of options.

linkTo(scope[, linkTypeStr])

Operator Type:	Function [other Function type actions]
Operator Scope of Action:	Group [operators of similar scope]
Operator Purpose:	Linking [other Linking operators]
Operator First Added:	Baseline
Operator Last Altered:	As at baseline
Operator Uses Scoped Arguments:	[More on scoped arguments in Action Code]
Operator Has Optional Arguments:	[More on optional operator arguments]

---

linkTo(scope[, linkTypeStr])

This creates an untitled type basic link to **scope** from the current note (i.e. an outbound link).

The **scope** argument must be quoted unless an attribute reference, e.g. `"Some note"` vs. `$MyString`. Ways to define **scope**.

scope can be **group** scoped including use of group designators and operators like find,() collect() and links().

linkType (string). Optionally, instead of an 'untitled' link the link can be of **linkTypeStr** type. An untitled type link can be explicitly specified using the string ""untitled". Unlike unlinking, this argument may only contain a single link type value.

Both arguments are evaluated. This operator does not require a left-side argument, simply calling effects a result. A new link *will not be created* if a link of the stated type already exists.

Examples

Linking to a note "Some note":

No link type: `linkTo("Some note");`

Link type 'agree': `linkTo("Some note","agree");`

Linking to the first child (via a designator):

```
linkTo(child);
linkTo(child,"agree");
```

Relevant similar operators: linkFrom, unlinkTo, unlinkFrom.

Use of this action does not shift note focus; in addition if **scope** contains operators (brackets, plus, minus, etc.) Tinderbox will first look for a match to the literal **item** string and only if there is no match will the app try evaluating to operators and testing the resulting string. For example:

```
linkTo("Example 1 (a test)")
```

will link to the note named 'Example 1 (a test)'. If no note matches this string, Tinderbox will attempt to evaluate the string. Thus for:

```
linkTo("2+2");
```

will link to the note named '2+2' but if no match will look for a note named '4'.

This function can link to an alias as opposed to an original (if the logical choice) and can accept a group scope.

Use in agents

Beware that the action is working on an *alias* of the current note and note the current note itself. As originals and aliases support discrete basic links this function should not generally be used in an agent. The best way to use the function is by using a prototype and apply a $Rule to it thus running the code in all notes using the prototype.

An alternative linkFromOriginal() code will ensure any link created is between two original notes regardless of whether an alias is the context of execution of the code.

linkToOriginal(scope[, linkTypeStr])

Operator Type:	Function [other Function type actions]
Operator Scope of Action:	Group [operators of similar scope]
Operator Purpose:	Linking [other Linking operators]
Operator First Added:	Baseline
Operator Last Altered:	As at baseline
Operator Uses Scoped Arguments:	[More on scoped arguments in Action Code]
Operator Has Optional Arguments:	[More on optional operator arguments]

---

linkToOriginal(scope[, linkType])

This function works exactly as the same as linkTo(), *except* for one important difference that the link created is *always between two originals* even if either/both the evaluated source or destination are an alias.

For more detail of use, see linkTo().

See also linkFromOriginal(), unlinkFromOriginal(), unlinkToOriginal().

List.isort([attributeRefStr])

Operator Type:	Function [other Function type actions]
Operator Scope of Action:	Item [operators of similar scope]
Operator Purpose:	Data manipulation [other Data manipulation operators]
Operator First Added:	Baseline
Operator Last Altered:	As at baseline
Operator Has Optional Arguments:	[More on optional operator arguments]

Syntax note: *Operators without any defined mandatory arguments may omit their empty closing parentheses*

List.isort([attributeRefStr])

List.isort()

The basic form

```
$MyList = $MyList.isort();
```

The operator can be chained with other dot-operators:

```
$MyList = $MyList.isort().reverse();
```

This function re-sorts the referenced list's values in lexical, case-insensitive order. This means sorting letters in alphabetical order "a,b,c'" with all upper case instances sorting before any lower case ones as in "ant;Ant;bee;Bee;cow;Cow". Lexical sort of numbers means '1,11,2' not 1,2,11' as might be expected.

Alternative sorts are a lexical case-sensitive List.sort(), and a numeric List.nsort(). Or, if using $Name or $Path data, use the optional long from (below) and let the nominated attribute's data type set the form of sort, e.g. for date sorting, sort on a Date-type attribute.

The resulting sort order can be reversed overall by chaining the List.isort() and List.reverse() functions.

This operator does not apply to Set-type lists because as from v9.0.0 Sets now auto-sort and cannot be (reliably) user-sorted to a different order.

List.isort($AttributeRefStr)

Here the referenced list *must be a list of note names or paths* , i.e. list $Name data or $Path data (but not a mix of the two). If note names are not unique within the document, path values must be used to achieve correct sort. This form of sort cannot be used with any other sort of value list. This operator cannot be applied to any list of values, unlike the short form above.

In this form, the sort order can be based on the value of the stipulated **AttributeRefStr** (a $-prefixed attribute name). The attribute's value is derived from the item identified by the $Name or $Path of the list item being evaluated:

```
$MyList = $collect(children, $Name).isort($StartDate);
$MyList = $collect_if(find($Prototype=="pArticle"), $Year!=,$Path).isort($Year);
```

In this long form usage, the sort ordering is based on the data type of **AttributeRefStr**:

- Number type: numerical sort
- Date type: sort on date (implied numerical)
- All other types lexical

With .isort() lexical sorts are *always* case-insensitive.

List.nsort([attributeRefStr])

Operator Type:	Function [other Function type actions]
Operator Scope of Action:	Item [operators of similar scope]
Operator Purpose:	Data manipulation [other Data manipulation operators]
Operator First Added:	Baseline
Operator Last Altered:	As at baseline
Operator Has Optional Arguments:	[More on optional operator arguments]

Syntax note: *Operators without any defined mandatory arguments may omit their empty closing parentheses*

List.nsort([attributeRefStr])

List.nsort()

The basic form

```
$MyList = $MyList.nsort();
$MyList = $MyList.nsort().reverse();
```

This function re-sorts the referenced list's values in ascending numerical sort order. This means sorting '1,2,11' not 1,11,2' such as occurs with sort() and a lexical sort. A numerical sort is really only useful for numerical data. For text, consider List.sort() or List.isort(), both of which give lexical sorts of differing case sensitivity. Or, if using $Name or $Path data, use the optional long from (below) and let the nominated attribute's data type set the form of sort, e.g. for date sorting, sort c a Date-type attribute.

The resulting sort order can be reversed overall by chaining the List.nsort() and List.reverse() functions.

This operator does not apply to Set-type lists because as from v9.0.0 Sets now auto-sort and cannot be (reliably) user-sorted to a different order.

List.nsort(attributeRefStr)

Here the referenced list *must be a list of note names or paths* , i.e. list $Name data or $Path data (but not a mix of the two). If note names are not unique within the document, path values must be used to achieve correct sort. This form of sort cannot be used with any other sort of value list. This operator cannot be applied to any list of values, unlike the short form above.

In this form, the sort order can be based on the value of the stipulated **attributeRefStr** (a $-prefixed attribute name). The attribute's value is derived from the item identified by the $Name or $Path of the list item being evaluated:

```
$MyList = $collect(children, $Name).nsort($PageNumber);
$MyList = $collect_if(find($Prototype=="pProduct"), $Price>0,$Path).nsort($Price);
```

In this long form usage, the sort ordering is based on the data type of **attributeRefStr**:

- Number type:- numerical sort
- Date type: sort on date (implied numerical)
- All other types lexical

List.select()

Operator Type:	Function [other Function type actions]
Operator Scope of Action:	Group [operators of similar scope]
Operator Purpose:	Data manipulation [other Data manipulation operators]
Operator First Added:	9.6.0
Operator Last Altered:	As at baseline
Operator Uses Scoped Arguments:	[More on scoped arguments in Action Code]

List.select()

From v9.6.0, the **List.select()** operator allows the document's current (UI) focus to be shifted, to use the item(s)m defined in **List**. The **List** may be a literal string, or the value of an attribute or variable.

```
[/Note A;/Note B].select();
$MyList.select();
vItems.select();
```

This specialist operator assist in the scenario where, whilst running action code, it is necessary to change the selection such that the locus of 'this' changes. Some action code operators only address the currently selected item(s). The **select(scope)** operator allows the selection to be changed on the fly without the user having to do so via the UI.

Because using select without arguments has a distinct and different function is it listed separately: select().

List.sort([attributeRefStr])

Operator Type:	Function [other Function type actions]
Operator Scope of Action:	Item [operators of similar scope]
Operator Purpose:	Data manipulation [other Data manipulation operators]
Operator First Added:	Baseline
Operator Last Altered:	As at baseline
Operator Has Optional Arguments:	[More on optional operator arguments]

Syntax note: *Operators without any defined mandatory arguments may omit their empty closing parentheses*

List.isort([attributeRefStr])

List.sort()

The basic form

```
$MyList = $MyList.sort();
$MyList = $MyList.sort().reverse();
```

This function re-sorts the referenced list's values in lexical, case-sensitive order. This means sorting *all* upper case letters before any lower case ones as in "Ant;Bee;Cow;ant;bee;cow". Users of non-accented languages (for the content of the notes) will likely find the case-insensitive List.isort() of more practical use. A forced numerical sort (1,2,11 not 1,11,2) can be achieved by using List.nsort(). Or, if using $Name or $Path data, use the optional long from (below) and let the nominated attribute's data type set the form of sort, e.g. for date sorting, sort on a Date-type attribute.

The resulting sort order can be reversed overall by chaining the List.sort() and List.reverse() functions.

Sets can be sorted, on the fly, so a sorted output can be passed to a List. Passing the sorted output of a Set to another Set has an unknown outcome, in terms of stored sort order.

List sorting can be used to process group designators. For example:

```
$MyList=$Colors(children);
```

finds a list of the colours of each child of the current note, and:

```
$MyNumber=$Width(children).max;
```

will find the maximum width of the current container's children. When applied to the attribute $Text:

```
$Text=$Text(children);
```

the texts of each child are appended, separated by paragraph breaks.

This operator does not apply to Set-type lists because, as from v9.0.0, Sets now auto-sort and cannot be (reliably) user-sorted to a different order.

List.sort(attributeRefStr)

Here the referenced list *must be a list of note names or paths* , i.e. list $Name data or $Path data (but not a mix of the two). If note names are not unique within the document, path values must be used to achieve correct sort. This form of sort cannot be used with any other sort of value list. This operator cannot be applied to any list of values, unlike the short form above.

In this form, the sort order can be based on the value of the stipulated **attributeRefStr** (a $-prefixed attribute name). The attribute's value is derived from the item identified by the $Name or $Path of the list item being evaluated:

```
$MyList = collect(children, $Name).sort($StartDate);
$MyList = collect_if(find($Prototype=="pArticle"), $Year!=,$Path).sort($Year);
```

In this long form usage, the sort ordering is based on the data type of **attributeRefStr**:

- Number type: numerical sort
- Date type: sort on date (implied numerical)
- All other types lexical

With .sort() lexical sorts are *always* case-insensitive.

Group designators are allowed in attribute references. For example:

```
$MyList=$Colors(children);
```

finds a list of the colours of each child of this note, and:

```
$MyNumber=$Width(children).max;
```

will find the maximum width of the container's children. When applied to the attribute $Text:

```
$Text=$Text(children);
```

the texts of each child are appended, separated by paragraph breaks.

List.unique()

Operator Type:	Function [other Function type actions]
Operator Scope of Action:	Item [operators of similar scope]
Operator Purpose:	Data manipulation [other Data manipulation operators]
Operator First Added:	Baseline
Operator Last Altered:	9.5.0

Syntax note: *Operators without any defined mandatory arguments may omit their empty closing parentheses*

List.unique()

List.unique

This returns a List of the unique values in the list, as a *sorted* but de-duped list. Hitherto, de-duping required passing data into a Set-type attribute and back. Trailing parentheses are optional for this function.

Be aware that this function results in a case insensitive A-Z sort, thus not 'bee;ant;cow' as might otherwise be assumed (to get such an outcome see further below). For example, if $MyList is 'bee;ant;cow;bee':

```
$MyList = $MyList.unique;
```

results in a list value 'ant;bee;cow'.

The function can be chained with .sort-type actions and .reverse.

```
$MyList = $MyList("Another note").unique;
$SomeList = collect(children, $MyNumberList).unique.nsort;
$MyList = collect(children, $MyList).unique.reverse;
```

The last above sets $MyList to a list of all the unique, discrete, values to be found in $MyList in every child of the current note. Use with collect() or collect_if() to act on a particular attribute across a group of notes. If a collect() with query scc is the designator 'all' the result will be every discrete value for the referenced list attribute across the whole document.

This function does not apply to Set-type lists because Sets automatically de-duplicate items so are always a list of unique values.

De-duping a list whilst retaining original sort order

The basic method is this:

```
$MyList.each(anItem){
   if(!$MyList2.contains(anItem)){
      $MyList2+=anItem;
   }
};
```

If is it desired to de-dupe $MyList back to itself, use a list-type variable:

```
var:list vList;
$MyList.each(anItem){
   if(!vList.contains(anItem)){
      vList+=anItem;
   }
};
$MyList = vList;
```

A further consideration is whether the tested list's items are in varying case ('ant' vs. 'Ant' vs. 'ANT' etc.). For instance, to take a mixed case list with duplicates and end up with a de-duped all-lowercase version, use:

```
var:list vList;
$MyList.lowercase.each(anItem){
   if(!vList.contains(anItem)){
      vList+=anItem;
   }
};
$MyList = vList;
```

Resulting order

From v9.5.0, **.unique()** preserves the order of elements in a list. Previously, the operator forced an A-Z order, but this was not ideal as one feature of List-type lists is that they allow duplicates do not auto-sort—unlike Set-type lists). Thus now:

```
"C:A;A;B".unique;
```

now returns "C;A;B" (previously it would have been "A;B;C", destroying the original ordering).

list(expressionList)

Operator Type:	Function [other Function type actions]
Operator Scope of Action:	Item [operators of similar scope]
Operator Purpose:	Data manipulation [other Data manipulation operators]
Operator First Added:	Baseline
Operator Last Altered:	As at baseline

---

list(expressionList)

This function returns the evaluated format of each of its *comma-delimited* argument list of expressions.

Expression arguments can be:

- Literal values: `Fred` though there is little point in using these unless the list is subsequently to be concatenated to a String.
- Basic literal operations `"Fred"+" "+Smith"`.
- Arithmetic operations: `6/2`.
- Attributes (values) both system e.g. `$Name` and user attributes e.g. `$MyNumber`.
- Action operators `sqrt(2)`.
- Action code expressions, as in whole code expressions `"Date: "+date("today").format("*")`.

The function is also useful as a method of assembling lists of attributes or expressions for action functions using lists such as count(), max() and min():

FAILS: `$MyDate = max($MyDateA,$MyDateB,$MyDateC);`

The latter fails as max() interprets the list as 3 literal strings "$MyDateA", etc., and does a lexical sort on those values. However:

WORKS: `$MyDate = max(list($MyDateA,$MyDateB,$MyDateC));`

Functions *creating* lists (sum(), links(), collect(), etc.) do not have the same problems with the likes of max as the former output a valid list that can be used directly. For instance:

WORKS: `$MyDate = max(links.outbound.attended.$MyDate);`

For more complex examples, where list items are action code expressions, it may be necessary to use eval() to wrap each list item expression, e.g. `list(eval(expressionA),eval(expressionB))`.

Examples

This returns a list of numbers resulting from simple evaluations:

Code: `list(4+5,9-3,100/2.5)`

Output value, a list: `9;6;40`

This example, this makes a single evaluated string out of a list of two expressions:

Code: `list("This note 's $Width is "+$Width, "the $Xpos is "+$Xpos).format(" and ")+"!")`

Output value, a string: `This note's $Width is 3 and the $Xpos is 48.5!`

This example is not overly complex but points to how list() can be used in constructing output strings/lists that are otherwise difficult to create. The more complex the expressions passed to list() the greater the likelihood of getting no output, c an unexpected one. If complex inputs do not work consider using more parentheses to help TB figure the order of sub-task execution or else put the result of expressions into new attributes and then pass the value of the latter into list() as an argument.

List/Set.any(loopVar, expressionStr)

Operator Type: Function [other Function type actions]
Operator Scope of Action: List [operators of similar scope]
Operator Purpose: Query Boolean [other Query Boolean operators]
Operator First Added: Baseline
Operator Last Altered: As at baseline
Operator Uses Loop Variable Argument: [More on loop variable arguments]

List/Set.any(loopVar, expressionStr)

This is `true` if any **loopVar** in the list matches the expression.

```
$MyBoolean = $MyList.any(x,x>5)
```

is `true` if any **loopVar** in $MyList is greater than 5.

The comparison may also be applied to literal lists:

```
"apple; pear; plum".any(x, x=="plum")
```

is `true` because at least one element has the value "plum".

If the target list or set is empty, .any() always returns `false`, and .every() always returns `true`.

The **loopVar** is a user-set *case-sensitive* string. "x", "anItem", etc., are equally applicable. Similar to a loop variable in .each(x){}, the point of the **loopVar** value, is to set a reference variable for each list element. This can then be used in the code provided by the **expressionStr** argument. Using a number for **loopVar**, e.g. '1' or '42' is not recommended. Choose a value that makes sense for your own style of work

The **expressionStr** is any action code expression that is a test resolving to a Boolean true/false answer.

For example, to test if any item exactly matches the value stored in the $MyString of 'Some note':

```
$MyBoolean = $MyList.any(anItem, anItem == $MyString("Some Note"));
```

Or, any list value that starts with the string 'Large':

```
$MyBoolean = $MyList.any(Z, Z.contains("^ large"));
```

List/Set.asString()

Operator Type: Function [other Function type actions]
Operator Scope of Action: Item [operators of similar scope]
Operator Purpose: Data manipulation [other Data manipulation operators]
Operator First Added: Baseline
Operator Last Altered: As at baseline

Syntax note: *Operators without any defined mandatory arguments may omit their empty closing parentheses*

List/Set.asString()

List/Set.asString

The dot-operator **.asString()**, converts sets and lists to a string representation. This addresses an issue where much-used operators **.contains()** and **.icontains()** behave differently for Strings (where it searches for a regular expression match compared to Lists and Sets (where it tests for set membership). Occasionally, is desirable to perform a regular expression test on a list or set—for example, to ask if any of the members of $MyList begin with the letter "a":

```
$MyBoolean = $MyList.asString().contains(^ a");
```

Note there are no arguments for this operator so the trailing parentheses optionally may be omitted:

```
$MyBoolean = $MyList.asString.contains(^ a");
```

The role of **.asString()** is simple:

```
$MyList = [winken;blinken;nod];
$MyString = $MyList.asString();
```

The value of MyString is now "winken;blinken;nod", still the literal stored value of my list complete with semicolon delimiters, but—importantly—Tinderbox now treats this is a literal String, despite the semicolons, and not as List/Set type data. That allows the **.contains()** example above to do a regex (String behaviour) match on what is otherwise treated as a 3-item list.

List/Set.at(itemNum)

Operator Type: Function [other Function type actions]
Operator Scope of Action: Item [operators of similar scope]
Operator Purpose: Data manipulation [other Data manipulation operators]
Operator First Added: Baseline
Operator Last Altered: As at baseline

List.at(itemNum)

This returns the value of item **itemNum** of the **List** as a string. List can be either Set or List type attributes (or string literal, regex, or expression equivalents thereof). This operator is read-only: list values can be read, but not set.

Note: for accessing look-up tables, use list.lookup().

The operator is zero-based, i.e. an **itemNum** value of 0 returns the first list item, an **itemNum** of 1 returns value #2, etc. If the value of N exceeds the number of items in the list an empty string (blank value) is returned. A negative number returns an item numbering in reverse, but *one-based* not zero-based, so '-1' returns the last item on the list, '-2' the last but one item, etc.

Examples (where $MyList is "ant;bee;cow"):

`$MyString = $MyList.at(0);` returns "ant"
`$MyString = "XX;YY;ZZ".at(2);` returns "ZZ"
`$MyString = $MyList.at(5);` returns "" (nothing)
`$MyString = $MyList.at(-2);` returns "bee"

See also the more recent List/Set[N] usage.

To address particular locations in a list, also see List/Set.first(), List/Set.last() and List/Set.randomItem().

Legacy use (pre-v8)

List.at("key")

This usage is **deprecated**, use List/Set.lookup("key") instead. The remainder of this section is for explanation of legacy code use only.

The .at() function is also useful for accessing values from look-up tables by providing the relevant key. Consider a look-up list:

```
$RegionList="AL:South;AK:NorthWest;default:North";
```

This allows actions like:

```
$Region=$RegionList.at("AL");
```

or

```
$Region=$RegionList.at($State);
```

List/Set.avg()

Operator Type: Function [other Function type actions]
Operator Scope of Action: List [operators of similar scope]
Operator Purpose: Dictionary, Set & List operations [other Dictionary, Set & List operations operators]
Operator First Added: Baseline
Operator Last Altered: As at baseline

Syntax note: *Operators without any defined mandatory arguments may omit their empty closing parentheses*

List/Set.avg()

List/set.avg

Returns the mean value of a list or set of numbers. For example, if MyList is a list of numerical values '3;4;8;3;9;2;12':

`$MyNumber = $MyList.avg;` returns 5.85714

To get only two decimal places:

`$MyNumber = $MyList.avg.format("2");` returns 5.86

List/Set.collect_if(loopVar, condition, expressionStr)

Operator Type:	Function [other Function type actions]
Operator Scope of Action:	List [operators of similar scope]
Operator Purpose:	Dictionary, Set & List operations [other Dictionary, Set & List operations operators]
Operator First Added:	Baseline
Operator Last Altered:	As at baseline
Operator Has Conditional Arguments:	[More on conditional operator arguments]
Operator Uses Loop Variable Argument:	[More on loop variable arguments]

List/Set.collect_if(loopVar, condition, expressionStr)

The dot-operator **.collect_if()** collects the members of a list that satisfy a condition. Each item in the list is bound in turn to **loopVar**, and then the **expressionStr** is evaluated.

loopVar is essentially the same as the loop variable used by the List.each() operator. In the examples below, for clarity the **loopVar** value "anItem" is used, but as with any loop variable a shorter less expressive values such a "x" can be used (e.g. by more expert users).

The **condition** argument is a conditional expression for which each tested item must return `true` or `false`.

The operator applies the action code **expressionStr** to only those list items for which condition is `true`. For only list items meeting condition, the result of **expressionStr** on **loopVar** is returned as List-type data. Unlike List/set.collect(), the returned list may well contain fewer items than the source list, unless all source items match **condition**.

If $MyList is "1;2;3;4;5", anItem, is 1, then 2, etc. For example:

```
$MyListA = $MyList.collect_if(anItem, anItem <3, anItem);
```
returns 1;2 (only 2 of 5 source items match **condition**)

```
$MyListA = $MyList.collect_if(anItem, mod(anItem,2), anItem);
```
returns 1;3;5 (only 3 of 5 source items match **condition**)

```
$MyListA = $MyList.collect_if(anItem, mod(anItem,2), anItem* anItem);
```
returns 1;9;25 (only 3 of 5 source items match **condition**)

```
$MyListA = $MyList.collect_if(anItem, anItem>0, anItem* anItem);
```
returns 1;4;9;16;25 (all 5 source items match **condition**)

In the first three examples above note how only *some* of the original 5 source list items are returned as some input items fail the the **condition** test. In the last example, as all 5 items are greater than zero (the **condition**) to the **expressionS** applied to every one of them and all are returned.

If $MyList is "Winken;Blinken;Nod", then:

```
$MyListA = $MyList.collect_if(anItem, anItem.contains('i'),anItem.lowercase);
```
returns "winken;blinken" (only 2 items are returned)

In the last example note how only 2 of the original 3 source list items are returned as the item "Nod" does not contain the letter 'i' and so fails the **condition** test.

List/Set.collect_if() vs. collect_if()

Although the two appear similar. This operator works directly on the source list values, whereas collect_if() creates a list of $Path values and returns on an attribute value from each of those paths (where the item at the $Path meets the condition).

List/Set.collect(loopVar, expressionStr)

Operator Type:	Function [other Function type actions]
Operator Scope of Action:	List [operators of similar scope]
Operator Purpose:	Dictionary, Set & List operations [other Dictionary, Set & List operations operators]
Operator First Added:	Baseline
Operator Last Altered:	As at baseline
Operator Uses Loop Variable Argument:	[More on loop variable arguments]

List/Set.collect(loopVar, expression)

Each item in the list is bound in turn to **loopVar**, and then the **expressionStr** is evaluated).

loopVar is essentially the same as the loop variable used by the List.each() operator. In the examples below, for clarity the **loopVar** value "anItem" is used, but as with any loop variable a shorter less expressive values such a "x" can be used (e.g. by more expert users).

The operator applies the action code **expressionStr** to each list item in turn. The result of expression on **loopVar** is returned as List-type data. The operator always returns a list of *all* the results, i.e. the size of the input and output lists are th same. By comparison, this may not be the case with the similar List/Set.collect_if() operator.

If $MyList is "1;2;3;4;5", anItem, is 1, then 2, etc. For example:

```
$MyListA = $MyList.collect(anItem,anItem+1);
```
returns 2;3;4;5;6

```
$MyListA = $MyList.collect(anItem, anItem* anItem);
```
returns 1;4;9;16;25

If $MyList is "Winken;Blinken;Nod", then:

```
$MyListA = $MyList.collect(anItem, anItem.lowercase);
```
returns "winken;blinken;nod"

List/Set.collect() vs. collect()

Although the two appear similar. This operator works directly on the source list values, whereas collect() creates a list of $Path values and returns on an attribute value from each of those paths.

List/Set.contains(matchStr)

Operator Type:	Function [other Function type actions]
Operator Scope of Action:	Item [operators of similar scope]
Operator Purpose:	Query Boolean [other Query Boolean operators]
Operator First Added:	Baseline
Operator Last Altered:	As at baseline

List.contains(matchStr)

This operator tests whether the string **matchStr** matches a whole discrete value string within a the referenced list/set attribute value. Unlike when used with a String, e.g. String.contains, there is no regex functionality. With lists the function behaves as if the chained list were being iterated and an equality (==) test was being run on each list item.

Matches are always case-*sensitive*, unlike List/Set.icontains() where the matches are always *case-sensitive*.

Used with a list (List or Set data types), **.contains()** cannot match to granularity less than a single whole item in the chained list. **Importantly**, this differs from the operator's use with String data, e.g. String.contains(), where regular expressic matching is applied. If regex parsing is needed, e.g. to match a partial list value, coerce the list to a string and use to operator on that string—see below.

A match gives a numerical result which is the 1-based matched list position. That number coerces to the boolean result needed for use in queries:) to `false` and 1 or more to `true`.

matchStr is one of:

- an action code expression (which includes just referencing a single attribute name)
- a quoted literal string (i.e. actual text). *Important*: do not omit the enclosing quotes. If omitted, Tinderbox will try to evaluate the string as an expression. Doing this *may* result in the expected result but this is actually a false positive. *Se remember to enclose your literals in quotes.*

$MyList.contains(matchStr)

The contains operator may also be used with both sets and lists, in which case it tests for set membership, i.e. matching to *complete individual values* rather than part of values. Thus:

```
$MyList.contains("Tuesday")
$MyList(parent).contains("Tuesday")
```

are both `true` if $MyList contains "Monday;Tuesday;Friday". A match can use an attribute value as the **matchStr**. Consider a single-value String-type attribute 'MyDay':

```
$MyList.contains($MyDay)
```

is `true` if the value of $MyDay for a given note is any of "Monday", "Tuesday or "Friday". Thus in an agent or find query, the regex varies by the source value in the currently-tested note.

The chained list may also be a literal list:

```
"Saturday;Sunday".contains("Sunday")
"Saturday;Sunday".contains($MyDay)
```

If the **matchStr** is found the function returns a number which is the 1-based matched list position. In the last example above the returned value will be `2`, because 'Sunday' is the second item in the list.

Testing a negative: "does not contain"

Use a ! prefix to the query argument:

```
!$MyList.contains("Tuesday")
```

Use of parentheses around the negated query term, can assist Tinderbox's parsing:

```
(!$MyList.contains("Tuesday"))
```

Matching partial list values

As changing to list suppresses the normal string regex parsing, interposing the . asString() operator allows the list to be treats a a string so as to behave like a String.contains() test. See the .asString() operator listing for more detail. This is a more elegant replacement for the old workaround of using `List.format("#").icontains("some match)` as may be seen in some older code samples.

List/Set.count_if(loopVar, condition)

Operator Type:	Function [other Function type actions]
Operator Scope of Action:	Group [operators of similar scope]
Operator Purpose:	Data manipulation [other Data manipulation operators]
Operator First Added:	Baseline
Operator Last Altered:	As at baseline
Operator Has Conditional Arguments:	[More on conditional operator arguments]
Operator Uses Loop Variable Argument:	[More on loop variable arguments]

List/Set.count_if(loopVar, condition)

The dot-operator **.count_if()** counts the members of a list that satisfy a **condition**. Each item in the list is bound in turn to **loopVar**, and *if* **condition** is met, then that item is added to the returned count.

loopVar is essentially the same as the loop variable used by the List.each() operator. In the examples below, for clarity the **loopVar** value "anItem" is used, but as with any loop variable a shorter less expressive values such a "x" can be used (e.g. by more expert users).

The **condition** argument is a conditional expression for which each tested item must return `true` or `false`.

For example if $MyList is "1;2;3;4;5", then:

```
$MyListA =- $MyList.count_if(anItem, anItem>3)
```

returns `2`, being the number of items in $MyList whose value is greater than 3. If $MyList is "1;1;2;2;3;3;4;4;5;5", then:

```
$MyListA =- $MyList.count_if(anItem, anItem>3)
```

returns `4`, being the number of items in $MyList whose value is greater than 3 and values '4' and '5' each occur twice.

List/Set.count_if() vs. .count_if()

Although the two appear similar. This operator works directly on the source list values, whereas count_if() creates a list of $Path values and returns on an attribute value from each of those paths (where the item at the $Path meets the condition).

List/Set.count()

Operator Type:	Property [other Property type actions]
Operator Scope of Action:	List [operators of similar scope]
Operator Purpose:	Data manipulation [other Data manipulation operators]
Operator First Added:	Baseline
Operator Last Altered:	As at baseline

Syntax note: *Operators without any defined mandatory arguments may omit their empty closing parentheses*

List/Set.count()

List/set.count

The property .count counts the Number of discrete items in the specified List or Set data type attribute.

This is better used instead of count(list) or List/Set.size.

The subject list is evaluated so can use a literal list or $attribute(note). It can also use more complex expressions to get data as long as the result is an attribute of the List or Set data type.

For example if $DisplayedAttributes for the current note is "Color;AccentColor;NameFont" then the code

```
$MyNumber = $DisplayedAttributes.count;
```

is effectively

```
$MyNumber = ("Color;AccentColor;NameFont").count;
```

and not surprisingly returns 3. Note that the count is not all unique values for the attribute across the whole TBX; scope is restricted to 'this' note or another nominated note. Specimen usage:

```
$MyNumber = $DisplayedAttributes.count;
$MyNumber = $DisplayedAttributes("some other note").count;
```

To use `.count` with a list of items that are attributes or expressions, use `list()` to pre-create a list:

Works: `$MyNumber = list(4+2,9+6).count;` (output: 2)

For more complex examples, where list items are action code expressions, it may be necessary to use eval() to wrap each list item expression e.g. `list(eval(expressionA),eval(expressionB))`.

Examples

The following is a trivial example (given you could use $ChildCount instead) but shows how count can be used in a more subtle way:

```
$MyNumber = collect(children,$Name).count;
```

The result of collect() is a List, in this case a number of note titles. `List.count` will return the number of values in the list (including duplicates). To get a de-duped count, chain the `.unique` function before the `.count`:

```
$MyNumber = collect(children,$Name).unique.count;
```

Similarly, find() returns a List but note that find() does *not* de-dupe for aliases. Thus test $IsAlias in the query to weed alias results from the returned list:

```
$MyNumber = find($Prototype=="pProject"&$IsAlias==false&$ChildCount>1).count;
```

List/Set.countOccurrencesOf(literalStr)

Operator Type:	Function [other Function type actions]
Operator Scope of Action:	List [operators of similar scope]
Operator Purpose:	Data manipulation [other Data manipulation operators]
Operator First Added:	Baseline
Operator Last Altered:	As at baseline

List/Set.countOccurrencesOf(literalStr)

This function returns the Number of times that the *literal* string **literalStr** appears in the source **List** or **Set**. For lists, such as List and set types, string of the attribute's raw concatenated list (i.e. with semicolon delimiters) is tested.

If $MyList contains "ant;bee;ant;cow;ant", then:

```
$MyNumber = $MyList.countOccurrencesOf("ant");
```
returns 3

Sets de-duplicate, but partial matches of Sets do not. If $MySet contains "cat;cut;hat;hit;hut;pat;sat", then

```
$MyNumber = $MySet.countOccurrencesOf("at");
```
returns 4

literalStr is literal and must *not* be a regular expression. If the latter is needed use List/Set.contains() or List/Set.icontains().

List/Set.each(loopVar){actions}

Operator Type:	Function [other Function type actions]
Operator Scope of Action:	Item [operators of similar scope]
Operator Purpose:	Data manipulation [other Data manipulation operators]
Operator First Added:	Baseline
Operator Last Altered:	As at baseline
Operator Uses Loop Variable Argument:	[More on loop variable arguments]

List/Set.each(loopVar){ action(s) }

This allows you to operate in turn on each item in a list or set. Put another way you 'iterate;' or loop though, every item in the list evaluating any code inside the `{ }` brackets. The user-defined (i..e. *named by the user*) loopVar can be used inside the loop to refer to the value of the current list item value being iterated. If the list is a list of path info (or $IDString, $ID, or unique $Name) then in-loop loopVar can be used for offset addressing the note at that path via an attribute offset argument, i.e. `$SomeAttribute(loopVar)`.

What is a loop variable?

For example, for a note with a list of values in $MyList:

```
$MyList.each(x){$Result=$Result +x+"\n" ;}
```

will replace the current note's value of $Result with a list of every value from $MyList, but each on its own line. The **loopVar**, here the in-loop variable 'x', is simply the string defined in .each(), and is *case-sensitive* and a '$' prefix is **not** required for in-loop references to the loop variable. For most Tinderbox users, a more readable/understandable version of the above is this:

```
$MyList.each(aString){
```

```
 $Result=$Result +aString+"\n" ;
}
```

What has changed? Firstly, the loop has been broken out onto speaker lines and in-loop code indented. Tinderbox ignores line breaks and white space between operators and values so both the above seems the same to the parser. Secondly the obfuscatory 'x' **loopVar** name has been changed to a more descriptive 'aString' based on the presumption that $Result is a String (it cannot be a Number because Number-type data cannot have line breaks within a single value) and, for the calling note, we are compiling a string the list $Result values.

Thus **.each(loopVar)** would iterate using references to **loopVar**, and thus `.each(Y)` would iterate using references to `'Y'`, whilst `.each(Abracadabra)` would iterate using references to `'Abracadabra'`, and so forth. Consider making the loop variable something clear to both the user and to Tinderbox. In the trivial example above, 'x' seems pretty clear but might be misread by the users—in a mathematical context—as a multiplication symbol; Tinderbox will not be confused as it uses * for multiplication but consider how something like 'LoopVar' might be clearer. By the same token make sure the loop variable name does not clash with existing attribute names or attribute string values of the same name that might be used as part of the data being processed.

A variable declared using var() may be altered from within the scope of an .each() loop.

If $Total is a numeric attribute and $MyList is a list of numbers:

```
$TotalNumber=0;
$MyList.each(aNumber){
 $TotalNumber += aNumber*aNumber; // parentheses not required but can be useful if calculation is not simple
}
```

computes the sum of the squares of the values in $MyList and stored it in $TotalNumber. As the list is a list of numbers, the *user-chosen* **loopVar** name used is 'aNumber'.

Note: remember that **you**, the user, choose the actual name of **loopVar** in your own code. It can vary per use, as in the examples on this page.

The **loopVar** can be a path and this can be used as a variable designator for attribute offset addressing inside the loop:

```
$Text="";
collect(children,$Path).each(aPath){
 $Text += "\n"+$Text(aPath);
}
```

In the above, a list is created on the fly of the current note's children. Then the current note's $Text is appended, for each child, with a line and the child's $Text. As the list is a list of paths, the *user-chosen* **loopVar** name used is 'aPath'. Thu for each iteration of the loop aPath's value is being used to provide the offset address in the loop to **$Text(loopVar)**.

If it is desired to iterate a list a particular number of times, another approach to the above is to use the range '...' operator which can be used to provide a numbered loop variable (see).

After each iteration of an each(){} loop, accumulated back-references are cleared. Formerly, back-references from each iteration were retained for the lifetime of the loop, making it very difficult to retrieve the desired reference.

To test for position in a loop, i.e. detecting if processing the first or last items, see List/Set.first() and List/Set.last().

To get a random item from a list without needing to use a loop, see List/Set.randomItem().

A worked example of looping is here.

Examples of using action() within a loop to create attribute references is given here.

List/Set.empty()

Operator Type: Property [other Property type actions]
Operator Scope of Action: Item [operators of similar scope]
Operator Purpose: Non-query Boolean [other Non-query Boolean operators]
Operator First Added: Baseline
Operator Last Altered: As at baseline

Syntax note: *Operators without any defined mandatory arguments may omit their empty closing parentheses*

List/Set.empty()

List/Set.empty

This returns a Boolean depending on whether the list is empty. In this context Set and List type can be regarded as interchangeable. If empty, the return value is `true`, if the attribute has content then `false` is returned.

```
$MyList = [hello;world]; $MyBoolean = $MyList.empty;
```

$MyBoolean is set to `false` as content is found. But

```
$MyList = []; $MyBoolean = $MyList.empty;
```

$MyBoolean is set to `true` as no content is found.

List/Set.every(loopVar, expressionStr)

Operator Type: Function [other Function type actions]
Operator Scope of Action: List [operators of similar scope]
Operator Purpose: Query Boolean [other Query Boolean operators]
Operator First Added: Baseline
Operator Last Altered: As at baseline
Operator Uses Loop Variable Argument: [More on loop variable arguments]

List/Set.every(loopVar, expressionStr)

This is `true` if every **loopVar** in the list meet

```
$MyList.every(x,x>5)
```

is `true` if every **loopVar** in $MyList is greater than 5.

The comparison may also be applied to literal lists:

```
"apple; pear; plum".every(x, x>"aardvark")
```

is `true` because every element follows "aardvark" in alphabetical (lexical sort) order.

If the target list or set is empty, .any() always returns `false`, and .every() always returns `true`.

The **loopVar** is a user-set *case-sensitive* string. "x", "anItem", etc., are equally applicable. Similar to a loop variable in .each(x){}, the point of the **loopVar** value, is to set a reference variable for each list element. This can then be used in the code provided by the **expressionStr** argument. Using a number for **loopVar**, e.g. '1' or '42' is not recommended. Choose a value that makes sense for your own style of work

The **expressionStr** is any action code expression that is a test resolving to a Boolean true/false answer.

For example, to test if every item exactly matches the value stored in the $MyString of 'Some note':

```
$MyBoolean = $MyList.every(anItem, anItem == $MyString("Some Note"));
```

Or, every list value starts with the string 'Large':

```
$MyBoolean = $MyList.every(Z, Z.contains("^ large"));
```

List/Set.first()

Operator Type: Property [other Property type actions]
Operator Scope of Action: List [operators of similar scope]
Operator Purpose: Dictionary, Set & List operations [other Dictionary, Set & List operations operators]
Operator First Added: Baseline
Operator Last Altered: As at baseline

Syntax note: *Operators without any defined mandatory arguments may omit their empty closing parentheses*

List/Set.first()

List/Set.first

Returns the first item of a list. If MyList is a 5-item list "ant;bee;cow;dog;eel":

```
$MyString = $MyList.first;
```
returns "ant"

List/Set.first(N)

Returns a list of the first **N** items of a list. With the same list as above:

```
$MyList2 = $MyList.first(2);
```
returns "ant;bee"

Testing for loop position

This operator can be used to test the current loop state, i.e. whether the currently processed item is the first in the list. Here the code in the commented section is run only when the first list item in $MyList is being processed:

```
$MyList.each(anItem){
 if(anItem==$MyList.first){
  // some code here ...
 };
};
```

Note that '$MyList.first' is not a test in itself. Rather, it supplies the value of the first list item which can be tested against the currently processed item.

See also List/Set.last.

List/Set.format(formatStr)

Operator Type: Function [other Function type actions]
Operator Scope of Action: Item [operators of similar scope]
Operator Purpose: Formatting [other Formatting operators]
Operator First Added: Baseline
Operator Last Altered: As at baseline

List/Set.format(formatStr)

If data is a List or Set, returns the list into a String, with discrete list elements formatted as per the **formatStr**:

```
$MyString = $MyList.format("formatString");
```

The process preserves original data; duplicate values in lists are maintained. For example

```
$MyString = $DisplayedAttributes.format(", ");
```

converts Displayed Attributes to a comma+space-separated list. To put each item on a separate line use this:

```
$MyString = $DisplayedAttributes.format("\n");
```

Doing the same for HTML export, you might want the rendered text to have each value on a new line so use:

```
$MyString = $DisplayedAttributes.format("<br>");
```

Thus $Text may be created from concatenation of other texts:

```
$Text = (collect(children, $Text)).format("\n");
```

Optionally, you may supply four arguments to format the list or set as an HTML list:

```
$MyString = List/Set.format("listPrefix","itemPrefix","itemSuffix","listSuffix");
```

For example:

```
$MyString = $Classes.format("<ul>","<li>","</li>","</ul>");
```

will return HTML code for a bulleted list with each set member marked up as a list item. Note that the tags must be in double quotes. To have each element on a separate line and indent the items add tabs and line breaks:

```
$MyString = $Classes.format("<ul>\n","\t<li>","</li>\n","</ul>\n");
```

To make this easier to use in a code export context, you might pass the output of format into another attribute and call the latter within the template with a ^value()^ code. By a repeated use of format it is possible to export lists of links.

Trailing semi-colons

Although Tinderbox happily accepts user-input lists with a trailing semi-colon after the last item, i.e. "cow;ant;" vs. "cow;ant"", trailing semi-colons can be problematic when formatting lists. This is because that process emits an extra empty item and preceded by the user specified delimiter. Thus the output my have an unwanted trailing delimiter. If encountered, the best approach to dealing with this is to clear the source data.

Similar functions

This supplements the existing format() function.

List/Set.icontains(matchStr)

Operator Type: Function [other Function type actions]
Operator Scope of Action: Item [operators of similar scope]
Operator Purpose: Query Boolean [other Query Boolean operators]
Operator First Added: Baseline
Operator Last Altered: As at baseline
Operator Uses Regular Expressions: [More on regular expressions in Tinderbox]

List.icontains(matchStr)

This operator tests whether the string **matchStr** matches a whole discrete value string within a the referenced list/set attribute value. Unlike when used with a String, e.g. String.contains, there is no regex functionality. With lists the function behaves as if the chained list were being iterated and an equality (==) test was being run on each list item.

Matches are always case-*insensitive*, unlike List/Set.contains() where the matches are always *case-sensitive*.

Used with a list (List or Set data types), **.icontains()** cannot match to granularity less than a single whole item in the chained list. **Importantly**, this differs from the operator's use with String data, e.g. String.icontains(), where regular expressi matching is applied. If regex parsing is needed, e.g. to match a partial list value, coerce the list to a string and use to operator on that string—see below.

A match gives a numerical result which is the 1-based matched list position. That number coerces to the boolean result needed for use in queries:) to `false` and 1 or more to `true`.

matchStr is one of:

- an action code expression (which includes just referencing a single attribute name')
- a quoted literal string (i.e. actual text). *Important*: do not omit the enclosing quotes. If omitted, Tinderbox will try to evaluate the string as an expression. Doing this *may* result in the expected result but this is actually a false positive. *Sc remember to enclose your literals in quotes.*

N.B. Unlike with String-type attributes, regex cannot be used.

$MyList.icontains(matchStr)

The contains operator may also be used with both sets and lists, in which case it tests for set membership, i.e. matching to *complete individual values* rather than part of values. Thus:

```
$MyList.icontains("Tuesday")
$MyList(parent).icontains("Tuesday")
```

are both `true` if $MyList contains "Monday;Tuesday;Friday". A match can use an attribute value as the **regexStr**. Consider a single-value String-type attribute 'MyDay':

```
$MyList.icontains($MyDay)
```

is `true` if the value of $MyDay for a given note is any of "Monday", "Tuesday or "Friday". Thus in an agent or find query, the regex varies by the source value in the currently-tested note.

The chained list may also be a literal list:

```
"Saturday;Sunday".icontains("Sunday")
"Saturday;Sunday".icontains($MyDay)
```

If the **matchStr** is found the function returns a number which is the 1-based matched list position. In the last example above the returned value will be `2`, because 'Sunday' is the second item in the list.

Testing a negative: "does not contain"

Use a ! prefix to the query argument:

```
!$MyList.icontains("Tuesday")
```

Use of parentheses around the negated query term, can assist Tinderbox's parsing:

```
(!$MyList.icontains("Tuesday"))
```

Matching partial list values

As changing to list suppresses the normal string regex parsing, interposing the . asString() operator allows the list to be treats a a string so as to behave like a String.icontains() test. See the .asString() operator listing for more detail. This is a more elegant replacement for the old workaround of using `List.format("#").icontains("some match)` as may be seen in some older code samples.

List/Set.intersect(aSet)

Operator Type: Function [other Function type actions]
Operator Scope of Action: List [operators of similar scope]
Operator Purpose: Data manipulation [other Data manipulation operators]
Operator First Added: Baseline
Operator Last Altered: As at baseline

Set.intersect(aSet)

This calculates the intersection of two List or Sets, the chained Set and the set in the **aSet** argument The .intersect() test is generally intended for use with Set-type data but will work with Lists, though the result is always a Set.

```
$MySet = $MySetA.intersect($MySetB);
$MySet = $MyList1.intersect($MyList2);
```

The result is a Set of all items in both $MySetA and $MySetB, or in the second example in both $MyList1 and $MyList2. As the result is *always* a set, any source list items are de-duped in the output.

Non-intersect

No special code is needed to find items in one set but not the other:

`$MySetC = $MySetA - $MySetB;` gives items only in $MySetA

`$MySetC = $MySetB - $MySetA;` gives items only in $MySetB

Use with Lists

Subtracting a Set from a List results in only one instance of each Set item being removed. Subtracting a List from a List each instance of a value in the second list is removed so multiple source List entries may be removed.

List/Set.last()

Operator Type:	Property [other Property type actions]
Operator Scope of Action:	List [operators of similar scope]
Operator Purpose:	Dictionary, Set & List operations [other Dictionary, Set & List operations operators]
Operator First Added:	Baseline
Operator Last Altered:	As at baseline

Syntax note: *Operators without any defined mandatory arguments may omit their empty closing parentheses*

List/Set.last()

List/Set.last

Returns the last item of a list. If MyList is a 5-item list "ant;bee;cow;dog;eel":

`$MyString = $MyList.last;` returns "eel"

List/Set.last(N)

Returns a list of the last **N** items of a list. With the same list as above:

`$MyList2 = $MyList.last(2);` returns "dog;eel"

Testing for loop position

This operator can be used to test the current loop state, i.e. whether the currently processed item is the last in the list. Here the code in the commented section is run only when the last list item in $MyList is being processed:

```
$MyList.each(anItem){
 if(anItem==$MyList.last){
  // some code here ...
 };
};
```

Note that '$MyList.last' is not a test in itself. Rather, it supplies the value of the last list item which can be tested against the currently processed item.

See also List/Set.first.

List/Set.lookup(keyStr)

Operator Type:	Function [other Function type actions]
Operator Scope of Action:	List [operators of similar scope]
Operator Purpose:	Dictionary, Set & List operations [other Dictionary, Set & List operations operators]
Operator First Added:	Baseline
Operator Last Altered:	As at baseline

List.lookup(keyStr)

The command **.lookup()** It is intended for look-up tables (i.e. single dimension arrays). For the key **keyStr** value is supplied, the matched key's value is return. We can set up an example listing, using a List-type attribute:

`$MyList = [ant:Wood ant;bee:Carder bee;cow:Jersey;dog:Labrador];`

This creates a 4 item look-up list. The first list item has two parts - the *key* 'ant' and its paired *value* 'Wood ant'. Passing a key via **.lookup**, returns its key:

`$MyString = $MyList.lookup("cow");` → "Jersey", as list item 3's key is matched.

If a key value with no match is passed, the result is an empty string

`$MyString = $MyList.lookup("pig");` → ""

But if we add a 'default' key/value pair (anywhere in the list):

`$MyList = [ant:Wood ant;bee:Carder bee;cow:Jersey;dog:Labrador;default:animal];`

and re-run the last example:

`$MyString = $MyList.lookup("pig");` → "animal"

There is still no match but as a default is defined, the default value of "animal" is returned.

More complex and nuanced use of **.lookup()** is described in the discussion of look-up tables.

Dictionary vs. Lookup

The newer Dictionary data-type offers a more efficient and feature rich way of working with lookup lists.

Legacy use (pre v8)

For look-up tables **.lookup()** is preferred to the older .at() for clarity, and to avoid ambiguity when the argument is numeric. Using the example list as above:

`$MyString = $MyList.at(3);` → "dog:Labrador", the whole fourth element of the list (do not forget N is counted from zero).

`$MyString = $MyList.lookup(3)` → "animal", the lookup result for key value `5` which doesn't exist, so we get the default.

List/Set.max()

Operator Type:	Property [other Property type actions]
Operator Scope of Action:	List [operators of similar scope]
Operator Purpose:	Data manipulation [other Data manipulation operators]
Operator First Added:	Baseline
Operator Last Altered:	As at baseline

Syntax note: *Operators without any defined mandatory arguments may omit their empty closing parentheses*

List/Set.max()

List/Set.max

The List/Set.max() operator returns the largest item in a **List** or **Set** data type attribute.

This is a replacement for/alternative to the max() operator.

Both the operators List/Set.max and List.Set.min use lexical comparison in most cases, but numeric comparison if the context is numeric (i.e. the reference is a Number-type attribute) and/or all list items are numbers. Thus:

`$Width=("1;100;2").max;`

Since "Width" is numeric, max() will be return 100.

`$Name=("1;100;2").max;`

Since the attribute "Name" is a string, max() will return 2.

If you do not have a set, create one on the fly:

`$MyMax = collect(all,$MyNumber).max;`

`$MyMax = collect(descendants,$Modified).max;`

This allows export via ^value^:

^value(collect(descendants,$Modified).max)^

When using Date-type data bear in mind that the unset Date default is "never" and that "never' is always before (i.e. less than) any set date. So to use .max with dates, filter out the unset values:

`$MyMin = collect_if(descendants,($Modified!="never"),$Modified).min;`

To use max() with a list of items that are attributes or expressions, use list():

Works: `$MyNumber = list(4+2,9+6).max;` (output: 15)

For more complex examples, where list items are action code expressions, it may be necessary to use eval() to wrap each list item expression e.g. `list(eval(expressionA),eval(expressionB))`.

List/Set.min()

Operator Type: Property [other Property type actions]
Operator Scope of Action: List [operators of similar scope]
Operator Purpose: Data manipulation [other Data manipulation operators]
Operator First Added: Baseline
Operator Last Altered: As at baseline

Syntax note: *Operators without any defined mandatory arguments may omit their empty closing parentheses*

List/Set.min()

List/Set.min

The List/Set.min operator returns the smallest item in a **List** or **Set** data type attribute.

This is a replacement for/alternative to the min() operator.

Both the operators List/Set.min and List/Set.max use lexical comparison in most cases, but numeric comparison if the context is numeric (i.e. the reference is a Number-type attribute) and/or all list items are numbers. Thus:

```
$Width=("100;2;70").min;
```

Since "Width" is numeric, min() will be return 2.

```
$Name=("100;2;70").min;
```

Since the attribute "Name" is a string, .min will return 100.

If you do not have a set, create one on the fly:

```
$MyMin = collect_if(all,$MyNum>0,$MyNum).min;
$MyMin = collect(descendants,$Modified).min;
```

This allows export via ^value^:

^value(collect(descendants,$Modified).min)^

When using Date-type data bear in mind that the unset Date default is "never" and that "never' is always before (i.e. less than) any set date. So to use .min with dates, filter out the unset values:

```
$MyMin = collect_if(descendants,($Modified!="never"),$Modified).min;
```

To use count() with a list of items that are attributes or expressions, use list():

Works: `$MyNumber = list(4+2,9+6).min;` (output: 6)

For more complex examples, where list items are action code expressions, it may be necessary to use eval() to wrap each list item expression e.g. `list(eval(expressionA),eval(expressionB))`.

List/Set.randomItem()

Operator Type: Function [other Function type actions]
Operator Scope of Action: List [operators of similar scope]
Operator Purpose: Dictionary, Set & List operations [other Dictionary, Set & List operations operators]
Operator First Added: 9.5.2
Operator Last Altered: As at baseline

Syntax note: *Operators without any defined mandatory arguments may omit their empty closing parentheses*

List/Set.randomItem()

From v9.5.2, the **.randomItem()** returns a randomly-selected item from a list (List or Set data types):

```
$MyString = $MyList.randomItem();
```

This replaces the extra coding needed for such a task if using rand().

This operator can be used not only on attributes but also literal lists using list():

```
$MyString = list("ant;bee;cow;dog").randomItem();
```

list-based variables:

```
var:list vList = "ant;bee;cow;dog"; $MyString = vList.randomItem();
```

and list-creating operators:

```
$MyString = collect(find($SomeAttribute=="xyz"),$Path).randomItem();
```

List/Set.remove(matchValue)

Operator Type: Function [other Function type actions]
Operator Scope of Action: List [operators of similar scope]
Operator Purpose: Data manipulation [other Data manipulation operators]
Operator First Added: Baseline
Operator Last Altered: As at baseline

List.remove(matchValue)

This removes items from a list or set. The argument **matchValue** may be of the desired data type for matching. For example,

```
$MyList = $MyList.remove(0);
```

returns a new list from which all elements equal to zero have been removed, and

```
$MyList = $MyList.remove("cat");
```

returns a new list from which call elements equal to "cat" have been removed.

```
$MyList = $MyList.remove("cat; dog; badger");
```

removes "cats", "dogs", and "badgers".

List/Set.replace(regexMatchStr, replacementStr)

Operator Type: Function [other Function type actions]
Operator Scope of Action: Item [operators of similar scope]
Operator Purpose: Data manipulation [other Data manipulation operators]
Operator First Added: Baseline
Operator Last Altered: As at baseline
Operator Uses Regular Expressions: [More on regular expressions in Tinderbox]

List.replace(regexMatchStr, replacementStr)

This operator allows simple text transformations without use of runCommand as was hitherto required; the result remains a List or Set as per the list of references supplied. Unlike in contains() type operators, ***some*** *regex are not supported* f either argument; regex use is discussed in more detail below.

regexMatchStr and **replacement** are one of:

- an action code expression (which includes just referencing a single attribute name')
- a quoted string, which may be either:
  - a literal string (i.e. actual text)
  - a regular expression (**regexMatchStr** only)

$MyList.replace(regexMatchStr, replacementStr)

List and set type attributes can use replace, though the scope of replacement is more limited than with strings. With listings, the match with and replacement of can only be for a complete list value and not part of a value.

A Replace action does not alter the original source

Using .replace() does not affect the source string unless the replacement output is used to overwrite the original source value. Thus if $MyString holds "Hello World" then:

```
$MyStringA = $MyString.replace(" World");
```

$MyString remains "Hello Word" and $MyStringA has value "Hello". The source is unchanged. But, if we set the source to the output

```
$MyString = $MyString.replace(" World");
```

Now $MyString becomes "Hello" and the original value is lost (overwritten by the new one.

Using regex (regular expressions)

Most basic regex expressions should work but string start (^) and string end ($) matches work in an unexpected way. When .replace() is run it looks at the internal string value of the Set or List.

Thus a list of values, like ant/bee/cow/dog/eel, is stored and matched as a single semi-colon delimited string " `ant;bee;cow;dog;eel`". Note Tinderbox does not create a final semi-colon after the last value, but will not complain if the user adds one, e.g. via manual input. Thus a ^ regex matches only before the 'a' of 'ant' and not the start of other list values. Similarly, $ matches after the 'g' of dog and not the end of other list values. It might be thought of as '^ant;bee;cow;dog;eel$' as opposed to '^ant$;^bee$;^cow$;^dog$;^eel$'.

So, in-list value boundaries still exist for regex matching but only as literal semicolons. Thus to change 'ee' to 'eet' in the above list but only for 'bee' and not 'eel':

```
$MyListA = $MyList.replace("ee;","eet;").replace("ee$","eet");
```

Note how *two* chained .replace() calls are needed, not one. The first is for inter-value boundaries and the second for the overall string end (had the data had a closing semi-colon the first match catches it so that scenario's still covered. To reverse the scenario and match the 'ee' at the start of a value:

```
$MyListA = $MyList.replace(";ee","ree;").replace("^ee","ree");
```

That changes 'eel' to 'reel' but leaves 'bee' unaltered.

It is possible to write back to the same attribute:

```
$MyList = $MyList.replace("ee;","eet;").replace("ee$","eet");
```

but, the former is a good idea whilst developing/testing code for this technique, only switching out latter once sure of the result.

Regex and Back-references

Regex can be used to set back-references in the **regexMatchStr** input string, as in an agent query, that can then be used in the **replacementStr** string. This is described in more detail here.

Trimming leading/trailing whitespace

```
$MyList = $MyList.replace("+","").replace(" +$","").replace(" *; *",";");
```

The ' +' means *one or more space characters* . The first replace finds such a sub-string immediately following the start of the whole string ^), whilst the second does the same for a sub-string immediately before the end of the string ($). The third replace finds *zero or more space characters* either side of a semi-colon (the per-item list delimiter). The latter also matches a normal ';' delimiter but the test save writing separate regexs for space before and after the delimiter (e.g. " +;" and "; +") so the zero-or-more test (*) is used here instead of the one-or-more (+) used for the start/end of the overall string. Thus, using the code above, a lists like these with items having undesired leading/trailing space:

```
" ant ; bee ; cow ; dog "
" ant ;bee ;cow ; dog"
```

…both become…

```
"ant;bee;cow;dog"
```

Dealing with inline quote characters

Because **regexMatchStr** is parsed for regular expressions, it may be possible to use the '\dnn' form described here to work around the lack of escaping from single double quotes within strings.

List/Set.reverse()

Operator Type:	Function [other Function type actions]
Operator Scope of Action:	Item [operators of similar scope]
Operator Purpose:	Data manipulation [other Data manipulation operators]
Operator First Added:	Baseline
Operator Last Altered:	As at baseline

Syntax note: *Operators without any defined mandatory arguments may omit their empty closing parentheses*

List/Set.reverse()

List/Set.reverse

This function reverses the order of the referenced list, or set. The function does not reverse the individual list values, but rather reverses the exact order of the individual values. Trailing parentheses are optional for this function.

```
$MyList = $MyList.reverse();
$MyList = [ant;bee;cow].reverse();
```

The function may be chained with the List.sort() and List.isort() functions, noting that sorting can only be used with List-type and not Set-type data:

```
$MyList = $MyList.sort().reverse();
$MyList = $MyList.isort().reverse();
```

List/Set.size()

Operator Type:	Property [other Property type actions]
Operator Scope of Action:	Item [operators of similar scope]
Operator Purpose:	Data manipulation [other Data manipulation operators]
Operator First Added:	Baseline
Operator Last Altered:	As at baseline

Syntax note: *Operators without any defined mandatory arguments may omit their empty closing parentheses*

List/Set.size()

List/Set.size

This returns the Number of discrete values in a List/Set. The returned number can be coerced to a string. Trailing parentheses are optional for this property. Examples:

```
$MyList = [hello;world]; $MyStringA = $MyList.size;
```

$MyStringA is set to "2".

This property is the same evaluation as List/Set.count or count(list).

This operator can also be used on other attribute data types that are string-like, URL, File, etc.

List/Set.sum_if(loopVar, condition[, expressionStr])

Operator Type:	Function [other Function type actions]
Operator Scope of Action:	Group [operators of similar scope]
Operator Purpose:	Mathematical [other Mathematical operators]
Operator First Added:	Baseline
Operator Last Altered:	As at baseline
Operator Has Optional Arguments:	[More on optional operator arguments]
Operator Has Conditional Arguments:	[More on conditional operator arguments]
Operator Uses Loop Variable Argument:	[More on loop variable arguments]

List/Set.sum_if(loopVar, condition[, expressionStr])

The dot-operator **.sum_if()** sums the values of members of a list that satisfy a **condition**. Each item in the list is bound in turn to **loopVar**, and *if* **condition** is met, the items value is added to the retuned sum. Optionally, the **expressionStr** is evaluated, allowing a transform to be carried out on a matched list member's value before it is added to the overall returned sum.

loopVar is essentially the same as the loop variable used by the List.each() operator. In the examples below, for clarity the **loopVar** value "anItem" is used, but as with any loop variable a shorter less expressive values such a "x" can be used (e.g. by more expert users).

The **condition** argument is a conditional expression for which each tested item must return `true` or `false`.

The operator applies the action code **expressionStr** to only those list items for which condition is `true`. For only list items meeting condition, the result of **expressionStr** on **loopVar** is returned as List-type data.

For example, if $MyList is "1;2;3;4;5", then

```
$MyListA = $MyList.sum_if(aValue, aValue>3)
```

returns the sum of all the members of $MyList that are greater than 3.

An optional third argument **expressionStr** allows a matched values to be transformed before they are added to the sum. So:

```
$MyListA = $MyList.sum_if(aValue, aValue>3, aValue*aValue)
```

returns the sum of the squares of each member that is greater than 3.

List/Set.sum_if() vs. sum_if()

Although the two appear similar. This operator works directly on the source list values, whereas sum_if() creates a list of $Path values and returns on an attribute value from each of those paths (where the item at the $Path meets the

condition).

List/Set.sum()

Operator Type: Function [other Function type actions]
Operator Scope of Action: Item [operators of similar scope]
Operator Purpose: Mathematical [other Mathematical operators]
Operator First Added: Baseline
Operator Last Altered: As at baseline

Syntax note: *Operators without any defined mandatory arguments may omit their empty closing parentheses*

List/Set.sum()

List/Set.sum

For Lists or sets of Number-type data, this adds up lists of numbers. If $MyList is 1;2;3;4, then for:

```
$MyNumber = $MyList.sum;
```

$MyNumber is 10, i.e. 1+2+3+4.

Non-numerical items

The general expectation is this operator is used for number-only lists. Non-numerical list items are ignored:

`"2; 4; 6; 12".sum` gives `24`

`"2; 8bees; 4; bee; 6; bee5; 12; bee2bee"` also gives `24`

However, be aware of note this edge case;

`2; 8 bees; 4; bee; 6; bee 5; 12; bee 2 bee` gives `32`

Here, the original result of 24 is supplemented by the opening 8 from '8 bees' and the trailing 5 from 'bee5' to give 32. *But*, the 2 entirely within 'bee 2 bee' is not counted.

List/Set.tr(inStr, outStr)

Operator Type: Function [other Function type actions]
Operator Scope of Action: Item [operators of similar scope]
Operator Purpose: Data manipulation [other Data manipulation operators]
Operator First Added: Baseline
Operator Last Altered: As at baseline

List.tr(inStr, outStr)

This operator allows simple single character string manipulation. It computes a new list, copying each character of the source list but converting any characters in **inStr** to the corresponding characters in **outStr**. For example:

```
$MyList = $MyList.tr("a","A");
```

returns a copy of $MyList in which every "a" is converted to "A".

Backslash characters must be quoted and escaped:

```
$MyList = $MyList("c","\\r");
```

converts every "c" to a Macintosh newline characters (\r) . Note the need in this context for an extra backslash escape (so Tinderbox knows the intended swap value is "\r" and not "r").

For further information, see the macOS X man page for the UNIX `tr` command.

List/Set[itemNum]

Operator Type: Function [other Function type actions]
Operator Scope of Action: List [operators of similar scope]
Operator Purpose: Dictionary, Set & List operations [other Dictionary, Set & List operations operators]
Operator First Added: Baseline
Operator Last Altered: As at baseline

List/Set[itemNum]

sourceNum is the zero-based item number in the referenced List or Set.

Elements in lists, sets, may be extracted (referenced) with the bracket operator:

```
$MyList[1]
```

This has the same effect as List/Set.at( ), but may be more convenient. Both

```
$MyList[1]
$MyList.at(1)
```

Returns the second list item (as list item addresses are zero-based).

This syntax can also assign values to specific elements of lists. For example,

```
$MyList=[apple; pear; plum; cherry];
$MyList[1]="persimmon";
```

Will replace "pear" with "persimmon".

Nested lists

```
MyList = [cow;bee;[fish;whale];ant]
$MyList[2][1] gives "whale"
```

locale([localeCodeStr])

Operator Type: Function [other Function type actions]
Operator Scope of Action: Document [operators of similar scope]
Operator Purpose: Date-time [other Date-time operators]
Operator First Added: Baseline
Operator Last Altered: As at baseline

Syntax note: *Operators without any defined mandatory arguments may omit their empty closing parentheses*

locale(localeCodeStr)

locale(), allows changing of the locale used to translate dates. The same date can be written in a different format in different parts of the world. For example, in the USA a date is written 12/1/2006 to denote December 1, whereas in UK the same date is written 1/12/2006 and so on for variations around the world.

To read the current host OS' locale, call the operator with no argument:

```
$MyString = locale();
```

To set a locale, e.g. here for 'British English', pass the desired locale's code value as **localeCodeStr**;

```
locale("en_GB")
```

Locale codes begin with a two-letter language code, followed by and underscore and a two-letter region code. These are ISO standards ISO-639 and ISO-3166 respectively. Code combinations are available for any language supported by macOS.

To read, or return to, the user's preferred locale, i.e. is derived from their Mac's OS account, use

```
$MyString = locale();
```

You may also save the old locale in an attribute for subsequent use. For example:

```
$OldLocale=locale("en_GB");
```

…do various things, and then reset the original locale…

```
locale($OldLocale);
```

… if simple call locale() with no argument if the host OS locale had remained unchanged.

Note that changing the locale() can be fairly time-consuming, as lots of machinery must be torn down and rebuilt for each change.

log(sourceNum)

Operator Type: Function [other Function type actions]
Operator Scope of Action: Item [operators of similar scope]
Operator Purpose: Mathematical [other Mathematical operators]
Operator First Added: Baseline
Operator Last Altered: As at baseline

log(sourceNum)

The operator **log()** computes the natural logarithm of its **sourceNum** number, which should be a number, a numeric attribute, or an expression that can be interpreted as a number.

```
$MyNumber = log(3);
```

returns '1.098612289' for an input of 3.

lowercase(dataStr)

Operator Type: Function [other Function type actions]
Operator Scope of Action: Item [operators of similar scope]
Operator Purpose: Formatting [other Formatting operators]
Operator First Added: Baseline
Operator Last Altered: As at baseline
Operator Has Newer Dot-Operator Variant: Yes

lowercase(dataStr)

The contents of text **dataStr** are returned entirely in lower case.

If $MyString is "Hello World":

```
$MyString = lowercase($MyString);
```

would set $MyString to "hello world".

Functionally equivalent to String.lowercase().

max(numberList)

Operator Type: Function [other Function type actions]
Operator Scope of Action: List [operators of similar scope]
Operator Purpose: Mathematical [other Mathematical operators]
Operator First Added: Baseline
Operator Last Altered: As at baseline
Operator Has Newer Dot-Operator Variant: Yes

max(numberList)

The max() operator returns the largest item in a list or the values or a **List** or **Set** data type attribute. As it operates off a list it may be more convenient to use the newer List/Set.max chained operator.

Both the operators max() and min() use lexical comparison in most cases, but numeric comparison if the context is numeric (i.e. the reference is a Number-type attribute) and/or all list items are numbers. Thus:

```
$Width=max("1;100;2");
```

Since "Width" is numeric, max() will be return 100.

```
$Name=max("1;100;2");
```

Since the attribute "Name" is a string, max() will return 2.

If you do not have a set, create one on the fly using list(), collect() or collect_if():

```
$MyMax = max(list($DateA,$DateB,$DateC));
$MyMax = max(collect(descendants,$Modified));
$MyMax = max(collect_if(all,$MyNum>0,$MyNum));
```

This allows export via ^value^:

^value(max(collect(descendants,$Modified)))^

When using Date-type data bear in mind that the unset Date default is "never" and that "never' is always before (i.e. less than) any set date. So to use max() with dates, filter out the unset values:

```
$MyMax = max(collect_if(descendants,($Modified!="never"),$Modified));
```

To use max() with a list of expressions, use list():

Works: `$MyNumber = max(list(4+2,9+6));` (output: 15)

For more complex examples, where list items are action code expressions, it may be necessary to use eval() to wrap each list item expression e.g. `list(eval(expressionA),eval(expressionB))`.

min(numberList)

Operator Type: Function [other Function type actions]
Operator Scope of Action: List [operators of similar scope]
Operator Purpose: Mathematical [other Mathematical operators]
Operator First Added: Baseline
Operator Last Altered: As at baseline
Operator Has Newer Dot-Operator Variant: Yes

min(numberList)

The min() operator returns the smallest item in a list or the values of a **List** or **Set** data type attribute. As it operates off a list it may be more convenient to use the newer List/Set.min chained operator.

Both the operators max() and min() use lexical comparison in most cases, but numeric comparison if the context is numeric (i.e. the reference is a Number-type attribute) and/or all list items are numbers. Thus:

```
$Width=min("100;2;70");
```

Since "Width" is numeric, min() will be return 2.

```
$Name=min("100;2;70");
```

Since the attribute "Name" is a string, min() will return 100.

If you do not have a set, create one on the fly using list(), collect() or collect_if():

```
$FistDate = min(list($DateA,$DateB,$DateC));
$MyMin = min(collect(descendants, $Modified));
$MyMin = min(collect_if(all,$MyNum>0,$MyNum));
```

This allows export via ^value^:

^value(min(collect(descendants,$Modified)))^

When using Date-type data bear in mind that the unset Date default is "never" and that "never' is always before (i.e. less than) any set date. So to use min() with dates, filter out the unset values:

```
$MyMin = min(collect_if(descendants,($Modified!="never"),$Modified));
```

To use min() with a list of items that are attributes or expressions, use list():

Works: `$MyNumber = min(list(4+2,9+6));` (output: 6)

For more complex examples, where list items are action code expressions, it may be necessary to use eval() to wrap each list item expression e.g. `list(eval(expressionA),eval(expressionB))`.

minute(aDate[, minutesNum])

Operator Type: Function [other Function type actions]
Operator Scope of Action: Item [operators of similar scope]
Operator Purpose: Date-time [other Date-time operators]
Operator First Added: Baseline
Operator Last Altered: As at baseline
Operator Has Optional Arguments: [More on optional operator arguments]
Operator Has Newer Dot-Operator Variant: Yes

---

minute(aDate[, minutesNum])

Alternatively, use Date.minute.

minute(aDate)

returns the hour element from the **aDate** date/time expression, which may simply be a date-type attribute value.

minute(aDate, minutesNum)

creates a new date based on the **aDate** expression, but in which the minute is **minutesNumDate** is not changed unless **aDate** is an attribute and the attribute is re-setting itself:

```
$MyDateA = minute($MyDate,14); $MyDate is not changed
$MyDate = minute($MyDate,14); $MyDate is changed
```

Examples. If $MyDate is 4 July 2009 09:30, then

```
$MyDateA=minute($MyDate,5);
```

will change $MyDate to 4 July 2009 19:05 whilst leaving $MyDate as 4 July 2009 09:30. However, if the code is self-referring:

```
$MyDate=minute($MyDate,5);
```

will change $MyDate to 4 July 2009 19:05.

Take care using the later self-referring form in a $Rule or agent as it fires every agent update cycle adding 5 minutes each time! Make sure you use a guard agent or conditional query to make the action out of scope after the first application. Or, consider using a Stamp, which only fires once per (manual) application.

minutes(startDate, endDate)

Operator Type: Function [other Function type actions]
Operator Scope of Action: Item [operators of similar scope]
Operator Purpose: Date-time [other Date-time operators]
Operator First Added: Baseline
Operator Last Altered: As at baseline

---

minutes(startDate, endDate)

returns the Number of whole minutes that elapsed between **startDate** and **endDate**. If **endDate** is earlier than **startDate** then the result is negative.

If $DateA has time 12:30 and $DateB has time 14:00, then:

```
$MyNumber = minutes($DateA,$DateB);
```

sets $MyNumber to 90.

Also see days(date1,date2).

mod(sourceNum, modulusNum)

Operator Type: Function [other Function type actions]
Operator Scope of Action: Item [operators of similar scope]
Operator Purpose: Mathematical [other Mathematical operators]
Operator First Added: Baseline
Operator Last Altered: As at baseline

---

mod(sourceNum, moduloNum)

mod() returns modulo of a number **sourceNum** as set by number **modulusNum**, i.e. the integer remainder of **sourceNum** divided by **modulusNum**. Thus:

```
$MyNumber = mod(11,3);
```

sets $MyNumber to 2.

month(aDate[, monthsNum])

Operator Type: Function [other Function type actions]
Operator Scope of Action: Item [operators of similar scope]
Operator Purpose: Date-time [other Date-time operators]
Operator First Added: Baseline
Operator Last Altered: As at baseline
Operator Has Optional Arguments: [More on optional operator arguments]
Operator Has Newer Dot-Operator Variant: Yes

---

month(aDate[, monthsNum])

Alternatively, use Date.month.

month(aDate)

returns the month from the **aDate** date/time expression, which may simply be a date-type attribute value.

month(aDate,value)

creates a new date based on the **aDate** date/time expression, but in which the month is **monthsNumDate** is not changed unless theDate is an attribute and the attribute is re-setting itself:

```
$MyDateA = month($MyDate,14);
```

$MyDate is unaltered

```
$MyDate = month($MyDate,14);
```

$MyDate is changed

Examples. If $MyDate is July 4,2009, then

```
$MyDate=month($MyDate,5);
```

will change $MyDate to May 4, 2009.

months(startDate, endDate)

Operator Type: Function [other Function type actions]
Operator Scope of Action: Item [operators of similar scope]
Operator Purpose: Date-time [other Date-time operators]
Operator First Added: Baseline
Operator Last Altered: As at baseline

---

months(startDate, endDate)

returns the Number of whole months that have elapsed between **startDate** and **endDate**. If **endDate** is earlier than **startDate** then the result is negative.

If $DateA is 3 January 2016 and $DateB is 9 April 2016, then:

```
$MyNumber = months($DateA,$DateB);
```

sets $MyNumber to 4.

Also see days(date1,date2).

neighbors(scope, distanceNum[, linkTypeStr])

Operator Type: Function [other Function type actions]
Operator Scope of Action: Group [operators of similar scope]
Operator Purpose: Data manipulation [other Data manipulation operators]
Operator First Added: Baseline
Operator Last Altered: As at baseline
Operator Uses Scoped Arguments: [More on scoped arguments in Action Code]
Operator Has Optional Arguments: [More on optional operator arguments]

---

neighbors(scope, distanceNum)

Returns a list of the path(s) of all those notes that can be reached by following exactly **distanceNum** links *outward* from the designated note or notes. Only the shortest possible path between two notes is considered. The operator will report links from the source node to itself, i.e. self-links. Prototype-type links are ignored. For example:

```
$MyList = neighbors(this, 3);
```

Returns the set of notes that are connected from this note by *exactly* three (outbound) links.

neighbors(scope, distanceNum, linkTypeStr)

Returns the set of notes that are connected by exactly **distanceNum** links *outward* from the designated note or notes, but within those links considering only links of the specified link type **linkTypeStr** to filter on unnamed links use the type "*untitled".For example:

```
$MyList = neighbors(children, 2, "example");
```

Returns the set of notes that are connected from any of this note's children by by *exactly* two (outbound) links of the link type "example".

Different variations of 'neighbor' operators

There are essentially two different subtypes within the 4 operators:

- **Direction**. If the operator name includes a '2' the direction of connecting links are ignored. Otherwise, only outbound links are assessed.
- **Distance**. If the operator name includes 'Within' then all notes at distances between 1 and **distanceNum** are considered. Otherwise, only notes at *exactly* **distanceNum** are considered.

neighbors2(scope, distanceNum[, linkTypeStr])

Operator Type: Function [other Function type actions]
Operator Scope of Action: Group [operators of similar scope]
Operator Purpose: Data manipulation [other Data manipulation operators]
Operator First Added: Baseline
Operator Last Altered: As at baseline
Operator Uses Scoped Arguments: [More on scoped arguments in Action Code]
Operator Has Optional Arguments: [More on optional operator arguments]

---

neighbors2(scope, distanceNum)

Returns a list of the path(s) of all those notes that can be reached by following exactly **distanceNum** links from the designated note or notes, regardless of the direction of those links (i.e. *whether inbound or outbound*). Only the shortest possible path between two notes is considered. The operator will report links from the source node to itself, i.e. self-links. Prototype-type links are ignored. For example:

```
$MyList = neighbors2(this, 3);
```

Returns the set of notes that are connected from this note by *exactly* three links of either direction.

neighbors2(scope, distanceNum, linkTypeStr)

Returns the set of notes that are connected by exactly **distanceNum** links from the designated note or notes, regardless of the direction of *a link* (i.e. *whether inbound or outbound*) and within those links considering only links of the specifiec link type **linkTypeStr**. Unnamed links are specified as the type "*untitled". For example:

```
$MyList = neighbors2(children, 2, "example");
```

Returns the set of notes that are connected from any of this note's children by by *exactly* two links of any direction but also of the link type "example".

Different variations of 'neighbor' operators

There are essentially two different subtypes within the 4 operators:

- **Direction**. If the operator name includes a '2' the direction of connecting links are ignored. Otherwise, only outbound links are assessed.
- **Distance**. If the operator name includes 'Within' then all notes at distances between 1 and **distanceNum** are considered. Otherwise, only notes at *exactly* **distanceNum** are considered.

neighbors2Within(scope, distanceNum[, linkTypeStr])

Operator Type: Function [other Function type actions]
Operator Scope of Action: Group [operators of similar scope]
Operator Purpose: Data manipulation [other Data manipulation operators]
Operator First Added: Baseline
Operator Last Altered: As at baseline
Operator Uses Scoped Arguments: [More on scoped arguments in Action Code]
Operator Has Optional Arguments: [More on optional operator arguments]

---

neighbors2Within(scope, distanceNum)

Returns a list of the path(s) of all those notes that can be reached by following between 1 and **distanceNum** links from the designated note or notes, regardless of the direction of those links (i.e. *whether inbound or outbound*). Only the shortest possible path between two notes is considered. The operator will report links from the source node to itself, i.e. self-links. Prototype-type links are ignored.

The starting note or notes are considered to be connected to themselves.

For example:

```
$MyList = neighbors2Within(this, 3);
```

Returns the set of notes that are connected from this note by between 1–3 (outbound) links.

neighbors2Within(scope, distanceNum, linkType)

Returns the set of notes that are connected by following between 1 and **distanceNum** links from the designated note or notes, regardless of the direction of those links (i.e. *whether inbound or outbound*), but within those links considering o links of the specified link type **linkTypeStr**.

The starting note or notes are considered to be connected to themselves. To filter on unnamed links use the type "*untitled". For example:

```
$MyList = neighbors2Within(children, 2, "example");
```

Returns the set of notes that are connected from any of this note's children by by between 1–2 (outbound) links of the link type "example".

Different variations of 'neighbor' operators

There are essentially two different subtypes within the 4 operators:

- **Direction**. If the operator name includes a '2' the direction of connecting links are ignored. Otherwise, only outbound links are assessed.
- **Distance**. If the operator name includes 'Within' then all notes at distances between 1 and **distanceNum** are considered. Otherwise, only notes at *exactly* **distanceNum** are considered.

neighborsWithin(scope, distanceNum[, linkTypeStr])

Operator Type: Function [other Function type actions]
Operator Scope of Action: Group [operators of similar scope]
Operator Purpose: Data manipulation [other Data manipulation operators]
Operator First Added: Baseline
Operator Last Altered: As at baseline
Operator Uses Scoped Arguments: [More on scoped arguments in Action Code]
Operator Has Optional Arguments: [More on optional operator arguments]

---

neighborsWithin(scope, distanceNum)

Returns a list of the path(s) of all those notes that can be reached by following between 1 and **distanceNum** links *outward* from the designated note or notes. Only the shortest possible path between two notes is considered. The operator wil

report links from the source node to itself, i.e. self-links. Prototype-type links are ignored.

The starting note or notes are considered to be connected to themselves.

For example:

```
$MyList = neighborsWithin(this, 3);
```

Returns the set of notes that are connected from this note by between 1–3 (outbound) links.

neighborsWithin(scope, distanceNum, linkTypeSTr)

Returns the set of notes that are connected by following between 1 and **distanceNum** links *outward* from the designated note or notes, but considering only links of the specified link type **linkTypeStr**.

The starting note or notes are considered to be connected to themselves. To filter on unnamed links use the type "*untitled". For example:

```
$MyList = neighborsWithin(children, 2, "example");
```

Returns the set of notes that are connected from any of this note's children by by between 1–2 (outbound) links of the link type "example".

Different variations of 'neighbor' operators

There are essentially two different subtypes within the 4 operators:

- **Direction**. If the operator name includes a '2' the direction of connecting links are ignored. Otherwise, only outbound links are assessed.
- **Distance**. If the operator name includes 'Within' then all notes at distances between 1 and **distanceNum** are considered. Otherwise, only notes at *exactly* **distanceNum** are considered.

notify(headlineStr[, detailsStr, deliveryDateTime])

Operator Type: Function [other Function type actions]
Operator Scope of Action: Item [operators of similar scope]
Operator Purpose: Data manipulation [other Data manipulation operators]
Operator First Added: Baseline
Operator Last Altered: As at baseline
Operator Has Optional Arguments: [More on optional operator arguments]

---

notify(headlineStr[, details, deliveryTime])

notify() lets notes post notifications to the host Mac's User Notification Center.

headlineStr is a string or string expression, and will be the headline of the notification.

```
notify("Hello world");
```

The optional **detailsStr** is another string or string expression, and provides an explanation of the notification.

```
notify("Hello world", "Some descriptive text");
```

The optional **deliveryDateTime** is a date or date expression that represents a delivery time in the future; if this argument is omitted or is already past, the notification is shown immediately.

```
notify("Hello world", "Some descriptive text",date("now + 1 hour"));
```

Number.ceil()

Operator Type: Function [other Function type actions]
Operator Scope of Action: Item [operators of similar scope]
Operator Purpose: Mathematical [other Mathematical operators]
Operator First Added: 9.5.2
Operator Last Altered: As at baseline

Syntax note: *Operators without any defined mandatory arguments may omit their empty closing parentheses*

Number.ceil()

From v9.5.2, Number.ceil() rounds the source number value of **sourceNum** up to next whole integer.

Number.floor()

Operator Type: Function [other Function type actions]
Operator Scope of Action: Item [operators of similar scope]
Operator Purpose: Mathematical [other Mathematical operators]
Operator First Added: 9.5.2
Operator Last Altered: As at baseline

Syntax note: *Operators without any defined mandatory arguments may omit their empty closing parentheses*

Number.floor()

From v9.5.2, Number.floor() rounds the source number value of **sourceNum** down to previous whole integer. This is a dot-operator

See also Number.ceil() and Number.round().

If $MyNumber is 3.9 then:

```
$MyNumber = $MyNumber.floor();
```

sets $MyNumber to 3. Note unlike normal rounding the value is set downwards to the next integer (i.e. whole number).

Number.format(decimalsNum[, widthNum, padStr]|formatStr)

Operator Type: Function [other Function type actions]
Operator Scope of Action: Item [operators of similar scope]
Operator Purpose: Formatting [other Formatting operators]
Operator First Added: Baseline
Operator Last Altered: 9.5.0
Operator Has Optional Arguments: [More on optional operator arguments]

Number.format(decimalsNum[, widthNum, padStr]|formatStr)

Number.format(decimalsNum, widthNum, padStr)

Returns **Number** as a String, formatted to **decimalsN** decimal places.

If **widthNum** is supplied, Number returned additionally left padded with spaces so that **widthNum** equals the sum of: [padding spaces]+[minus sign]+integer number(s)+decimal point+[decimal numbers]. Note that with **widthNum**, decimal character is *not* counted as part of the number. The presence of a minus sign is allowed for.

This function supplements the existing format() and Number.precision() functions.

For example, if $MyNumber is 3.1415927, then

`$MyString = $MyNumber.format(2);` is "3.14"

`$MyString = $MyNumber.format(0);` is "3"

`$MyString = $MyNumber.format(2,7);` is " 3.14" (3 left padding spaces + 1 integer number + decimal point + 3 decimal numbers = 7)

But if $MyNumber is negative, e.g. -3.1415927, then

`$MyString = $MyNumber.format(2,7)` is " -3.14" (2 left padding spaces + minus sign + 1 integer number + decimal point + 3 decimal numbers = 7)

Literal numbers, e.g. 3.1415927, can also be worked with:

`$MyString = 5.1415927.format(2);` is "5.14"

The above works but the following syntax may seem less ambiguous by using parentheses to delimit the literal number:

`$MyString = (5.1415927).format(2);` is "5.14"

`$MyString = (5.1415927).format(1,5);` is ' 5.1' (two left padding spaces + 1 integer number + decimal point + 1 decimal number = 5)

If a group of numbers are being formatted so as to vertically decimal-point align as a column figures, e.g. financial data, it is necessary to know the widthN of 'longest' number to be used, remembering that a negative number adds one to its width count; 45 is width 2, -45 is width 3. In the example below the longest (**widthNum**) number in a set of currency figures has been worked out stored in a user Number attribute $MaxNumLen. Being currency, 2 decimal places will be enforced, and each number can be evaluated by a common formatting:

```
[the number].format(2,$MaxLenNum)
```

If the widthN for a set of numbers cannot easily be assessed, an alternate option is simply to use a number known to be bigger than all likely width valid. Thus *every* number, including the longest, gets left-padded but all end up correctly aligned. In the latter example if all numbers are always likely to be less than 20, then $MaxLenNumber could be set to 20, or simply used directly:

```
[the number].format(2,$MaxLenNum)
```

There is no easy way to sort a list of numbers on size (i.e. their **widthNum**), other than by looping the list via List.each() transforming each to a string (using zero decimal places!), saving the String.size of each of these as a number in a new lis then List.nsort and take the last item, .at(-1). As .nsort() sorts on ascending numerical order, the latter will be the size of the longest string (including negative numbers) in the original list. Assuming $MySizeList has all the size strings:

```
$MaxLenNumber = ($MySizeList.nsort).at(-);
```

Why leave out decimal places when coercing the numbers to strings? 1234.56 is a bigger number than 12.34567, but the latter is the bigger size. However, 1234 is both bigger and 'wider' than 12.

As can be seen, just setting a large arbitrary **widthNum** might save a lot of messing about!

If the optional **padStr** is given, this specifies the character used for padding: use of a **widthNum** argument is expected. The default is a space:

```
$MyString = 7.format(0,3); gives " 7"
$MyString = 7.format(0,3,"0"); gives "007"
$MyString = 7.format(0,3,"#"); gives "##7"
```

Number.format("formatString")

An alternate usage is to supply a quoted **formatString** argument that is always enclosed in double quotes.

Supported **formatString** values (described in more detail further below) are:

- `"l"` (lowercase letter 'L') gives **Number** as a string in (OS) locale-dependent group & decimal delimiters.
- `"$"` formats **Number** to a string in the local currency to two decimal points.
- `"$0"` formats **Number** to a string in the local currency rounded to the neared whole major unit (i.e. whole dollars only not dollars.cents.)
- (v9.5.0) `"X"` converts **Number** to a string in Roman numerals
- (v9.5.01) `"o"` (lowercase letter 'O') converts **Number** to a string in the localised ordinal

Localised, locale-dependent, numbers. Using **formatString** `"l"` (lowercase letter 'L') will return a string of the source **Number** formatted with (OS) locale-dependent group & decimal delimiters. For example, for the US locale these are a comma and a period; in other locales they may vary. For example, if $MyNumber is 4562781.4, and it is desired to display it as a string with thousands delimited:

```
$MyString = $MyNumber.format("l"); gives "4,562,781.4"
```

Of course, depending on the users local, the delimiter may be something else. For instance in a German locale setting, it would be "4.562.781,4". For more on such difference see Wikipedia's article on ' Decimal Separators'.

Currency formatting. Two currency-related format strings that can be used, again locale-based, to turn a **Number** into number string with a currency symbol prefix. Using **formatString** `"$"` formats the number to the local currency to two decimal points. Thus if $MyNumber is `1246.878`:

```
$MyString = $MyNumber.format("$");
```

sets $MyString to "$1,246.88" isn the US, "£1,246.88" in UK, "€1,246.88" in France, etc. Note how the format also respects the local thousands separate, adding a comma after the initial digit.

Use format string `"$0"` if the currency string is needed rounded to the neared whole major unit, e.g. whole dollars and not dollars.cents, :

```
$MyString = $MyNumber.format("$0");
```

which sets $MyString to "$1,247" isn the US, "£1,247" in UK, "€1,247" in France, etc.

Roman Numerals. From v9.5.0, **formatString** `"X"` converts **Number** to a string in Roman numerals. For example, if $MyNumber is `3`:

```
$MyString = $MyNumber.format("X")
```

gives a $MyString value of "`III`".

Ordinals. From v9.5.0, **formatString** `"o"` (lowercase letter 'O') converts **Number** to a string of localised ordinal value. For example, if $MyNumber is `3`:

```
$MyString = $MyNumber.format("o")
```

gives a $MyString value of "`3rd`" in English but "`3e`" in French, etc.

Number.precision(decimalsNum)

Operator Type: Function [other Function type actions]
Operator Scope of Action: Item [operators of similar scope]
Operator Purpose: Formatting [other Formatting operators]
Operator First Added: Baseline
Operator Last Altered: As at baseline

Number.precision(decimalsNum)

This function makes it easier to format numbers. The function returns the associated Number-type attributes value rounded to **decimalsNum** decimal places as *Number*-type data.. If $MyNumber is 268.3289 then:

```
$MyNumber.precision(2);
```

…sets $MyNumber to 268.33, whilst:

```
$SomeNumber = $MyNumber.precision(2);
```

…sets $OtherNumber to 268.33. The above examples are the equivalent of the older syntax using format():

```
format($MyNumber,2);
```

Literal numbers, e.g. 3.1415927, can also be worked with:

```
$MyNumber = 5.1415927.precision(2); is 5.14
```

The above works but the following syntax may seem less ambiguous by using parentheses to delimit the literal number:

```
$MyNumber =(5.1415927).precision(2); is 5.14
```

Since Number.precision() was added, format() has been supplemented by a Number.format(). The latter, in single input version equates to Number.precision(). However that Number.format() outputs a *String*-type, even if that can be coerced back to a number .

The .precision() function can be used to add trailing zeroes to a decimal. If $MyNumber is 214.40, it will display as 214.4 which can be unhelpful if this actually represents £214.40. $MyNumber.precision(2) will return 214.40 but be careful abou under-the-hood number/string coercion such as can happen in contexts like $DisplayExpression.

```
$MyOtherNumber = $MyNumber.precision(2); gives 214.4
$MyString = $MyNumber.precision(2); gives "214.40"
```

In a $DisplayExpression:

```
$MyNumber.precision(2); gives "214.40"
$Name+" : "+$MyNumber.precision(2); gives "214.40"
$Name+ " : " + sum(children,$MyNumber).precision(2);
```

It may still be easier to use format() as no extra parentheses are required:

```
$Name+ " : " + format(sum(children,$MyNumber),2);
```

Number.round()

Operator Type: Function [other Function type actions]
Operator Scope of Action: Item [operators of similar scope]
Operator Purpose: Mathematical [other Mathematical operators]
Operator First Added: 9.5.2
Operator Last Altered: As at baseline

Syntax note: *Operators without any defined mandatory arguments may omit their empty closing parentheses*

Number.round()

From v9.5.2, Number.round() rounds the value of its chained source number argument to the nearest integer. A partial integer over .50 always round up, everything else rounds down.

See round() for a fuller explanation of the logic and possible usage.

See also Number.ceil() and Number.floor().

originalLinkedFrom(scope[, linkTypeStr])

Operator Type: Function [other Function type actions]
Operator Scope of Action: Item [operators of similar scope]
Operator Purpose: Query Boolean [other Query Boolean operators]
Operator First Added: Baseline
Operator Last Altered: As at baseline
Operator Uses Scoped Arguments: [More on scoped arguments in Action Code]
Operator Has Optional Arguments: [More on optional operator arguments]

originalLinkedFrom(scope[,linkTypeStr])

A test for inbound links. This returns `true` if the current note corresponding to this is linked from the designated **scope**, i.e. an **item** or **group** of items (defining **scope**); this is optionally filtered to only links of type **linkTypeStr**. Put another way, "Does an inbound link exist to the original current note from item(s)?". This is especially useful in agents, where **this** is bound to an alias owned by the agent but the user is interested in links to the original note.

This is effectively only a query term, it returns a Boolean. If you are trying to collect data about the linked note(s), use links() instead.

Ways to define item. The item argument must be quoted unless an attribute reference, e.g. "Some note" vs. $MyString.

Ways to define group. In group scope, a wildcard "*" designator matches all notes and replaces the normal "all" group designator.

Links of type 'prototype' are ignored. If using linkType, you must use the value "*untitled" to match an 'untitled' type link (rather than "" or "untitled").

Thus, to test if any original note using the 'Event' prototype has an inbound link of the 'untitled' link type the agent query would be:

```
$Prototype=="Event" & originalLinkedFrom("*", "*untitled");
```

The logical opposite of this test is originalLinkedTo().

originalLinkedTo(scope[, linkTypeStr])

Operator Type: Function [other Function type actions]
Operator Scope of Action: Item [operators of similar scope]
Operator Purpose: Query Boolean [other Query Boolean operators]
Operator First Added: Baseline
Operator Last Altered: As at baseline
Operator Uses Scoped Arguments: [More on scoped arguments in Action Code]
Operator Has Optional Arguments: [More on optional operator arguments]

originalLinkedTo(scope[,linkTypeStr])

A test for outbound links. This returns `true` if the current note corresponding to this is linked from the designated **scope**, i.e. an **item** or **group** of items (defining **scope**); this is optionally filtered to only links of type **linkTypeStr**. Put another way, "Does an outbound link exist from the original current note to item(s)?". This is especially useful in agents, where **this** is bound to an alias owned by the agent but the user is interested in links to the original note.

This is effectively only a query term, it returns a Boolean. If you are trying to collect data about the linked note(s), use links() instead.

Ways to define item. The item argument must be quoted unless an attribute reference, e.g. "Some note" vs. $MyString.

Ways to define group. In group scope, a wildcard "*" designator matches all notes and replaces the normal "all" group designator.

Links of type 'prototype' are ignored. If using linkType, you must use the value "*untitled" to match an 'untitled' type link (rather than "" or "untitled").

Thus, to test if any original note using the 'Event' prototype has an outbound link of the 'untitled' link type the agent query would be:

```
$Prototype=="Event" & originalLinkedTo("*", "*untitled");
```

The logical opposite of this test is originalLinkedFrom().

play(soundNameStr)

Operator Type: Function [other Function type actions]
Operator Scope of Action: Item [operators of similar scope]
Operator Purpose: Data manipulation [other Data manipulation operators]
Operator First Added: Baseline
Operator Last Altered: As at baseline

play(soundNameStr)

This causes the stated OS alert sound to be played, the sound name being supplied as soundNameStr. To play the default error sound:

```
play("Basso");
```

The **soundNameStr** name is case-sensitive.

pow(sourceNum, powerNum)

Operator Type: Function [other Function type actions]
Operator Scope of Action: Item [operators of similar scope]
Operator Purpose: Mathematical [other Mathematical operators]
Operator First Added: Baseline
Operator Last Altered: As at baseline

pow(sourceNum, powerNum)

pow() returns the **sourceNum** raised to the **powerNum**.

```
$MyNumber = pow(3,4);
```

returns 81.

radians(degreesNum)

Operator Type: Function [other Function type actions]
Operator Scope of Action: Item [operators of similar scope]
Operator Purpose: Mathematical [other Mathematical operators]
Operator First Added: Baseline
Operator Last Altered: As at baseline

radians(degreesNum)

radians() converts its **degreesNum**, in degrees, to *radians*.

See also degrees() which converts an angle in radians to degrees.

```
$MyNumber = radians(90);
```

returns '1.570796327' for an input of 90.

rand([maxNumber])

Operator Type: Function [other Function type actions]
Operator Scope of Action: Item [operators of similar scope]
Operator Purpose: Mathematical [other Mathematical operators]
Operator First Added: Baseline
Operator Last Altered: 9.5.2
Operator Has Optional Arguments: [More on optional operator arguments]

Syntax note: *Operators without any defined mandatory arguments may omit their empty closing parentheses*

rand([maxNumber])

rand()

rand

rand() returns a pseudo-random number between 0 and 1. No argument is required.

rand([maxNumber])

From v9.5.2, supplying an optional parameter **maxNumber**, which must be an integer, returns a random *integer* between 0 and **maxNumber**-1. This can be useful when wanting random list items, noting that list items are addressed using zero-based integers.

N.B. if working with list-based items, consider the List/Set.randomItem() operator (new to v9.5.2), which abstracts away the need to know the size of the list. Essentially it encapsulates calling `list.at(rand(list.count))`, so much easie to write/use

Getting 1-based ranges vs. zero-based ranges of values

Consider the original round() method:

```
$MyNumber=round(rand()*10);
```

$MyNumber will be set to one of *eleven* integers in the range 0 through to 10. But, what if the '10' argument about is actually a child count:

```
$MyNumber=round(rand()*$ChildCount);
```

If the $ChildCount was 10 and $MyNumber were used to fetch a random child using $SiblingOrder($MyNum) the process would fail if $MyNumber were zero as $SiblingOrder numbers from 1. Indeed, in this case the need is for *ten* integers, one through to ten.

Previously, 1-based range can be achieved:

```
$MyNumber=round(rand()*(10-1))+1;
$MyNumber=round(rand()*($ChildCount-1))+1;
```

If the main input is 10, by subtracting 1 the process returns a randomised integer in in the range 0-9 (*ten* numbers) and then adding back 1 shifts the value range to 1-10.

Using the newer **rand(maxNumber)** method the above becomes:

```
$MyNumber=round(rand($ChildCount-1))+1;
```

This is why the List/Set.randomItem() may be an easier clear approach.

require(featureName)

Operator Type: Function [other Function type actions]
Operator Scope of Action: Document [operators of similar scope]
Operator Purpose: Document configuration [other Document configuration operators]
Operator First Added: 9.6.0
Operator Last Altered: As at baseline

---

require(featureName)

New to v9.6.0, the require() operator adds the /Hints container and its subsidiary containers. It is equivalent to selecting Built In Hints from the File menu. If any of the containers already exists, that container is left unchanged. For example:

```
require("Hints");
```

featureName is a mandatory argument. It is a literal string and case-sensitive.

"Hints" is currently the only **featureName** value recognised by **require()**. But, this method allows for other values to be added in the future as need arises.

return

Operator Type: Statement [other Statement type actions]
Operator Scope of Action: Document [operators of similar scope]
Operator Purpose: Data manipulation [other Data manipulation operators]
Operator First Added: Baseline
Operator Last Altered: As at baseline

---

return

The **return** statement is only used inside a function, and indicates that the value (defined by code following the statement) is to be passed back to the code calling the function at which point the function stops. Trailing empty parentheses are _not_ used.

See further explanation of the use of **return** within functions, including how the returned value is coded.

rgb(redNum, greenNum, blueNum)

Operator Type: Function [other Function type actions]
Operator Scope of Action: Item [operators of similar scope]
Operator Purpose: Color [other Color operators]
Operator First Added: Baseline
Operator Last Altered: As at baseline

---

rgb(redNum, greenNum, blueNum)

Creates a Color type: its arguments are integers ranging from 0 to 255 or an expression or attribute reference resolving to a number in this range. The value set is in hex form. For example:

`$MyColor = rgb(255,0,0);` sets a value of "#ff0000"

The individual channels of an RGB colour can be set or read using the '.red', '.green' and '.blue' Color-type data operators.

round(sourceNum)

Operator Type: Function [other Function type actions]
Operator Scope of Action: Item [operators of similar scope]
Operator Purpose: Mathematical [other Mathematical operators]
Operator First Added: Baseline
Operator Last Altered: As at baseline
Operator Has Newer Dot-Operator Variant: Yes

---

round(sourceNum)

round() rounds the value of its **sourceNum** argument to the nearest integer. Note that from v9.5.2, there is now a dot-operator version (with the same behaviour) Number.round().

The Number.format, Number.precision and format() functions all round numbers in this manner when shortening numbers during formatting. A partial integer over .50 always round up, everything else rounds down. Thus:

`$MyNumber = round(4.0);` gives 4
`$MyNumber = round(4.2);` gives 4
`$MyNumber = round(4.5);` gives 5
`$MyNumber = round(4.7);` gives 5

The round function with work on string literal or String attribute values that are just numbers:

`$MyNumber = round("4.2");` gives 4
`$MyNumber = round("4.7");` gives 5
`$MyString = round(4.2);` gives "4"
`$MyString = round(4.7);` gives "5"
`$MyString = round("4.2");` gives "4"
`$MyString = round("4.7");` gives "5"

There are also functions to always force rounding upwards (ceiling)—see ceil(), or to force rounding downwards (floor)—see floor().

For a practical example, assume you would like to round you calculation (upward) to the nearest 100, so if the calculation output is 167 it should be 200, for a result of 540 a result of 600, and so on. This can be done like so:

```
$MyNumber = 100*ceil($MyNumber/100);
```

whereas if you used round(), as in:

```
$MyNumber = 100*round($MyNumber/100);
```

the result would vary depending on whether $MyNumber started above or below the nearest 50. An opposite of the first example, i.e. where everything rounds downwards to the nearest 100 can be done using floor():

```
$MyNumber = 100*floor($MyNumber/100);
```

runCommand(commandStr[, inputsStr, dirStr])

Operator Type:	Function [other Function type actions]
Operator Scope of Action:	Item [operators of similar scope]
Operator Purpose:	Data manipulation [other Data manipulation operators]
Operator First Added:	Baseline
Operator Last Altered:	As at baseline
Operator Uses Regular Expressions:	[More on regular expressions in Tinderbox]
Operator Has Optional Arguments:	[More on optional operator arguments]

---

runCommand(commandStr, inputStr)

runCommand(commandStr, inputStr, dirStr) /runCommand(commandStr, , dirStr)

The operator, **runCommand()**, lets rules and actions use the command line with an assumed working directory of the users OS home folder: see below for using **dirStr** for setting a different context of execution.

The full detail of command line is out of scope for aTbRef and should be researched online. For novices, a good approach is to make a command line work in the macOS Terminal and only then try to use it via **runCommand()**. Similarly, lea customising the input via attribute/variable values until after a fixed string is working as this makes it much easier to troubleshoot the correct issue.

Understanding the latter can help avoid trying to echo data into a command within the ' **commandStr**' string. See the 'quotify' example further below.

runCommand(commandStr)

The operator passes **commandStr** to the OS's Unix shell. The new shell processes **commandStr**. The process's standard output (`stdout`) is returned as the result of calling **runCommand()**. However, *there is no requirement for a left-side argument* if the output of **runCommand()** is needed, e.g. if it is just a true/false success/fail message.

For example, to set the $Text of a note to a listing of all files/folders in the users home directory `~`, run this code as a stamp:

```
$Text = runCommand("ls -a");
```

Whereas a stamp:

```
runCommand("ls -a");
```

would run the same code but the listing would not be used. Not useful in this example but if the need it just to run a script where the reult is not of interest, **runCommand()** can called on its own.

runCommand(commandStr, inputStr)

The operator passes **commandStr** to the OS's Unix shell. The new shell process receives **inputsStr**, if any, as its standard input `stdin`. The process's standard output (`stdout`) is returned as the result of calling **runCommand()**. The above syntax, in one-line command line usage, can also be thought of as: **runCommand(command, stdIn)**.

For example, the `wc` (word count) command can take a file or literal string as an input. On the command line, an `echo` command would be needed to pipe the literal string to `wc`, like so:

```
echo "hello world" | wc -w
```

to get the result '2' (two words in the input string) in Terminal. To do this via **runCommand()**:

```
runCommand("wc -w","hello world");
```

But, you don't see the returned value unless you pass the output of the command to something. a number is expected here so $MyNumber is sensible start:

```
$MyNumber = runCommand("wc -w","hello world");
```

and $MyNumber will now have the value '2'.

Or, if using attributes or variables to pass in the data:

```
var:string vCmd = "wc -w";
var:string vArgs= $MyString; // which has the text whose word count we want
$myNumber = runCommand(vCmd, vArgs);
```

Whether the desired command line tool accepts inputs on `stdIn` is something the user needs to understand before using this command.

IMPORTANT: Note that if using **inputsStr**, to pass *arguments* otherwise normally passed as part of **commandStr**, it is necessary to prefix the command used in **commandStr** with 'xargs' so that the arguments passed via stdin are correctly handled. Thus if using the command "`mv`" in **commandStr** with arguments in **inputStr**, the **commandStr** needs to be "`xargs mv`". This is shown in examples further below under heading "Working around using literal strings as inputs".

Setting the working directory using dirStr

runCommand(commandStr, inputStr, dirStr)

Where is the working directory location assumed for executing scripts ?

If the optional **dirStr** is specified as a POSIX path, it sets the working directory in which **commandStr** is executed. Otherwise, by default, the working directory is the user's home folder , i.e. `~` or `/Users/[shortusername]` or in full form.

If using **dirStr**, an **inputStr** *must* be provided even if only an empty string: **runCommand(commandStr, "", dirStr)**.

dirStr can be passed either as a full path from drive root or using the tilde (~) notation. **dirStr** should not include a forward slash at the end: e.g. "~/test" not ~~"~/test/"~~. The last part of the supplied path should be the folder which is to be the context of execution. Thus "~/test" implies a folder test that is a child folder of the users home folder.

Working around using literal strings as inputs

Any or all of **commandStr**, **inputsStr**, or **dirStr** may use attribute or variable values in place of literal strings. Thus:

```
runCommand("xargs mv", '"x/tb 1.png" "y/tb 1.png"', "~/test")
```

will move file 'tb 1.png' from sub-folder 'x' at path '~/test' to sub-folder 'y'. (*Note: the source and target paths need quoting because the file 'tb 1.png' has a space in its name and the paths would otherwise be mis-parsed by the shell. If the various strings.*). Placing the start and end relative paths into $SourcePath and $TargetPath, the command above could also be written clearly/verbosely as:

```
var:string vCmd = "mv";
var:string vCmd += '"'+$SourcePath+'" "'+$TargetPath+'"';
vFolder = "~/test";
runCommand(vCmd, "", vFolder);
```

or, the same but passing some of the arguments as inputs using the `xargs` method:

```
var:string vCmd = "xargs mv";
var:string vArgs = '"'+$SourcePath+'" "'+$TargetPath+'"';
vFolder = "~/test";
runCommand(vCmd, vArgs, vFolder);
```

Further examples

If a note called "Jane Doe" is dropped on a container with this OnAdd action:

```
$MyResult = runCommand("sendmail -f "+$Email+" "+$Email(parent), "Subject:"+$Name+"\nHello\n.")
```

This assumes both dropped and container notes have a valid email address in Email. If so, Jane Doe will get an email with subject line "Jane Doe" and body text "Hello"; the email will be from the dropped note's $Email address and to the container's $Email address. In the above example:

commandStr: `sendmail - f jade@doe.com someone@other.com'`

inputsStr (i.e. via `stdin`): `"Subject: Jane Doe\nHello\n."`

User attribute 'MyResult' will receive any message back from standard output.

The **runCommand()** operator does not require a left-side to the expression where the result of the command line is not needed by Tinderbox. Thus the same example as above can run as:

```
runCommand("sendmail -f "+$Email+" "+$Email(parent), "Subject:"+$Name+"\nHello\n.")
```

Bear in mind that in this latter case there is no way of knowing if the command executed successfully.

To use external commands like above you may need to check the encoding of the strings you extract from your TBX attributes. Do not forget to allow for characters like spaces/quotes/apostrophes in attribute values; these will invariably nee escaping for safe use in a command line using operators like urlEncode():

```
$MyResult=runCommand("/usr/bin/curl -d 'status="+urlEncode($Name)+"' -u myusername:mypassword https://twitter.com/statuses/update.atom");
```

In the above, if the value of $Name were "Mark's project", the use of urlEncode() will ensure the string passed to the command line is actually "Mark%27s%20project".

exportedString() can also help with ensuring the necessary encoding, allowing a template to be used to help with formatting the string passed into the command.

Exactly where you do/do not need to encode attribute values will depend on the syntax of the particular operation you are performing.

To help get around issues of quoting, a note's text (or other string attribute) can be used for either or both arguments. Consider a note called 'quotify' holding this command line:

```
sed 's:"\([^"]*\)":"\1":g'
```

A stamp might then hold this code:

```
$Text = runCommand($Text("quotify"),$Text);
```

This effectively adds a menu item to the Stamps to change ordinary double quotes to their 'smart form', allowing the process to be run *once*, on demand, avoiding repeat use every agent cycle as would happen if using a rule or agent action

Dealing with inline quote characters

Because input arguments are parsed for regular expressions (by the OS, not Tinderbox), it may be possible to use the '\dnn' form described here to work around the lack of escaping from single double quotes within strings.

Legacy only

Using just command on its own is akin to using the back-tick action syntax.

seconds(startDate, endDate)

Operator Type: Function [other Function type actions]
Operator Scope of Action: Item [operators of similar scope]
Operator Purpose: Date-time [other Date-time operators]
Operator First Added: Baseline
Operator Last Altered: As at baseline

seconds(startDate, endDate)

returns the Number of whole seconds that elapsed between **startDate** and **endDate**. If **endDate** is earlier than **startDate** then the result is negative.

If $DateA has time 12:30:00 and $DateB has time 12:31:30, then:

```
$MyNumber = seconds($DateA,$DateB);
```

sets $MyNumber to 90.

select()

Operator Type: Function [other Function type actions]
Operator Scope of Action: Group [operators of similar scope]
Operator Purpose: Data manipulation [other Data manipulation operators]
Operator First Added: 9.6.0
Operator Last Altered: As at baseline
Operator Uses Scoped Arguments: [More on scoped arguments in Action Code]

select()

From v9.6.0, the **select()** operator de-selects all items currently selected in the document. The usage is simple:

```
select();
```

Because using select with arguments has a distinct and different function is it listed separately: select(scope).

select(scope)

Operator Type: Function [other Function type actions]
Operator Scope of Action: Group [operators of similar scope]
Operator Purpose: Data manipulation [other Data manipulation operators]
Operator First Added: 9.6.0
Operator Last Altered: As at baseline
Operator Uses Scoped Arguments: [More on scoped arguments in Action Code]
Operator Has Newer Dot-Operator Variant: Yes

select(scope)

From v9.6.0, the **select(scope)** operator allows the document's current (UI) focus to be shifted. The **scope** argument defines a list of *one or more* notes. The argument may be a literal string, or the value of an attribute or variable.

```
select([/Note A;/Note B]);
select($MyList);
select(vItems);
```

This specialist operator assist in the scenario where, whilst running action code, it is necessary to change the selection such that the locus of 'this' changes. Some action code operators only address the currently selected item(s). The **select(scope)** operator allows the selection to be changed on the fly without the user having to do so via the UI.

Because using select without arguments has a distinct and different function is it listed separately: select().

show(msgString[, backgroundColor[,colorString]])

Operator Type: Function [other Function type actions]
Operator Scope of Action: Item [operators of similar scope]
Operator Purpose: Data manipulation [other Data manipulation operators]
Operator First Added: 9.5.2
Operator Last Altered: As at baseline
Operator Has Newer Dot-Operator Variant: Yes

show(msgString[, backgroundColor[,colorString]])

New to v9.5.2, the **show(msgString)** function allows a plain-text message to be shown in the front window's message placard.

The source text, **msgString**, may be a literal string, variable or a String-type attribute value.

The **backgroundColor** argument may be used on its own, but if the optional **colorString** arguments is used, then the **backgroundColor** argument *must* be given as well.

The full features and limitations of messages sent to the placard are described separately under the Message placards article.

The String.show() operator offers a dot-operator alternative.

similarTo(item[, notesNum])

Operator Type: Function [other Function type actions]
Operator Scope of Action: Item [operators of similar scope]
Operator Purpose: Query Boolean [other Query Boolean operators]
Operator First Added: Baseline
Operator Last Altered: As at baseline
Operator Has Optional Arguments: [More on optional operator arguments]

similarTo(item,notesNum)

Returns Boolean `true` if **item** is amongst the (optionally **notesNum** notes) most similar to the current note.

The **item** argument must be quoted unless an attribute reference. Ways to define **item**.

Similarity is based on several factors, including:

- the text of the note
- the note title
- any text contained in user attributes

In addition, weighting is applied for:

- notes having the same prototype
- notes having roughly similar amounts of text

This data replicates that seen in the similar tab of Get Info and in the export code ^similarTo()^.

Legacy issues

This operator replaces the legacy #similarTo query operator and what in old version's agent query creation pop-ups and Find dialogs was listed as "is similar to".

sin(sourceNum)

Operator Type: Function [other Function type actions]
Operator Scope of Action: Item [operators of similar scope]
Operator Purpose: Mathematical [other Mathematical operators]
Operator First Added: Baseline
Operator Last Altered: As at baseline

---

sin(sourceNum)

sin() converts its **sourceNum**, in *radians*, to the sine of that value.

```
$MyNumber = sin(3)
```

returns '0.1411200081' for an input of 3 radians.

sqrt(sourceNum)

Operator Type: Function [other Function type actions]
Operator Scope of Action: Item [operators of similar scope]
Operator Purpose: Mathematical [other Mathematical operators]
Operator First Added: Baseline
Operator Last Altered: As at baseline

---

sqrt(sourceNum)

sqrt() computes the square root of its **sourceNum**.

```
$MyNumber = sqrt(9);
```

sets $MyNumber to 3.

stamp([scope,]stampName)

Operator Type: Function [other Function type actions]
Operator Scope of Action: Group [operators of similar scope]
Operator Purpose: Data manipulation [other Data manipulation operators]
Operator First Added: Baseline
Operator Last Altered: 9.0.0
Operator Has Optional Arguments: [More on optional operator arguments]

---

stamp([scope,]stampName)

stamp(stampName)

This operator applies the stamp **stampName** is applied to the `this` note, i.e. the current note. So:

```
stamp("Do Stuff");
```

runs the stamp called "Do Stuff" on the current note.

stamp(scope, stampName)

scope is an optional argument that can be a single note or a group or designator defined in any of the normal methods (literal, designator, expression).

If a **scope** argument is supplied the stamp named **stampName** is applied discretely to each of **scope** items (defining **scope**). So:

```
stamp("SomeNote","Do Stuff");
```

runs the stamp called "Do Stuff" on the note with the unique $Name "Some Note". Or:

```
stamp($MyList,"Do Stuff");
```

runs the stamp called "Do Stuff" on every note described by an item in the list stored in the value of $MyList for this note.

String.beginsWith(matchStr)

Operator Type: Function [other Function type actions]
Operator Scope of Action: Item [operators of similar scope]
Operator Purpose: Data manipulation [other Data manipulation operators]
Operator First Added: Baseline
Operator Last Altered: As at baseline

String.beginsWith(matchStr)

String.beginsWith() returns Boolean `true` if the chained string begins with a specific substring.

For example, if $MyString is "I do not like green eggs and ham":

`$MyString.beginsWith("I do")` returns `true`
`$MyString.beginsWith("Hello")` returns `false`

Also, with string literals:

`("There are gentlemen now abed.").beginsWith("There are")` returns `true`
`("There are gentlemen now abed.").beginsWith("Hello")` returns `false`

This operator searches for literal strings, not regular expressions. Matches are case sensitive

`("There are gentlemen now abed.").beginsWith("there are")` returns `false`

See also String.endsWith(). If you need to search a String for a regular expression, use String.contains().

String.capitalize()

Operator Type: Function [other Function type actions]
Operator Scope of Action: Item [operators of similar scope]
Operator Purpose: Formatting [other Formatting operators]
Operator First Added: Baseline
Operator Last Altered: As at baseline

Syntax note: *Operators without any defined mandatory arguments may omit their empty closing parentheses*

String.capitalize()

String.capitalize

Returns the referenced string, transforming the initial letter of each word to a capital letter and leaving all other characters unchanged.

The function can be chained both to string data and to string literals:

```
$MyString.capitalize()
"my new title".capitalize()
```

The latter gives "My New Title".

The trailing parentheses may be omitted:

```
$MyString.capitalize
```

If 'title case' is required from a mixed case string, chain with .lowercase():

```
$MyString = "my nEW title".lowercase.capitalize;
```

sets $MyString to "My New Title".

Bear in mind the latter will not deal with all capital acronyms (UPS, UNHCR) or CamelCase words (AstroTurf, FedEx).

Functionally equivalent to capitalize().

The .capitalize() method may also be used on Lists or Sets. Consider the list [Ant;BEE;Cow] when stored in $MyList:

```
$MyList = $MyList.capitalize;
```
… giving [Ant;Bee;Cow].

String.captureJSON()

Operator Type: Function [other Function type actions]
Operator Scope of Action: Item [operators of similar scope]
Operator Purpose: Stream parsing [other Stream parsing operators]
Operator First Added: Baseline
Operator Last Altered: As at baseline

Syntax note: *Operators without any defined mandatory arguments may omit their empty closing parentheses*

String.captureJSON()
String.captureJSON
Attempts to parse the string as fully/partially JSON and fails if unsuccessful. The parsed JSON is saved as the current JSON (stream) object.
Essentially, this re-scopes the current stream so that it contains only the contents of the first section of JSON detected within the original stream. The focus of the stream parsing is set to the beginning of the extracted JSON code.
Only one JSON object may be current at any time. If the source stream contains multiple discrete JSON code sections, only the first is detected/used.

String.captureLine([targetAttributeStr])

Operator Type: Function [other Function type actions]
Operator Scope of Action: Item [operators of similar scope]
Operator Purpose: Stream parsing [other Stream parsing operators]
Operator First Added: Baseline
Operator Last Altered: As at baseline
Operator Has Optional Arguments: [More on optional operator arguments]

Syntax note: *Operators without any defined mandatory arguments may omit their empty closing parentheses*

String.captureLine(targetAttributeStr)
String.captureLine()
String.captureLine
Optionally stores the rest of the line in the specified **targetAttributeStr**, a quoted name of an attribute, and returns the string that follows this line and the **String** moves forwards to the end of the current line.
If the **targetAttributeStr** is omitted the **String** is advanced to the next line without any data being saved.
The attribute given for **targetAttribute** can be a quoted literal string, e.g. `("Text")` for $Text. Or it can be an unquoted variable/loopVariable, e.g. `(vString)` for a variable 'vString'.

String.captureNumber([targetAttributeStr])

Operator Type: Function [other Function type actions]
Operator Scope of Action: Item [operators of similar scope]
Operator Purpose: Stream parsing [other Stream parsing operators]
Operator First Added: Baseline
Operator Last Altered: As at baseline
Operator Has Optional Arguments: [More on optional operator arguments]

Syntax note: *Operators without any defined mandatory arguments may omit their empty closing parentheses*

String.captureNumber("targetAttribute")
String.captureNumber()
String.captureNumber
Searches forward in the **String** and matches the first detected number in the **String** and optionally stores that number in the attribute specified **targetAttributeStr**, a quoted name of an attribute. It then returns the string that follows that number (to any left side recipient) whilst the **String** is advanced to a position after the number. The operator fails if the match is empty or entirely white space, with no result.
A number is assumed to be any character continuous sequence of one or more number characters between 0–9 (zero through nine). Formatted numbers will not be fully matched. Any further text after the number is ignored, both for the captu
and advancing the **String**.
Thus sequence "There are 1234 items" makes a 'number' match of 1234. But for "There are 1,234 items" or "There are 1.234 items" the match will be the first number (1) only as it is followed by a non-number character.
If the **targetAttribute** is omitted the **String** is advanced to after the number without any data being saved.
The attribute given for **targetAttribute** can be a quoted literal string, e.g. `("Text")` for $Text. Or it can be an unquoted variable/loopVariable, e.g. `(vString)` for a variable 'vString'.

String.captureRest([targetAttributeStr])

Operator Type: Function [other Function type actions]
Operator Scope of Action: Item [operators of similar scope]
Operator Purpose: Stream parsing [other Stream parsing operators]
Operator First Added: Baseline
Operator Last Altered: As at baseline
Operator Has Optional Arguments: [More on optional operator arguments]

Syntax note: *Operators without any defined mandatory arguments may omit their empty closing parentheses*

String.captureRest(targetAttributeStr)
String.captureRest()
String.captureRest
Stores all of the **String** from the current position to the end in the specified **targetAttributeStr**, a quoted name of an attribute, and returns an empty string.
If the **targetAttributeStr** is omitted the **String** is advanced to the end so processing ceases.
The attribute given for **targetAttributeStr** can be a quoted literal string, e.g. `("Text")` for $Text. Or it can be an unquoted variable/loopVariable, e.g. `(vString)` for a variable 'vString'.

String.captureTo(matchStr[, targetAttributeStr])

Operator Type: Function [other Function type actions]
Operator Scope of Action: Item [operators of similar scope]
Operator Purpose: Stream parsing [other Stream parsing operators]
Operator First Added: Baseline
Operator Last Altered: As at baseline
Operator Uses Regular Expressions: [More on regular expressions in Tinderbox]
Operator Has Optional Arguments: [More on optional operator arguments]

String.captureTo(matchStr, targetAttributeStr)
String.captureTo(matchStr)
Stores the source text string up to, but not including, the designated literal **matchStr** into the specified **targetAttributeStr**, a quoted name of an attribute, and returns the string that follows **matchStr**. Note that **matchStr** is *not* a regular expression and can only be a literal string.
If the **targetAttributeStr** is omitted the **String** is advanced to the end so processing ceases.

The attribute given for **targetAttributeStr** can be a quoted literal string, e.g. `("Text")` for $Text. Or it can be an unquoted variable/loopVariable, e.g. `(vString)` for a variable 'vString'.

String.captureToken([targetAttributeStr])

Operator Type: Function [other Function type actions]
Operator Scope of Action: Item [operators of similar scope]
Operator Purpose: Stream parsing [other Stream parsing operators]
Operator First Added: Baseline
Operator Last Altered: As at baseline
Operator Has Optional Arguments: [More on optional operator arguments]

Syntax note: *Operators without any defined mandatory arguments may omit their empty closing parentheses*

String.captureToken(targetAttributeStr)
String.captureToken()
String.captureToken

Captures a sequence of non-whitespace characters ('tokens'), the matched token string is passed to the specified **targetAttributeStr**, a quoted name of an attribute. The **String** stream is not advanced and any chained parsing continues fror the same point. Punctuation (notably @) is part of a token but *not* part of a word.

If the **targetAttributeStr** is omitted the **String** is advanced to the end so processing ceases.

The attribute given for **targetAttributeStr** can be a quoted literal string, e.g. `("Text")` for $Text. Or it can be an unquoted variable/loopVariable, e.g. `(vString)` for a variable 'vString'.

String.captureWord([targetAttributeStr])

Operator Type: Function [other Function type actions]
Operator Scope of Action: Item [operators of similar scope]
Operator Purpose: Stream parsing [other Stream parsing operators]
Operator First Added: Baseline
Operator Last Altered: As at baseline
Operator Has Optional Arguments: [More on optional operator arguments]

Syntax note: *Operators without any defined mandatory arguments may omit their empty closing parentheses*

String.captureWord(targetAttributeStr)
String.captureWord()
String.captureWord

Finds and stores the next word in the string in the specified **targetAttributeStr**, a quoted name of an attribute, and returns the string that follows this word. Fails if the string is empty or entirely white space.

What constitutes a word

If the **targetAttributeStr** is omitted the **String** is advanced to the end so processing ceases.

The attribute given for **targetAttributeStr** can be a quoted literal string, e.g. `("Text")` for $Text. Or it can be an unquoted variable/loopVariable, e.g. `(vString)` for a variable 'vString'.

String.captureXML()

Operator Type: Function [other Function type actions]
Operator Scope of Action: Item [operators of similar scope]
Operator Purpose: Stream parsing [other Stream parsing operators]
Operator First Added: Baseline
Operator Last Altered: As at baseline

Syntax note: *Operators without any defined mandatory arguments may omit their empty closing parentheses*

String.captureXML()
String.captureXML

Attempts to parse the string as fully/partially XML and fails if unsuccessful. The parsed XML is saved as the current XML (stream) object.

Essentially, this re-scopes the current stream so that it contains only the contents of the first section of XML detected within the original stream. The focus of the stream parsing is set to the beginning of the extracted XML code.

Only one XML object may be current at any time. If the source stream contains multiple discrete XML code sections, only the first is detected/used.

String.contains(regexStr)

Operator Type: Function [other Function type actions]
Operator Scope of Action: Item [operators of similar scope]
Operator Purpose: Query Boolean [other Query Boolean operators]
Operator First Added: Baseline
Operator Last Altered: As at baseline
Operator Uses Regular Expressions: [More on regular expressions in Tinderbox]

String.contains(regexStr)

This operator tests whether regular expression pattern **regexStr** matches the referenced string attribute's value in whole or part. Matches are always *case-sensitive*, unlike String.icontains(). The match gives a coerced boolean result: if **regexStr** is matched (literally or via regex) the function returns the match offset+1 in the source string, where offset is the distance from the start of the string to the start of the matched regex. No match, coerces to `false`. The +1 works around the fact the match is zero-based, so a match at position 0 (the start of the string), returns as 1 (0+1) and a value of 1 or more coerces to a boolean `true`. The boolean allows queries' normal true/false to evaluate as normal.

If concerned over unwanted case-sensitivity, use String.icontains() which is *always* case-insensitive.

regexStr is one of:

- an unquoted action code expression, which includes just referencing a single attribute name e.g. `MyString`
- a quoted string; quoted strings may be either:
  - a literal string (i.e. actual text)
  - a regular expression.

Important: do not omit the enclosing quotes for literal strings or regex. If omitted, Tinderbox will try to evaluate the string as an expression. Doing this *may* result in the expected result but this is actually a false positive. *So, remember to enclose your regex or literals in quotes.*

For example:

```
$MyString.contains("regex")
```

is `true` if $MyString matches **regexStr**'s pattern. Other more complex usage:

```
$MyString.contains($MyMatchText)
$MyString.contains($MyString(agent))
$MyString(parent).contains("Tuesday")
"Any day like Saturday is good".contains($MyDay)
"Any day like Saturday is good".contains("Saturday")
```

Note that regex/literal strings are quoted whilst action expressions are not.

Getting the offset of the (first) regex match

If the regular expression **regexStr** is found the function returns the match offset+1, where offset is the distance from the start of the string to the start of the matched regex. If there is more than one match, the offset of the *first* match is returned. Formerly, .contains() returned `true` if the regex was found. The '+1' modifier ensures that a match at position zero return a number higher than zero which would otherwise coerce to `false`. Since 1+offset is always `true`, no chang are required in existing documents but the function also gives usable offset information. Thus, if $MyString is "abcdefgehEi":

```
$MyNumber = $MyString.contains("e");
```
returns `5`.

```
$MyNumber = $MyString.contains("E");
```
returns `10`.

```
$MyNumber = $MyString.contains("eh");
```
returns `8`.

Testing "does not contain"

Use a ! prefix to the query argument:

```
!$MyString.contains("Tuesday")
```

Use of parentheses around the negated query term, can assist Tinderbox's parsing:

```
(!$MyString.contains("Tuesday"))
```

Using back-references

In an agent query or if() conditions the function can return back-references to matches of (sub-)strings.

String**.contains()** clears the list of back references from previous processes, so $0 and $1 correspond to its own results, not those from prior expressions.

Dealing with inline quote characters

Because **regex** is parsed for regular expressions, it may be possible to use the '\xNN' form described here to work around the lack of escaping from single double quotes within strings.

Legacy format

This operator is the replacement of the older form of AttributeName(regex) which is deprecated and should not be used in new work (legacy support may fall away). Apart from anything else, the current syntax should remove the confusion over whether/when to use the $ prefix with attribute names in queries.

String.containsAnyOf(regexList)

Operator Type: Function [other Function type actions]
Operator Scope of Action: List [operators of similar scope]
Operator Purpose: Query Boolean [other Query Boolean operators]
Operator First Added: Baseline
Operator Last Altered: As at baseline
Operator Uses Regular Expressions: [More on regular expressions in Tinderbox]

String.containsAnyOf(regexList)

The operator **.containsAnyOf(regexList)** is `true` if any of the words in a set of words (i.e. list) is contained in the chained-to target string. As shown below, the list may have only one entry. The test is *case-sensitive* (previously it was *case-insensitive*). For example:

```
$MyBoolean = $Text.containsAnyOf("emulate");
```

Will be `true` if the tested note's $Text contains the word "emulate". A more applied example:

```
$MyBoolean = $Text.containsAnyOf(wordsRelatedTo("emulate"));
```

Will be `true` if the tested note's $Text contains the word "aspire".

regexList implies using a list of values (ideally with no dupes). This can be a literal list of 1 or more values—as in the example above, or an attribute reference holding a list of values, for instance:

```
$MyBoolean = $Text.containsAnyOf($MySet);
```

It is important to note that .containsAnyOf() is always a *case-insensitive* test. Thus in the first example above, it will match "emulate" but not "Emulate" or any other case variant of the word.

Although the examples above use whole words the list in **regexList** is actually processed assuming they are regular expressions (which may of course be literal strings). Thus in the first example above, it will match both "emulate" and "emulated" but not "emulating". The test value "emulate\b", expecting a word break after the final 'e' would this match "emulate" but *not* "emulated".

For a case-insensitive version of this operator see String.icontainsAnyOf().

If wishing to test a List or Set, chain to List/Set.asString():

```
$MyBoolean = $MyList.asString.containsAnyOf(wordsRelatedTo("emulate"));
```

String.countOccurrencesOf(literalStr)

Operator Type: Function [other Function type actions]
Operator Scope of Action: Item [operators of similar scope]
Operator Purpose: Data manipulation [other Data manipulation operators]
Operator First Added: Baseline
Operator Last Altered: As at baseline

String.countOccurrencesOf(literalStr)

This function returns the Number of times that the *literal* argument '**literalStr**' appears in the String. If $MyString contains the word "aardvark", then:

`$MyNumber = $MyString.countOccurrencesOf("a");` returns 3
`$MyNumber = $MyString.countOccurrencesOf("aa");` returns 1
`$MyNumber = $MyString.countOccurrencesOf("r");` returns 2

literalStr is literal and must *not* be a regular expression. If the latter is needed use String.contains() or String.icontains().

String.deleteCharacters(characterSet)

Operator Type: Function [other Function type actions]
Operator Scope of Action: Item [operators of similar scope]
Operator Purpose: Data manipulation [other Data manipulation operators]
Operator First Added: Baseline
Operator Last Altered: As at baseline

String.deleteCharacters(characterSet)

Returns a copy of the string from which all instances of every discrete character in the quoted literal string-based set **characterSet** have been removed. For example:, if $MyString is "1Hello1 2world3";

```
$MyStringA = $MyString.deleteCharacters("1234567890");
```

would remove any digits from the target string, thus $MyStringA will be "Hello world", having removed two instances of '1', and a single instance of '2' and of '3'.

But were $MyString "Hello world: 45";

```
$MyStringA = $MyString.deleteCharacters("1234567890");
```

the result would be "Hello world: ". Importantly, note the trailing space as only the digits (i.e. characters in **characterSet**) have been removed.

String.eachLine(loopVar[:condition]){actions}

Operator Type: Function [other Function type actions]
Operator Scope of Action: List [operators of similar scope]
Operator Purpose: Stream parsing [other Stream parsing operators]
Operator First Added: Baseline
Operator Last Altered: 9.6.0
Operator Has Optional Arguments: [More on optional operator arguments]
Operator Has Conditional Arguments: [More on conditional operator arguments]
Operator Uses Loop Variable Argument: [More on loop variable arguments]

String.eachLine(loopVar[:condition]){ action(s) }

The .eachLine() operator, iterates through each line of a steam, where a line is one or more characters ending in a carriage return, line feed, unicode paragraph separator, or the end of the string (so in $Text, it means discrete paragraphs).

Each line, in turn, is bound to a temporary variable **loopVar**, and the action is then performed. The name of the variable is set *by the user* when writing the code, i.e. it is whatever string is entered where **loopVar** is shown above. Something like 'aLine' might be a more useful variable name.

If the optional **condition** is specified, only lines that satisfy the condition are processed using the action.

For example:

```
$MyNumber=0;
$Text.eachLine(aLine){
   $MyNumber=$MyNumber+1;
};
```

will set $MyNumber to the number of lines (paragraphs!) in the $Text of this note.

```
$MyNumber=0;
$Text.eachLine(aLine:aLine.contains("@")){
   $MyNumber=$MyNumber+1;
};
```

will set $MyNumber to the number of lines in the $Text that contain the "@" symbol.

When parsing text paragraphs, this operator can substitute for the older method of chaining .paragraphList.each(){}.

From v9.6.0, **.eachLine(){}** no longer skips whitespace at the start of the line.

String.empty()

Operator Type: Property [other Property type actions]
Operator Scope of Action: Item [operators of similar scope]
Operator Purpose: Non-query Boolean [other Non-query Boolean operators]
Operator First Added: Baseline
Operator Last Altered: As at baseline

Syntax note: *Operators without any defined mandatory arguments may omit their empty closing parentheses*

String.empty()

String.empty

This returns a Boolean depending on whether the string is empty. If empty, the return value is `true`, if the attribute has content then *false* is returned.

`$MyString = ""; $MyBoolean = $MyString.empty;` $MyBoolean is set to `true`.

`$MyString = "hello"; $MyBoolean = $MyString.empty;` $MyBoolean is set to `true`.

This operator can also be used on other attribute data types that are string-like, URL, File, Action, Color, etc., and which have no value at all by default. However, for a Number or Date, the default values of `0` or `never`, coerce to a literal str of "`0`" or "`never`" so the `.empty()` test does not work the same way. For the latter types, use the short Boolean test form, !AttributeName.

String.endsWith(matchStr)

Operator Type: Function [other Function type actions]
Operator Scope of Action: Item [operators of similar scope]
Operator Purpose: Data manipulation [other Data manipulation operators]
Operator First Added: Baseline
Operator Last Altered: As at baseline

String.endsWith(matchStr)

String.endsWith() returns Boolean `true` if the chained string ends with a specific substring.

For example, if $MyString is "I do not like green eggs and ham":

`$MyString.endsWith("ham")` returns true

`$MyString.endsWith("eggs")` returns false

Also, with string literals:

`("There are gentlemen now abed.").endsWith("abed")` returns `true`

`("There are gentlemen now abed.").endsWith("night")` returns `false`

This operator searches for literal strings, not regular expressions. Matches are case sensitive:

`("There are gentlemen now abed.").endsWith("Abed")` returns `false`

See also String.beginsWith(). If you need to search a String for a regular expression, use String.contains().

String.expect(matchStr)

Operator Type: Function [other Function type actions]
Operator Scope of Action: Item [operators of similar scope]
Operator Purpose: Stream parsing [other Stream parsing operators]
Operator First Added: Baseline
Operator Last Altered: As at baseline

String.expect(matchStr)

The operator tests if the String begins with the **matchStr** string with the following results:

- success: operator returns the rest of string that follows **matchStr**.
- failure: nothing is returned

String.expectNumber()

Operator Type: Function [other Function type actions]
Operator Scope of Action: Item [operators of similar scope]
Operator Purpose: Stream parsing [other Stream parsing operators]
Operator First Added: Baseline
Operator Last Altered: As at baseline

Syntax note: *Operators without any defined mandatory arguments may omit their empty closing parentheses*

String.expectNumber()

String.expectNumber

If the string begins with one or more contiguous character(s) that could be interpreted as a number, the **String** stream cursor skips to the end of that number sequence and parsing continues.

"1234 items" is passed so the remaining stream is "items"

"1,234 items" is passed so the remaining stream is ",234 items"

Note how the presence of a formatting comma breaks an otherwise continuous number sequence.

String.expectWhitespace()

Operator Type: Function [other Function type actions]
Operator Scope of Action: Item [operators of similar scope]
Operator Purpose: Stream parsing [other Stream parsing operators]
Operator First Added: Baseline
Operator Last Altered: As at baseline

Syntax note: *Operators without any defined mandatory arguments may omit their empty closing parentheses*

String.expectWhitespace()

String.expectWhitespace

If the string begins the whitespace, skips the whitespace and succeeds. Otherwise, fails.

String.expectWord()

Operator Type: Function [other Function type actions]
Operator Scope of Action: Item [operators of similar scope]
Operator Purpose: Stream parsing [other Stream parsing operators]
Operator First Added: Baseline
Operator Last Altered: As at baseline

Syntax note: *Operators without any defined mandatory arguments may omit their empty closing parentheses*

String.expectWord()

String.expectWord

If the string begins with one or more contiguous character(s) that could be interpreted as a word the **String** stream cursor skips to the end of that sequence and parsing continues.

String.extract(regexStr[, caseInsensitiveBln])

Operator Type: Function [other Function type actions]
Operator Scope of Action: Item [operators of similar scope]
Operator Purpose: Data manipulation [other Data manipulation operators]
Operator First Added: Baseline
Operator Last Altered: 9.5.0
Operator Uses Regular Expressions: [More on regular expressions in Tinderbox]

String.extract(regexStr)

The operator returns the *first* matched subexpression of a quote-enclosed regular expression **regexStr** in the source String. If the regular expression has no subexpressions, the entire match is returned up to the end of the current line/paragraph, i.e. the first line break character encountered.

For example, take a String which is the $Text "We are very tired, Harrison! #things #Memes #August":

```
$MyString = $Text.extract("[A-Z][A-Za-z+]");
```

this returns "We" as that is the first match (although there are others). This sort of use is the presumed most likely used of this operator.

If we amend the regex

```
$MyString = $Text.extract("#([A-Z][A-Za-z+])");
```

it now returns "Memes" (the first hash tag starts with a lowercase letter does not match).

If multiple matches are expected/wanted, see String.extractAll().

String.extract(regex[, caseInsensitiveBln])

From v9.5.0, **String.extract()** now accepts an optional boolean second argument **caseInsensitiveBln**. If that argument is `true`, the **regex** argument's regular expression search is case-insensitive. The default value is `false` respecting the pre-existing behaviour.

String.extractAll(regexStr[, caseInsensitiveBln])

Operator Type: Function [other Function type actions]
Operator Scope of Action: Item [operators of similar scope]
Operator Purpose: Data manipulation [other Data manipulation operators]
Operator First Added: Baseline
Operator Last Altered: 9.5.0
Operator Uses Regular Expressions: [More on regular expressions in Tinderbox]

String.extractAll(regexStr)

The operator returns a (semi-colon delimited) list of *all* matches for a quote-enclosed **regexStr** found in the source String.

For example:

```
$MyList = $Text.extractAll("#[A-Za-z]+");
```

would return a list of *all* discrete tag instances in the source String such as: "#Tinderbox;#Stuff;#Thing;#tinderbox;#Tinderbox;#Cars".

To get a de-duped list, without the hashes we can chain other operators:

```
$MyList = $Text.extractAll("#[A-Za-z]+").unique.replace("#","");
```

That refines the returned list: "Cars;Stuff;Thing;Tinderbox;tinderbox". Note the extra operators also sort the list too.

String.extractAll(regexStr[, caseInsensitiveBln])

From v9.5.0, **String.extractAll()** now accepts an optional boolean second argument **caseInsensitiveBln**. If that argument is `true`, the **regex** argument's regular expression search is case-insensitive. The default value is `false` respecting th pre-existing behaviour.

String.failed()

Operator Type: Function [other Function type actions]
Operator Scope of Action: Item [operators of similar scope]
Operator Purpose: Stream parsing [other Stream parsing operators]
Operator First Added: Baseline
Operator Last Altered: As at baseline

Syntax note: *Operators without any defined mandatory arguments may omit their empty closing parentheses*

String.failed()

String.failed

.failed tests for a failed Stream processing action String.try{}, including an explicit fail() call. It returns `true` if the current operation has failed, and *false* otherwise.

When processing a string, it may be that the string is not what was expected. Thus the **.failed** operator raises a flag to say this process did not work.

A **.failed** flag is reset to `false` when the current try{} expires, or when the current action is complete.

For example, to test if a 'try' sequence has failed:

```
if ( $MyString.try{ … }.failed() ) {
 $Color="bright red";
}
```

String.find(matchStr)

Operator Type: Function [other Function type actions]
Operator Scope of Action: Item [operators of similar scope]
Operator Purpose: Data manipulation [other Data manipulation operators]
Operator First Added: Baseline
Operator Last Altered: As at baseline

String.find(matchStr)

This operator returns a Number with the location of the first occurrence of a literal (not regex) substring **matchStr** within the source string. Thus, the operator searches for literal strings, not regular expressions. Matches are case sensitive. Th offset is zero-based (position 1 is zero). If the string is not found, String.find(matchStr) returns -1.

For example, if $MyString is "I do not like green eggs and ham":

`$MyNumber = $MyString.find("not");` returns 5
`$MyNumber = $MyString.find("blue");` returns -1 (not found in match string)
`$MyNumber = $MyString.find("Not");` returns -1 (not found, due to case-sensitive matching)

String.following(matchStr)

Operator Type:	Function [other Function type actions]
Operator Scope of Action:	Item [operators of similar scope]
Operator Purpose:	Data manipulation [other Data manipulation operators]
Operator First Added:	Baseline
Operator Last Altered:	As at baseline

String.following(matchStr)

The .**following( )** operator extracts information from one or more paragraphs of a string using a *literal quoted string* as the match (i.e. no regex support). If **matchStr** appears in a line, then whatever follows *after* the **matchStr** is the result. If **matchStr** appears in more than one line, the return value is a list of results.

For example, if the text of a note is:

```
From: Mark
To: Clotilde
Subject: Mignardise
```

Then:

```
$MyString = $Text.following("To:");
```

would return "Clotilde".

```
$MyString = $Text.following(":");
```

would return "Mark;Clotilde:Mignardise".

In addition, the .following() operator binds any text that precedes the **matchStr** string to $0. If the **matchStr** appears several times, the first prefix is bound to $0, the second to $1, and so forth.

The .following() operator's argument is interpreted as a sequence of characters, *not* a regular expression. If the flexibility of regular expressions is needed, use .find() instead.

The .following() operator is useful for extracting formal and semi-formal data from free text and simple interchange formats like email and RIS.

String.highlights([aColor])

Operator Type:	Property [other Property type actions]
Operator Scope of Action:	Item [operators of similar scope]
Operator Purpose:	Data manipulation [other Data manipulation operators]
Operator First Added:	Baseline
Operator Last Altered:	As at baseline
Operator Has Optional Arguments:	[More on optional operator arguments]

Syntax note: *Operators without any defined mandatory arguments may omit their empty closing parentheses*

String.highlights(aColor)

String.highlights

The expression returns a list of strings that have been highlighted in the $Text using Format ▸ Highlight.

```
$MyList = $Text.highlights;
```

String.highlights(aColor)

The result may be limited to strings highlighted in a specific colour by supplying **aColor** as an argument. **aColor** may be any of the currently supported highlight colours: "red", "green", "blue", "yellow", or "magenta". Thus:

```
$MyList = $Text.highlights("red");
```

would return only those highlighted $Text portions using a red highlight colour.

String.icontains(regexStr)

Operator Type:	Function [other Function type actions]
Operator Scope of Action:	Item [operators of similar scope]
Operator Purpose:	Query Boolean [other Query Boolean operators]
Operator First Added:	Baseline
Operator Last Altered:	9.0.0
Operator Uses Regular Expressions:	[More on regular expressions in Tinderbox]

String.icontains(regexStr)

This operator tests whether regular expression pattern **regexStr** matches the referenced string attribute's value in whole or part. Matches are always case- *insensitive*, unlike String.contains(). The match gives a coerced boolean result: if **regexStr** is matched (literally or via regex) the function returns the match offset+1 in the source string, where offset is the distance from the start of the string to the start of the matched regex. No match, coerces to `false`. The +1 works around the fact the match is zero-based, so a match at position 0 (the start of the string), returns as 1 (0+1) and a value of 1 or more coerces to a boolean `true`. The boolean allows queries' normal true/false to evaluate as normal.

If needing case-sensitivity, use String.contains() which is *always* case-sensitive

regexStr is one of:

- an unquoted action code expression, which includes just referencing a single attribute name e.g. `MyString`
- a quoted string; quoted strings may be either:
  - a literal string (i.e. actual text)
  - a regular expression.

Important: do not omit the enclosing quotes for literal strings or regex. If omitted, Tinderbox will try to evaluate the string as an expression. Doing this *may* result in the expected result but this is actually a false positive. *So, remember to enclose your regex or literals in quotes.*

For example:

```
$MyString.icontains("regex")
```

is `true` if $MyString matches **regexStr**'s pattern. Other more complex usage:

```
$MyString.icontains($MyMatchText)
$MyString.icontains($MyString(agent))
$MyString(parent).icontains("Tuesday")
"Any day like Saturday is good".icontains($MyDay)
"Any day like Saturday is good".icontains("Saturday")
```

Note that regex/literal strings are quoted whilst action expressions are not.

Getting the offset of the (first) regex match

If the regular expression **regexStr** is found the function returns the match offset+1, where offset is the distance from the start of the string to the start of the matched regex. If there is more than one match, the offset of the *first* match is returned. Formerly, .contains() returned `true` if the regex was found. The '+1' modifier ensures that a match at position zero return a number higher than zero which would otherwise coerce to `false`. Since 1+offset is always `true`, no changes are required in existing documents but the function also gives usable offset information. Thus, if $MyString is "abcdefgehEi":

```
$MyNumber = $MyString.icontains("e");
```
returns `5`.

```
$MyNumber = $MyString.icontains("E");
```
returns `5`.

```
$MyNumber = $MyString.icontains("eh");
```
returns `8`.

Testing "does not contain"

Use a ! prefix to the query argument:

```
!$MyString.icontains("Tuesday")
```

Use of parentheses around the negated query term, can assist Tinderbox's parsing:

```
(!$MyString.icontains("Tuesday"))
```

Using back-references

In an agent query or if() conditions the function can return back-references to matches of (sub-)strings.

String.**icontains()** clears the list of back references from previous processes, so $0 and $1 correspond to its own results, not those from prior expressions.

Dealing with inline quote characters

Because **regex** is parsed for regular expressions, it may be possible to use the '\xNN' form described here to work around the lack of escaping from single double quotes within strings.

Legacy format

This operator is the replacement of the older form of AttributeName(regex) which is deprecated and should not be used in new work (legacy support may fall away). Apart from anything else, the current syntax should remove the confusion over whether/when to use the $ prefix with attribute names in queries.

String.icontainsAnyOf(regexList)

Operator Type: Function [other Function type actions]
Operator Scope of Action: List [operators of similar scope]
Operator Purpose: Query Boolean [other Query Boolean operators]
Operator First Added: Baseline
Operator Last Altered: As at baseline
Operator Uses Regular Expressions: [More on regular expressions in Tinderbox]

String.icontainsAnyOf(regexList")

The operator **.icontainsAnyOf("regexList)** is `true` if any of the words in a set of words (i.e. list) is contained in the chained-to target string. As shown below, the list may have only one entry. The test is *case-insensitive*. For example:

```
$MyBoolean = $Text.icontainsAnyOf("emulate");
```

Will be *true* if the tested note's $Text contains the word "emulate". A more applied example:

```
$MyBoolean = $Text.icontainsAnyOf(wordsRelatedTo("emulate"));
```

Will be *true* if the tested note's $Text contains the word "aspire".

regexList implies using a list of values (ideally with no dupes). This can be a literal list of 1 or more values—as in the example above, or an attribute reference holding a list of values, for instance:

```
$MyBoolean = $Text.icontainsAnyOf($MySet);
```

It is important to note that .containsAnyOf() is always a *case-insensitive* test. Thus in the first example above, it will match both "emulate" and "Emulate" or any other case variant of the word.

Use of regular expressions

Although the examples above use whole words the list in **regexList** is actually processed assuming they are regular expressions (which may of course be literal strings). Thus in the first example above, it will match both "emulate" and "emulated" but not "emulating". The test value "emulate\b", expecting a word break after the final 'e' would this match "emulate" but *not* "emulated".

For a case-sensitive version of this operator see String.containsAnyOf().

If wishing to test a List or Set, chain to List/Set.asString():

```
$MyBoolean = $MyList.asString.icontainsAnyOf(wordsRelatedTo("emulate"));
```

String.json()

Operator Type: Function [other Function type actions]
Operator Scope of Action: Item [operators of similar scope]
Operator Purpose: Formatting [other Formatting operators]
Operator First Added: Baseline
Operator Last Altered: As at baseline

Syntax note: *Operators without any defined mandatory arguments may omit their empty closing parentheses*

String.json()

String.json

NOTE: Deprecated in v9.1.0+ - use String.jsonEncode() instead

This operator returns a JSON-encoded UTF-8 version of a string attribute's value. The apostrophe (straight single quote), straight double quote, solidus (forward slash) and backslash characters are all escaped by a preceding backslash character.

See also jsonEncode().

String.jsonEncode()

Operator Type: Function [other Function type actions]
Operator Scope of Action: Item [operators of similar scope]
Operator Purpose: Formatting [other Formatting operators]
Operator First Added: Baseline
Operator Last Altered: As at baseline

Syntax note: *Operators without any defined mandatory arguments may omit their empty closing parentheses*

String.jsonEncode()

String.jsonEncode

The **.jsonEncode()** operator returns a JSON-encoded UTF-8 version of the chained source **String**'s value. Forbidden characters such as the apostrophe (straight single quote), straight double quote, solidus (forward slash) and backslash characters are all escaped by a preceding backslash character.

The general expectation is String will be a quoted literal string or a String-type attribute or variable, e.g. $Text:

```
var:string vEncoded = $Text.jsonEncode()
^value("Apostrophe's are often wrongly used".jsonEncode())^
```

This operator was formerly know as .json, and is effectively a replacement for that operator. The naming also better reflects the non dot-operator jsonEncode().

Using the latter, the same outcome can be encoded as:

```
var:string vEncoded = jsonEncode($Text)
^value(jsonEncode("Apostrophe's are often wrongly used"))^
```

For attributes, variables and very long literal string, the chained dot-operator seems more helpful when coding, whereas the older form may be more intuitive with short literal strings (as shown in the last example above.)

String.lowercase()

Operator Type: Function [other Function type actions]
Operator Scope of Action: Item [operators of similar scope]
Operator Purpose: Formatting [other Formatting operators]
Operator First Added: Baseline
Operator Last Altered: As at baseline

Syntax note: *Operators without any defined mandatory arguments may omit their empty closing parentheses*

String.lowercase()

String.lowercase

Returns the referenced string, transforming all uppercase letters to lowercase.

The function can be chained to both string data and to string literals:

```
$MyString.lowercase()
"My NEW Title".lowercase()
```

The latter results in "my new title".

The trailing parentheses may be omitted:

```
$MyString.lowercase
```

Functionally equivalent to lowercase().

If $MyString is "Hello World":

```
$MyString = $MyString.lowercase;
```

$MyString is set to "hello world".

The .lowercase() method may also be used on Lists or Sets. Consider [Ant;BEE;Cow] stored in $MyList:

```
$MyList = $MyList.lowercase;
```

… giving [ant;bee;cow].

String.next()

Operator Type: Function [other Function type actions]
Operator Scope of Action: Item [operators of similar scope]
Operator Purpose: Data manipulation [other Data manipulation operators]
Operator First Added: Baseline
Operator Last Altered: As at baseline

Syntax note: *Operators without any defined mandatory arguments may omit their empty closing parentheses*

String.next()

String.next

The operator **.next** generates convenient note names and other strings in a sequence. For example:

```
$MyString = "footnote".next ; → "footnote 1"
$MyString = "footnote 1".next; → "footnote 2"
```

Specifically, **.next** searches a string for its last run of digits. If no digits are found, **.next** returns the string followed by " 1". Otherwise, the number is increments and placed in the same position in the string.

```
$MyString = "Agent 007 (active)".next; → "Agent 008 (active)"
```

String.nounList()

Operator Type: Property [other Property type actions]
Operator Scope of Action: Item [operators of similar scope]
Operator Purpose: Data manipulation [other Data manipulation operators]
Operator First Added: Baseline
Operator Last Altered: As at baseline

Syntax note: *Operators without any defined mandatory arguments may omit their empty closing parentheses*

String.nounList()

String.nounList

This returns a list of each noun in the supplied string, excepting those recognised as pronouns and proper names. Note that the output list is all lowercase, regardless of source case. This operator requires running on macOS 10.14 and later.

```
$MyList = "I am the very model of a modern Major-General.".nounList;
```

then MyList holds "model;major;general".

String.paragraph(paraNum)

Operator Type: Function [other Function type actions]
Operator Scope of Action: Item [operators of similar scope]
Operator Purpose: Data manipulation [other Data manipulation operators]
Operator First Added: Baseline
Operator Last Altered: As at baseline

String.paragraph(paraNum)

returns the text of the **paraNum** paragraph in the source string. **paraNum** is zero-based, i.e. the first paragraph is .paragraph(0); empty paragraphs are ignored. If the string does not contain **paraNum** paragraphs, the result is an empty strin

If **paraNum** is negative, Tinderbox counts from the last paragraph. Thus, `$Text.paragraph(-1)` is the last paragraph in the text.

For example:

```
$MyString = $Text.paragraph(2);
```

sets $MyString the the contents of the *third* paragraph of $Text—recall at N is zero-based, so counting 0/1/2 means 2 is #3 in the sequence.

String.paragraphCount()

Operator Type: Property [other Property type actions]
Operator Scope of Action: Item [operators of similar scope]
Operator Purpose: Data manipulation [other Data manipulation operators]
Operator First Added: Baseline
Operator Last Altered: As at baseline

Syntax note: *Operators without any defined mandatory arguments may omit their empty closing parentheses*

String.paragraphCount()

String.paragraphCount

returns the Number of paragraphs in a string; empty paragraphs are ignored. Thus, `$Text.paragraphCount` is the number of paragraphs in the text of the currently selected note.

```
$MyNumber = $Text.paragraphCount;
```

sets $MyNumber to the number of discrete paragraphs within $Text.

String.paragraphList()

Operator Type: Property [other Property type actions]
Operator Scope of Action: Item [operators of similar scope]
Operator Purpose: Data manipulation [other Data manipulation operators]
Operator First Added: Baseline
Operator Last Altered: As at baseline

Syntax note: *Operators without any defined mandatory arguments may omit their empty closing parentheses*

String.paragraphList()

String.paragraphList

The **.paragraphList** operator returns a list of paragraphs contained in a string. This operator requires running on macOS 10.14 and later. For example, to get a list of the discrete paragraphs in $Text:

```
$MyList = $Text.paragraphList;
```

To set a string to the third paragraph of $Text (recall the .at(N) operator is zero-based):

```
$MyString = $Text.paragraphList.at(2);
```

If wanting to iterate and test paragraphs, rather than chain $Text.paragraphList.each(){}, use the newer stream parsing method $Text. eachLine(){}. In both instance a line—or paragraph—is a substring—delimited by one of more successive lir breaks.

Source paragraphs containing semicolons

As **.paragraphList** returns a List, if any source paragraph containing a semicolon it will create more than one item in the returned list. In such circumstances, in may be better to use .eachLine() instead. Or, if encountered, a technique is to fi sanitise the source by changing semicolons to another string and then back:

```
var:string vSource = $Text.replace(";","@@@");
$MyList = vSource.paragraphList;
$MyString = $MyList.at(4)replace("@@@",";");
```

String.paragraphs(parasNum)

Operator Type:	Function [other Function type actions]
Operator Scope of Action:	Item [operators of similar scope]
Operator Purpose:	Data manipulation [other Data manipulation operators]
Operator First Added:	Baseline
Operator Last Altered:	As at baseline

String.paragraphs(parasNum)

This function extracts the first **parasNum** paragraphs of the referenced string or String-type attribute. Examples:

```
$MyString = "Paragraph 1\nParagraph 2".paragraphs(1);
```

In the example the "\n" simulates a line break. The code would set $MyString to the string "Paragraph 1". A more real example:

```
$Text = $Text("Some note").paragraphs(2);
```

In the second example the current note's $Text would be set to the first 2 paragraphs of note "Some note".

To get a single given paragraph of a multi line/paragraph string, such as $Text, see String.split() (beware of semicolons in the source string being misinterpreted as list item delimiters, as discussed in that article).

This function respects existing rich text styling.

To get all paragraphs as a list, e.g. in order to iterate against them using list .each(), then use string.split(), e.g.:

```
$MyList = $Text.split("\n+");
```

Or from a different note, e.g. 'another note':

```
$MyList = $Text("another note").split("\n+");
```

The `\n+` implies to split on and remove sub-strings of one or more consecutive line breaks, thus avoiding the creation of unwanted creating blank list items in the output where there are several line breaks between paragraphs.

Working with styled text

This operator is capable of worthing with StyledString operators: StyledString.bold, StyledString.fontSize(), StyledString.italic and StyledString.strike.

String.replace(regexMatchStr, replacementStr)

Operator Type:	Function [other Function type actions]
Operator Scope of Action:	Item [operators of similar scope]
Operator Purpose:	Data manipulation [other Data manipulation operators]
Operator First Added:	Baseline
Operator Last Altered:	As at baseline
Operator Uses Regular Expressions:	[More on regular expressions in Tinderbox]

String/List.replace(regexMatchStr, replacementStr)

This operator allows simple text transformations without use of runCommand as was hitherto required.

regexMatchStr and **replacement** are one of:

- an action code expression (which includes just referencing a single attribute name')
- a quoted string, which may be either:
  - a literal string (i.e. actual text)
  - a regular expression (**regexMatchStr** only)

$MyString.replace(regexMatchStr, replacementStr)

In its simplest form, the operator creates a new string in which each occurrence of regex is replaced by the string replacement, i.e. global replacement. The source string is not changed by replace; if you wish to change the string itself, write back over the source attribute:

```
$MyString = $MyString.replace("Hello","Goodbye");
```

transforms a $MyString value of "Hello World" to "Goodbye World".

Where **regexMatchStr** is a regular expression, and may contain wildcard characters such as "." (which matches any character) or "*" (which matches 0 or more occurrences of the preceding character). Thus ".*" matches zero or more instances of any character.

Where parentheses in **regexMatchStr** create multiple back-references, **replacementStr** strings can include $1, $2, etc., to insert the relevant back-reference matched string ($1 through $9 allowed, and with $0 being the entire match).

Examples:

```
$MyString.replace("Spenser","Spencer");
```

changes all instance of "Spenser" to "Spencer".

```
$MyString.replace("(a|e|i|o|u)","");
```

deletes all vowels

```
$MyString = "I do not like green eggs".replace("(green) (eggs)","$2 $1");
```

returns "I do not like eggs green". Or:

```
$Text=$Text.replace("From: (.+)@(.*)","—$1—\n$2");
```

Will replace

```
From: mark@example.com
```

with

```
—mark—
example.com
```

Note that if the source text is part of a larger text, e.g. a whole email's plain text, consider using stream parsing methods.

A replace action does not alter the original source

Using .replace() does not affect the source string unless the replacement output is used to overwrite the original source value. Thus if $MyString holds "Hello World" then:

```
$MyStringA = $MyString.replace(" World");
```

$MyString remains "Hello Word" and $MyStringA has value "Hello". The source is unchanged. But, if we set the source to the output

```
$MyString = $MyString.replace(" World");
```

Now $MyString becomes "Hello" and the original value is lost (overwritten by the new one). This distinction is one to bear in mind when using .replace() with $Text.

Using .replace() with $Text and formatting operators

When applied to $Text, .replace() allows style operators to be applied to the replacement argument. For example,

```
replace("^ from: .*",$0.bold)
```

will embolden all lines beginning with "From:".

If using style operators, do not place operator-modified back-reference within quotes. To re-use the example from above, consider $text containing "I do not like green eggs and ham":

```
$Text = "$Text.replace("(green) (eggs)",$2.bold+" "+ $1.strike);
```

gives "I do not like **eggs** ~~green~~ and ham".

Multiple, but differing replacements

Although multiple matches can be replaced with the same string, to replace multiple matches with different strings requires chained .replace() calls. Consider formatting a large number to Continental style. This means inserting spaces as the group delimiter and a comma for the decimal delimiter. Assume $MyNumber's value is 1234567.89:

```
$MyString = $MyNumber.replace("(\d)(?=(?:\d{3})+([^\d]))","$1 ").replace("\.",",");
```

Now, 1234567.89 becomes "1 234 567,89".

Some comma-delimited formats use straight double quotes for all/some values and demand that if this character appears in a value that is it escaped by doubling the character. If $Text is

```
He shouted "Hello!" at the top of his voice.
```

Then it could be escaped for CSV export like so:

```
…,"^value($Text.replace('"','""'))^",…
```

That exports:

```
…,"He shouted ""Hello!"" at the top of his voice.",…
```

Note that in this instance using a single straight quote (instead of the more normal straight double quote) to contain the find and replace regexs works just fine. Also, there is no need to escape typographic double quotes, i.e. 'smart' or 'curly' quotes in this context.

This function respects existing rich text styling.

Short form for deletions

If the **replacement** string is omitted, the one-argument form $MyString.replace(regex) returns a copy of $MyString in which every occurrence of the regex is removed.

Trimming leading/trailing whitespace

```
$MyString = $MyString.replace("^ +","").replace(" +$","");
```

The ' +' means *one or more space characters* . The first replace finds such a sub-string immediately following the start of the whole string ^), whilst the second does the same for a sub-string immediately before the end of the string ($). If

applied to multi-paragraph string, e.g. with line breaks such as in $Text, *every* paragraph is trimmed. Likely this is what is desired, but care is needed in more specialist situations. So, testing sample strings/texts is a good idea before changing actual data of value. If working with List or Sets a slightly different code is needed.

Trimming leading/trailing quotes

Because Tinderbox cannot escape quote characters (i.e. \" or \' do *not* escape the quote), use String.substr() to trim enclosing quotes on a string. Note that the latter, working on location in the string and not character type, cannot work on paragraphs within a string, such as in the example above.

Dealing with inline quote characters

Because **regex** is parsed for regular expressions, it may be possible to use the '\xNN' form of character encoding described here to work around the lack of escaping from single double quotes within strings.

Handling changes to $Text including link anchors

When using .replace is used on a note's $Text, the replaced text also updates the position of pre-existing text link anchor text.

Working with styled text

This operator is capable of worthing with StyledString operators: StyledString.bold, StyledString.fontSize(), StyledString.italic and StyledString.strike.

String.reverse()

Operator Type: Function [other Function type actions]
Operator Scope of Action: Item [operators of similar scope]
Operator Purpose: Data manipulation [other Data manipulation operators]
Operator First Added: Baseline
Operator Last Altered: As at baseline

Syntax note: *Operators without any defined mandatory arguments may omit their empty closing parentheses*

String.reverse()
String.reverse

This function reverses the order of the referenced string or string literal. Trailing parentheses are optional for this function.

```
$MyString = $MyString.reverse();
$MyString = $MyString.reverse;
$MyString = "man bites dog".reverse();
```

The latter gives "god setib nam" not "dog bites man". It is the order of *characters* that is reversed, not words within it.

String.sentence([sentenceNum])

Operator Type: Function [other Function type actions]
Operator Scope of Action: Item [operators of similar scope]
Operator Purpose: Data manipulation [other Data manipulation operators]
Operator First Added: Baseline
Operator Last Altered: As at baseline
Operator Has Optional Arguments: [More on optional operator arguments]

Syntax note: *Operators without any defined mandatory arguments may omit their empty closing parentheses*

String.sentence([sentenceNum])
String.sentence()
String.sentence

The dot-operator **.sentence(sentenceNum)** extracts the **sentenceNum**th sentence from the source String, which most likely will be $Text. For example, if $Text contains the passage "Mr. Smith went to Washington. He shook hands. He kissed babies.", then"

```
$MyString = $Text.sentence(0);
```

returns 'Mr. Smith went to Washington.' and

```
$Text.sentence(2)
```

returns "He kissed babies."

If the **sentenceNum** argument is omitted, the initial sentence is returned. This is sentence 0 (zero) as **sentenceNum** is a zero-based index.

The definition of a 'sentence' is heuristic, and varies depending on the locale. In the example above, notice Tinderbox (in en-US locale) recognises that the period following "Mr." ends an abbreviation, not a sentence. The locale is derived from the users macOS settings but can also be set contextually using locale().

String.show([backgroundColor[,colorString]])

Operator Type: Function [other Function type actions]
Operator Scope of Action: Item [operators of similar scope]
Operator Purpose: Data manipulation [other Data manipulation operators]
Operator First Added: 9.5.2
Operator Last Altered: As at baseline

Syntax note: *Operators without any defined mandatory arguments may omit their empty closing parentheses*

String.show([backgroundColor[,colorString]])
String.show()
String.show

New to v9.5.2, the **String.show()** function allows a plain-text message to be shown in the front window's message placard. The source text may be a literal string, variable or a String-type attribute value. The trailing parentheses are optional.

The **backgroundColor** argument may be used on its own, but if the optional **colorString** arguments is used, then the **backgroundColor** argument *must* be given as well.

The features and limitations of messages sent to the placard are described separately under the Message placards article.

The show(MsgString) operator offers a non-dot-operator alternative.

String.size()

Operator Type: Property [other Property type actions]
Operator Scope of Action: Item [operators of similar scope]
Operator Purpose: Data manipulation [other Data manipulation operators]
Operator First Added: Baseline
Operator Last Altered: As at baseline

Syntax note: *Operators without any defined mandatory arguments may omit their empty closing parentheses*

String.size()
String.size

This returns the Number of characters in a string value. The returned number can be coerced to a string. Examples:

```
$MyString = "hello world"; $MyNumber = $MyString.size; $MyStringA = $MyString.size;
```

The value of $MyNumber will be the number 11, $MyStringA will be the string "11".

This operator can also be used on other attribute data types that are string-like, URL, File, etc.

The 'size' of $Text is pre-computed and accessed via the read-only system attribute $TextLength.

More detail on how character counts are made.

String.skip(charsNum)

Operator Type: Function [other Function type actions]
Operator Scope of Action: Item [operators of similar scope]
Operator Purpose: Stream parsing [other Stream parsing operators]
Operator First Added: Baseline
Operator Last Altered: As at baseline

String.skip(charsNum)

Moves the current stream position **charsNum** characters forward (right). A failure occurs if **charsNum** reaches past the end of the **String** stream.

String.skipLine()

Operator Type: Function [other Function type actions]
Operator Scope of Action: Item [operators of similar scope]
Operator Purpose: Stream parsing [other Stream parsing operators]
Operator First Added: Baseline
Operator Last Altered: As at baseline

Syntax note: *Operators without any defined mandatory arguments may omit their empty closing parentheses*

String.skipLine()
String.skipLine

The dot-operator **String.skipLine** skips forward to the next carriage return or to the end of the **String** stream. **String.skipLine** fails if the string is exhausted, i.e. the stream cursor remains where it was before the call.

The sort of scenario with which this helps is where the desired line(s) has no identifiable label but is preceded by a separate area of the stream that can be found. Thus in a raw email transcript the content includes labels like 'To:' and 'Subject:' but has nothing for the main body copy, which simple form all content after the Subject line.

Most stream operators work up to the end of the current line but *exclude* the line break characters (`\n`) that separates each line. In the above scenario, where the needed line cannot be detected but the *preceding* line can be, .skipLine allow code like this:

```
$Source.captureTo("Subject:").skipLine.captureRest("Text")
```

String.skipTo(matchStr)

Operator Type: Function [other Function type actions]
Operator Scope of Action: Item [operators of similar scope]
Operator Purpose: Stream parsing [other Stream parsing operators]
Operator First Added: Baseline
Operator Last Altered: As at baseline
Operator Uses Regular Expressions: [More on regular expressions in Tinderbox]

String.skipTo(matchStr)

Looks for the **matchStr** (N.B. this is **not** a regular expression) in the source **String** stream. If not found, it returns nothing. If found, it returns the string that follows **matchStr**.

String.skipToNumber()

Operator Type: Function [other Function type actions]
Operator Scope of Action: Item [operators of similar scope]
Operator Purpose: Stream parsing [other Stream parsing operators]
Operator First Added: Baseline
Operator Last Altered: As at baseline

Syntax note: *Operators without any defined mandatory arguments may omit their empty closing parentheses*

String.skipToNumber()
String.skipToNumber

Advances the **String** stream to the next number (i.e. one of more continuous number characters). A failure occurs if the stream is exhausted.

String.skipWhitespace()

Operator Type: Function [other Function type actions]
Operator Scope of Action: Item [operators of similar scope]
Operator Purpose: Stream parsing [other Stream parsing operators]
Operator First Added: Baseline
Operator Last Altered: As at baseline

Syntax note: *Operators without any defined mandatory arguments may omit their empty closing parentheses*

String.skipWhitespace()
String.skipWhitespace

Advances the **String** stream to the first character that is not whitespace. A failure occurs if the stream is exhausted.

String.speak([voiceNameStr])

Operator Type: Function [other Function type actions]
Operator Scope of Action: Item [operators of similar scope]
Operator Purpose: Data manipulation [other Data manipulation operators]
Operator First Added: Baseline
Operator Last Altered: 9.5.0
Operator Has Optional Arguments: [More on optional operator arguments]

Syntax note: *Operators without any defined mandatory arguments may omit their empty closing parentheses*

String.speak([voiceNameStr])

speaks a string using Mac text-to-speech. If another string is being spoken, the new phrase is spoken after the current phrase is complete.

```
$Text.speak();
```

An optional second argument **voiceNameStr** identifies the voice the speech synthesiser should use. If the nominated **voiceNameStr** is not present no sound is played. The closing parentheses are only needed if the extra argument is being passed:

```
$Text.speak("Tessa");
```

From v9.5.0, quoted literal strings can be chained to .speak():

```
"Look on my Works, ye Mighty, and despair".speak();
'Look on my Works, ye Mighty, and despair'.speak();
```

String.split(regexStr)

Operator Type:	Function [other Function type actions]
Operator Scope of Action:	Item [operators of similar scope]
Operator Purpose:	Data manipulation [other Data manipulation operators]
Operator First Added:	Baseline
Operator Last Altered:	As at baseline
Operator Uses Regular Expressions:	[More on regular expressions in Tinderbox]

String.split(regexStr)

This operator splits a string into a List, as divided by instances of regular expression pattern **regexStr** in the original string. Source characters matched by **regexStr** are not passed to the list. The source string itself is not affected.

regexStr is one of:

- an action code expression (which includes just referencing a single attribute name')
- a quoted string; quoted strings may be either:
  - a literal string (i.e. actual text)
  - a regular expression

Useful **regex** values are:

- `"\W+"`. This splits the source at word boundaries removing spaces and punctuation.
- `"\n"`. This divides the string into discrete paragraphs, ignoring blank lines and/or lines/paragraphs with only spaces but no textual content.
- `"\."`. This divides on sentences ending with a period. It will strip the terminating punctuation.
- `"[\.\?\!]"`. As above but the sentence may end with any of full stop, question mark or exclamation mark.

The result of the operator is a List-type attribute value, i.e. the data should be passed to a list. Passing the output to a Set-type attribute will de-dupe any list values in the output with the first instance of any duplicates forming its set entry.

For example:

```
$MyList = [ant bee ant cow].split(" "); gives [ant;bee;ant;cow]
$MySet = [ant bee ant cow].split(" "); gives [ant;bee;cow]
$MyList = [ant, bee, cow].split("\W+ "); gives [ant;bee;cow]
$MyList = [ant, bee, cow].split(" "); gives [ant;bee,;cow]
$MyList = $MyString.split($MyString(agent));
$MyList = $MyString(parent).split("and");
```

If the string, stored in $MyString, is multi-line:

```
ant
bee
cow
```

…then:

```
$MyList = $MyString.split("\n");
```

gives [ant;bee;cow].

This approach can be useful if trying to retrieve a specific paragraph of $Text, perhaps from notes exploded from a larger consistently formatted text source. To get a string holding just paragraph #3 of the source $Text (or other multi-line string data):

```
$MyString = $Text.split("\n").at(2);
```

Do not overlook the fact that that List.at() is zero-based. That means the first list item is .at(0) and so the third list item is '2' and not '3' as might otherwise be assumedhe last item is '-1':

```
$MyString = $Text.split("\n").at(-1);
```

There is one one limitation of this approach to working with $Text or multi-line strings. The issue is that blank lines or lines with only spaces, are ignored; lists do not hold 'empty' items. So if the string $MyString is multi-line and contains blank lines, like so:

```
ant

bee
cow
```

…then:

```
$MyList = $MyString.split("\n");
```

still gives [ant;bee;cow].

It does not matter if the blank is just two successive line returns or actually contains some white space, no list item is created for it.

Luckily there is a simple workaround is to seed empty lines with a single hyphen (or whatever placeholder you prefer, e.g. "N/A" or such). Thus:

```
$MyList = $Text.replace("\n\n","\n-\n").split("\n");
```

…now gives $MyList [ant;-;bee;cow] such that "bee" is still paragraph #3 of the new list, as in the original text. If you wanted to make a deliberate review of such data you might use a more distinctive marker string:

```
$MyList = $Text.replace("\n\n","\n#####\n").split("\n");
```

You could then query for `$MyList.contains("#####")`.

Dealing with inline quote characters

Because **regex** is parsed for regular expressions, it may be possible to use the '\dnn' form described here to work around the lack of escaping from single double quotes within strings.

Dealing with inline semi-colons

As this function outputs a list, where values are semi-colon delimited, if the source string—such as $Text—has semicolons in it they act as extra (unexpected!) splits when viewing the outcome. To get around this, escape the semicolons on the fly:

```
$MyList = $Text.replace(";","\\;").split("\n");
```

However, the surviving inline semicolons in the resulting List items will get misread, when interrogating the List, as item delimiters. In such cases, first replace inline semicolons with another character(s) before using the split-generated list and then reverse the replacement before actual use of the text. For example:

```
var:string vSource = $Text.replace(";","@@@");
$MyList = vSource.paragraphList;
$MyString = $MyList.at(4)replace("@@@",";");
```

String.substr(startNum[, lengthNum])

Operator Type:	Function [other Function type actions]
Operator Scope of Action:	Item [operators of similar scope]
Operator Purpose:	Data manipulation [other Data manipulation operators]
Operator First Added:	Baseline
Operator Last Altered:	As at baseline
Operator Has Optional Arguments:	[More on optional operator arguments]

String.substr(startNum[, lengthNum])

This operator allows extraction of a substring from a string attribute. The source string is not affected.

```
$MyString.substr(startN)
```

returns the substring of $MyString beginning **startNum** characters from the beginning. The numbers for startN and lengthN are 0-based, i.e. zero is character position #1. A negative **startNum** value counts back from the end of the string .Negative values are 1-based, i.e. the -1 represents the last character in the string.

In the examples below assume $MyString's value is "Hello World". Examples:

```
$MyString = "abcde".substr(2); returns "cde"
$MyString = "abcde".substr(-2); returns "de"
$MyString = $MyString.substr(6); returns "World"
```

If the string does not contain at least **startNum** characters, the empty string is returned.

A second argument **lengthNum** specifies the length of the returned string. If unspecified, the entire remaining string is returned.

```
$MyString = "abcde".substr(2,2); returns "cd"
$MyString = "abcde".substr(-3,2); returns "cd"
$MyString = $MyString.substr(0,5); returns "Hello"
```

If the length of the substring is negative, it is treated as an offset from the end of the string.

```
$MyString = "Hello".substr(1,-1); → "ell"
```

Besides strings and string literals, this operator can also be used on other attribute data types that are string-like, URL, File, etc. Although the operator also works on lists/sets, the source data is all the values as a single semi-colon-joined

string literal so there is less point in its use in this context.

This function respects existing rich text styling.

Trimming leading/trailing quotes

See String.trim().

Working with styled text

This operator is capable of worthing with StyledString operators: StyledString.bold, StyledString.fontSize(), StyledString.italic and StyledString.strike.

String.toNumber()

Operator Type: Function [other Function type actions]
Operator Scope of Action: Item [operators of similar scope]
Operator Purpose: Data manipulation [other Data manipulation operators]
Operator First Added: Baseline
Operator Last Altered: As at baseline

Syntax note: *Operators without any defined mandatory arguments may omit their empty closing parentheses*

String.toNumber()

String.toNumber

Used for explicit coercion of numbers in strings to Number type data. Examples:

```
$MyNumber = $MyString.toNumber;
$MyNumberA = "40".toNumber;
```

This can be useful if default type coercion is not working as expected.

String.tr(inStr[, outStr])

Operator Type: Function [other Function type actions]
Operator Scope of Action: Item [operators of similar scope]
Operator Purpose: Data manipulation [other Data manipulation operators]
Operator First Added: Baseline
Operator Last Altered: As at baseline
Operator Has Optional Arguments: [More on optional operator arguments]

String.tr(inStr[, outStr])

$MyString.tr(inStr, outStr)

This operator allows simple single character string manipulation. It computes a new string, copying each character of $MyString but converting any characters in **inStr** to the corresponding characters in **dataOut**. For example:

```
$MyString = $MyString.tr("a","A");
```

returns a copy of MyString in which every "a" is converted to "A".

Backslash characters must be quoted and escaped:

```
$MyString = $MyString.tr("c","\\r");
```

converts every "c" to a Macintosh newline characters (\r). Note the need in this context for an extra backslash escape (so Tinderbox knows the intended swap value is "\r" and not "r").

Multiple characters can be replaced:

```
$MyString = "Hello, world".tr("aeiou","AEIOU");
```

gives "HEllO, wOrld"

Note that in the later example the number of characters in **inStr** and **outStr** must match and pairs must list in order, otherwise unmatched characters will as as for the syntax below.

$MyString.tr(inStr)

If **outStr** is omitted or left empty, any matches to **inStr** are deleted from the referenced string.

```
$MyString = "Hello, world".tr("aeiou");
```

gives "Hll, wrld"

For further information, see the macOS man page for the UNIX tr command.

Performance

If using actions with a *lot* of .tr() calls, based on some testing in large complex docs, it may—counter-intuitively—be faster using .replace() instead.

String.trim([filterStr])

Operator Type: Function [other Function type actions]
Operator Scope of Action: Item [operators of similar scope]
Operator Purpose: Data manipulation [other Data manipulation operators]
Operator First Added: Baseline
Operator Last Altered: As at baseline
Operator Has Optional Arguments: [More on optional operator arguments]

Syntax note: *Operators without any defined mandatory arguments may omit their empty closing parentheses*

String.trim([filterStr])

String.trim()

String.trim

The .trim operator removes unwanted characters from the start and end of a string. With no arguments, **.trim()** removes whitespace and end of line characters.

```
"Hello world ".trim; → "Hello world"
```

It may also be used with a literal string argument **filterStr**. The only accepted value for **filterStr** is "*punctuation*", in which case the operator also removes punctuation marks.

```
"[tab] Hello world?! ".trim("punctuation") → "Hello world"
```

The resulting string retains any stying in the source string.

String.try{actions}[.thenTry{actions}]

Operator Type: Function [other Function type actions]
Operator Scope of Action: Item [operators of similar scope]
Operator Purpose: Stream parsing [other Stream parsing operators]
Operator First Added: Baseline
Operator Last Altered: As at baseline
Operator Has Optional Arguments: [More on optional operator arguments]

String.try{ action(s) }[.thenTry{ action(s) }]

If the first test action fails, they restore the string and allow you to process it a different way. The current success/failure state can be tested using the .failed() operator.

Saves the value of an attribute (the steam source) and attempts an action. If the action fails because one of its operators fails or the fail() operator is performed, the original value of **String** stream source is restored.

A .try{} may be followed by one or more .thenTry{} clauses. If the original .try{} succeeds, all subsequent .thenTry{} clauses are ignored. If the original .try{} fails, then each .thenTry{} is attempted in turn. Once a .thenTry{} clause succeeds, subsequent .thenTry{} clauses are ignored.

For example:

```
$MyString.try{
   $MyString=$MyString.skipToWord("To:").captureWord;
}.thenTry{
   fail(); //signal an explicit failure and return a 'false' value to an enclosing expression
};
```

looks for strings such as "To:John Doe"
If the string "To:" is not found, $MyString will be unchanged. If found, $MyString's value is set to the word that follows, up to the next whitespace character or the end of the string, i.e. "John".

String.uppercase()

Operator Type: Function [other Function type actions]
Operator Scope of Action: Item [operators of similar scope]
Operator Purpose: Formatting [other Formatting operators]
Operator First Added: Baseline
Operator Last Altered: As at baseline

Syntax note: *Operators without any defined mandatory arguments may omit their empty closing parentheses*

String.uppercase()
String.uppercase
Returns the referenced String, transforming all uppercase letters to lowercase.
The function can be chained to both string data and to string literals:

```
$MyString.uppercase()
"my new title".uppercase()
```

The latter gives "MY NEW TITLE".
The trailing parentheses may be omitted:

```
$MyString.uppercase
```

Functionally equivalent to uppercase().
If $MyString is "hello world":

```
$MyString = $MyString.uppercase;
```

$MyString is set to "HELLO WORLD".
The .uppercase() method may also be used on Lists or Sets. Consider [Ant;BEE;Cow] stored in $MyList:

```
$MyList = $MyList.uppercase;
```

… giving [ANT;BEE;COW].

String.wordCount()

Operator Type: Property [other Property type actions]
Operator Scope of Action: Item [operators of similar scope]
Operator Purpose: Data manipulation [other Data manipulation operators]
Operator First Added: 9.5.2
Operator Last Altered: As at baseline

Syntax note: *Operators without any defined mandatory arguments may omit their empty closing parentheses*

String.wordCount()
String.wordCount
From v9.5.2, **String.wordCount** returns the number of words in a string. The string is first scanned to determine its dominant language, and the word count is based on the conventions of that language (q.v. $WordCount). If the language cannot be determined or if it is not known to macOS, the conventions of English are used.

String.wordList()

Operator Type: Property [other Property type actions]
Operator Scope of Action: Item [operators of similar scope]
Operator Purpose: Data manipulation [other Data manipulation operators]
Operator First Added: Baseline
Operator Last Altered: As at baseline

Syntax note: *Operators without any defined mandatory arguments may omit their empty closing parentheses*

String.wordList()
String.wordList
This returns a list of each noun in the supplied string, excepting those recognised as pronouns and proper names. Note that the output list is all lowercase, regardless of source case. This operator requires running on macOS 10.14 and later.

```
$MyList = "I am the very model of a modern Major-General.".wordList;
```

then MyList holds "i;am;the;very;model;of;a;modern;major;general".

String.words(wordsNum)

Operator Type: Function [other Function type actions]
Operator Scope of Action: Item [operators of similar scope]
Operator Purpose: Data manipulation [other Data manipulation operators]
Operator First Added: Baseline
Operator Last Altered: As at baseline

String.words(wordsNum)
returns the first **number** words of a string. If the string has fewer than **wordsNum** words, returns the entire string. If **wordsNum** is negative, it returns a negative - **wordsNum** words from the end of the string. If $MyString is "Romans, they g
house?".

```
$MyString = $MyString.words(2);
```

returns "Romans, they".

```
$MyString = $MyString.words(-1);
```

returns "house?". Note that punctuation contiguous to words gets passed through as part of the returned word(s).

StyledString.bold()

Operator Type: Function [other Function type actions]
Operator Scope of Action: Item [operators of similar scope]
Operator Purpose: Formatting [other Formatting operators]
Operator First Added: Baseline
Operator Last Altered: As at baseline

Syntax note: *Operators without any defined mandatory arguments may omit their empty closing parentheses*

StyledString.bold()
StyledString.bold
The function sets the referenced string's RTF styled text to bold text.

```
$Text = $Text + "This is " + "some".bold + " text".plain;
```

adds "this is **some** text" to the end of $Text. The '.plain' command reasserts no style is to be applied.
Note that $Text is currently the only attribute capable of holding styled text. String attributes and action code variables, i.e. var, cannot store styled text. If passed styled text the latter will store the unstyled version of the text being passed. S

a more detailed explanation.

String operators respecting StyledString operators

String.paragraphs(), String.replace() and String.substr().

StyledString.fontSize(pointSizeNum)

Operator Type:	Function [other Function type actions]
Operator Scope of Action:	Item [operators of similar scope]
Operator Purpose:	Formatting [other Formatting operators]
Operator First Added:	Baseline
Operator Last Altered:	As at baseline

StyledString.fontSize(pointSizeNum)

The function sets the referenced string's (RTF) styled text to font size **pointSizeNum**, where **pointSizeNum i**s the desired font size in points.

```
$Text = $Text + "some bigger text".fontSize(N);
```

adds test "some bigger text" in 24pt to the end of $Text.

It is also a good automatic way to reset $Text containing old (or accidental) size-based auto-headings. Code like this can be used as a stamp or other action to reset multiple selected notes:

```
$Text = $Text.fontSize($TextFontSize);
```

This will make each (selected) note use it's own $TextFontSize (which might differ) to reset the size of the entire $Text of the note.

Note that $Text is currently the only attribute capable of holding styled text. String attributes and action code variables, i.e. var, cannot store styled text. If passed styled text the latter will store the unstyled version of the text being passed. Se
a more detailed explanation.

String operators respecting StyledString operators

String.paragraphs(), String.replace() and String.substr().

StyledString.italic()

Operator Type:	Function [other Function type actions]
Operator Scope of Action:	Item [operators of similar scope]
Operator Purpose:	Formatting [other Formatting operators]
Operator First Added:	Baseline
Operator Last Altered:	As at baseline

Syntax note: *Operators without any defined mandatory arguments may omit their empty closing parentheses*

StyledString.italic()

StyledString.italic

The function sets the referenced string's (RTF) styled text to italic.

```
$Text = $Text + "This is " + "some".italic + " text".plain;
```

adds "this is *some* text" to the end of $Text. The '.plain' command reasserts no style is to be applied.

Note that $Text is currently the only attribute capable of holding styled text. String attributes and action code variables, i.e. var, cannot store styled text. If passed styled text the latter will store the unstyled version of the text being passed. Se
a more detailed explanation.

String operators respecting StyledString operators

String.paragraphs(), String.replace() and String.substr().

StyledString.plain()

Operator Type:	Function [other Function type actions]
Operator Scope of Action:	Item [operators of similar scope]
Operator Purpose:	Formatting [other Formatting operators]
Operator First Added:	Baseline
Operator Last Altered:	As at baseline

Syntax note: *Operators without any defined mandatory arguments may omit their empty closing parentheses*

StyledString.plain()

StyledString.plain

The function (re-)sets the referenced string's (RTF) styled text to the default face.

```
$Text = $Text + "This is " + "some".italic + " text".plain;
```

adds "this is *some* text" to the end of $Text. The '.plain' command reasserts no style is to be applied.

Note that $Text is currently the only attribute capable of holding styled text. String attributes and action code variables, i.e. var, cannot store styled text. If passed styled text the latter will store the unstyled version of the text being passed. Se
a more detailed explanation.

Re-setting a note's $Text styling to defaults via code or stamp

The Format ▸ Style menu allows the user to reset the default $TextFont/$TextSize and (ruler) styles. To replicate all those defaults for all the existing $Text, use:

```
$Text = $Text.plain;
```

String operators respecting StyledString operators

String.paragraphs(), String.replace() and String.substr().

StyledString.strike()

Operator Type:	Function [other Function type actions]
Operator Scope of Action:	Item [operators of similar scope]
Operator Purpose:	Formatting [other Formatting operators]
Operator First Added:	Baseline
Operator Last Altered:	As at baseline

Syntax note: *Operators without any defined mandatory arguments may omit their empty closing parentheses*

StyledString.strike()

StyledString.strike

The function sets the referenced string's (RTF) styled text to stuck-through text.

```
$Text = $Text + "This is " + "some".strike + " text".plain;
```

adds "this is ~~some~~ text" to the end of $Text. The '.plain' command reasserts no style is to be applied.

Note that $Text is currently the only attribute capable of holding styled text. String attributes and action code variables, i.e. var, cannot store styled text. If passed styled text the latter will store the unstyled version of the text being passed. Se
a more detailed explanation.

String operators respecting StyledString operators

String.paragraphs(), String.replace() and String.substr().

StyledString.textColor(aColor)

Operator Type: Function [other Function type actions]
Operator Scope of Action: Item [operators of similar scope]
Operator Purpose: Data manipulation [other Data manipulation operators]
Operator First Added: Baseline
Operator Last Altered: As at baseline

StyledString.textColor(aColor)

The operator **.textColor(aColor)**, when applied to styled text, sets the foreground colour of the text. The **color** argument can be either a named colour or a hexadecimal string. For example, both of these examples add 'example' in bright red text to the current note's $Text.

```
$Text = $Text + "example".textColor("#FF0000")
```

Note that $Text is currently the only attribute capable of holding styled text. String attributes and action code variables, i.e. var, cannot store styled text. If passed styled text the latter will store the unstyled version of the text being passed. Se a more detailed explanation.

.textColor() respects the current text font, e.g. if set to a non-default colour.

substr(dataStr, startNum[, lengthNum])

Operator Type: Function [other Function type actions]
Operator Scope of Action: Item [operators of similar scope]
Operator Purpose: Data manipulation [other Data manipulation operators]
Operator First Added: Baseline
Operator Last Altered: As at baseline
Operator Has Optional Arguments: [More on optional operator arguments]
Operator Has Newer Dot-Operator Variant: Yes

substr(dataStr, startNum, lengthNum)

Where **dataStr** is a string literal, string attribute or expression evaluating to a string.

Where **startNum** is the zero-based position to start the substring. Negative **startNum** values are not supported, though String.substring() does so and may be used instead.

Where **lengthNum** is the length of the returned string. If not specified, the entire string from (or back from) **startNum** is returned.

Extracts a substring of data. For example:

```
$MyString = substr("test",0,1);
```

returns "t", while

```
$MyString = substr("test",1,2);
```

returns "es". All arguments are evaluated; if the designated characters do not exist, an empty string is returned.

If the length of the substring is negative, it is treated as an offset from the end of the string.

```
$MyString = substr("Hello",1,-1);
```
→ "ell"

Note that in expressions like

```
$Initial=substr($Name,0,1);
```

the value of $Name is not changed.

sum_if(scope, condition, expressionStr)

Operator Type: Function [other Function type actions]
Operator Scope of Action: Conditional Group [operators of similar scope]
Operator Purpose: Mathematical [other Mathematical operators]
Operator First Added: Baseline
Operator Last Altered: As at baseline
Operator Uses Regular Expressions: [More on regular expressions in Tinderbox]
Operator Uses Scoped Arguments: [More on scoped arguments in Action Code]
Operator Has Conditional Arguments: [More on conditional operator arguments]
Operator Has Newer Dot-Operator Variant: Yes

sum_if(scope, condition, expressionStr)

This computes the Number sum of every **expressionStr** value in each **scope** item (defining **scope**), as filtered by a **condition** expression. See sum() for a related non-conditional operator.

scope describes the notes to be examined and may be any group designator including a find() query. sum_if() omits notes for which $Searchable is false.

In addition, where may be argument that designates a particular (single) note other than this.

condition is action code forming a valid conditional query test, i.e. it equates to `true` when matched. Some query-style operators terms may allow use of regular expressions.

expressionStr can be an expression, but is typically a Number-attribute value or a List/Set holding a list of numbers. It can also be a literal number 1, i.e. if the test is `true` from that item then add one to the retuned value of sum_if().

For example,

```
$MyNumber = sum_if(children,$Prototype=="p_Problem",1);
```

sums the number of children of the current note whose prototype is 'p_Prototype'. If tested value is a string with spaces, e.g. "p Prototype" vs. "p_Prototype" then use double quotes around the value.

The newer count_if() offers a more intuitive method of counting matches rather than the value of matched items.

sum(scope, expressionStr)

Operator Type: Function [other Function type actions]
Operator Scope of Action: Group [operators of similar scope]
Operator Purpose: Mathematical [other Mathematical operators]
Operator First Added: Baseline
Operator Last Altered: As at baseline
Operator Uses Scoped Arguments: [More on scoped arguments in Action Code]
Operator Has Newer Dot-Operator Variant: Yes

sum(scope, expressionStr)

This computes the arithmetic Number of every **expressionStr** value in each in-**scope** item (defining **scope**). See sum_if() for a related conditional operator.

The **sum()** operator omits notes for which $Searchable is `false`.

expressionStr can be an expression, but is typically a Number-attribute value or a List/Set holding a list of numbers. It can also be a literal number 1, i.e. if the test is `true` from that item then add one to the retuned value of sum(). The latte less likely than with the sibling operator sum_if().

For example,

```
$MyNumber = sum(children,$WordCount);
```

constructs the current word count of the children of the note.

See also List/Set.sum().

tan(radiansNum)

Operator Type: Function [other Function type actions]
Operator Scope of Action: Item [operators of similar scope]
Operator Purpose: Mathematical [other Mathematical operators]
Operator First Added: Baseline
Operator Last Altered: As at baseline

tan(radiansNum)

tan() converts its **radiansNum**, in *radians*, to the tangent of that value.

```
$MyNumber = tan(3)
```

returns '-0.1425465431' for an input of 3 radians.

time(aDate, hoursNum, minutesNum, secondsNum)

Operator Type: Function [other Function type actions]
Operator Scope of Action: Item [operators of similar scope]
Operator Purpose: Date-time [other Date-time operators]
Operator First Added: Baseline
Operator Last Altered: As at baseline

---

time(aDate, hoursNum, minutesNum, secondsNum)

creates a new Date based on the **aDate** expression, but in which the time is set by **hoursNum**, **minutesNum**, and **secondsNumDate** is not changed.

For example, to make a stamp or rule that sets the time element of all event start/end dates in a timeline to 12:00:00 AM (midday) use this code:

```
$StartDate=time($StartDate,12,0,0);
if($EndDate){
   $EndDate=time($EndDate,12,0,0)
};
```

time(aDate)

Operator Type: Function [other Function type actions]
Operator Scope of Action: Item [operators of similar scope]
Operator Purpose: Date-time [other Date-time operators]
Operator First Added: Baseline
Operator Last Altered: As at baseline

---

time(taDate)

returns a string of the hours/minutes time part the **aDate** date/time expression, which may simply be a date-type attribute value. For instance:

```
$MyString = time($StartDate);
```

If the time of $StartTime is 14:20 then $MyString will be "14:20". Passing to a string will elide leading zeroes, so a time of 05:08 will return "5:08".

To get seconds as well as hours and minutes, use the .format() or format() operator on **aDate** with an appropriate formatting string.

Note that the **time()** operator does not ignore seconds in the source data.

twitter(usernameStr, statusStr)

Operator Type: Function [other Function type actions]
Operator Scope of Action: Item [operators of similar scope]
Operator Purpose: Data manipulation [other Data manipulation operators]
Operator First Added: Baseline
Operator Last Altered: As at baseline

---

twitter(usernameStr,statusStr)

NOW DEPRECATED and no longer working - for info only

allows posting to the user's twitter account (**usernameStr**) of a tweet (**statusStr**).

Tinderbox will request permission to use your Twitter account, and will send the tweet to that account.

```
twitter("myusername","my test tweet")
```

The twitter() action is no longer supported (reflecting changes to Twitter API).

type(attributeNameStr)

Operator Type: Function [other Function type actions]
Operator Scope of Action: Document [operators of similar scope]
Operator Purpose: Document configuration [other Document configuration operators]
Operator First Added: Baseline
Operator Last Altered: As at baseline

---

type(attributeNameStr)

This operator may be replaced by the more flexible attribute() operator.

The operator **type(attributeNameStr)** returns a string representing the type of the designated attribute. The attribute may simply be the attribute name (without quotes or a preceding "$") or an expression that, when evaluated, yields an attribute name.

```
$MyString = type(Width); → "number"
$MyString = type("AccentColor"); → "color"
$MyString="Modified"; type($MyString); → "date"
```

Note that **type($MyString)** returns the type of the attribute who name is stored in $MyString, while **type("MyString")** or **type(MyString)** returns the type of the attribute MyString.

If no such attribute exists in the document, the operator returns the empty string.

unlinkFrom(scope[, linkTypeStr])

Operator Type: Function [other Function type actions]
Operator Scope of Action: Group [operators of similar scope]
Operator Purpose: Linking [other Linking operators]
Operator First Added: Baseline
Operator Last Altered: 9.6.0
Operator Uses Scoped Arguments: [More on scoped arguments in Action Code]
Operator Has Optional Arguments: [More on optional operator arguments]

---

unlinkFrom(scope[, linkTypeStr])

This removes *all* basic links from **scope** to the current note (i.e. inbound links).

The **scope** argument must be quoted unless an attribute reference, e.g. `"Some note"` vs. `$MyString`. Ways to define **scope**. Note that here, unlike other **scope** uses, *$Path can **not** be used*.

scope may be **group** scoped including use of group designators and operators like find,() collect and links().

linkTypeStr (string). Optionally, link deletion can be constrained to one or more types supplied in **linkTypeStr**. The argument allows simple regular expressions; "disagree|example" would delete links of type 'disagree' and 'example' but leav
others untouched. If no **linkTypeStr** argument is supplied, only untitled (i.e. no link type) links are deleted. To delete all links of any type of none use "*".

Both arguments are evaluated. This operator does not require a left-side argument, simply calling effects a result. No change occurs if the described link does not exist.

Examples

Unlinking a note "Some note":

all link types: `unlinkFrom("Some note");`

only link type 'agree': `unlinkFrom("Some note","agree");`

either of 2 link types: `unlinkFrom("Some note", "example|disagree");`

Unlinking the first child (via a designator):

```
unlinkFrom(child);
unlinkFrom(child,"agree");
```

See further below for group-scope references.

From v9.6.0, **unlinkFrom()** no longer deletes prototype links.

Relevant similar operators: linkTo, linkFrom, unlinkTo.

Use of this action does not shift note focus; in addition if **scope** contains operators (brackets, plus, minus, etc.) Tinderbox will first look for a match to the literal **scope** string and only if there is no match will the app try evaluating to operato and testing the resulting string. For example:

```
unlinkFrom("Example 1 (a test)");
```

will unlink the note named 'Example 1 (a test)'. If no note matches this string, Tinderbox will attempt to evaluate the string. Thus for:

```
unlinkFrom("2+2");
```

will unlink the note named '2+2' but if there is no match Tinderbox will look for a note named '4'.

This function can unlink an alias as opposed to an original (if the logical choice) and can also accept a group scope. An example of group scope is the following code that could be used in a rule or stamp to remove in/bound footnote links from one or more selected notes. Use of the 'all' designator removes the need to know the name of the notes for which the selection are footnotes:

```
unlinkFrom(all,"note");unlinkTo(all,"note+");
```

Use in agents

Beware that the action is working on an *alias* of the current note and note the current note itself. As originals and aliases support discrete basic links this function should not generally be used in an agent. The best way to use the function is by using a prototype and apply a $Rule to it thus running the code in all notes using the prototype.

An alternative unlinkFromOriginal() code will ensure any link deleted is between two original notes regardless of whether an alias is the context of execution of the code.

unlinkFromOriginal(scope[, linkTypeStr])

Operator Type:	Function [other Function type actions]
Operator Scope of Action:	Group [operators of similar scope]
Operator Purpose:	Linking [other Linking operators]
Operator First Added:	Baseline
Operator Last Altered:	9.6.0
Operator Uses Scoped Arguments:	[More on scoped arguments in Action Code]
Operator Has Optional Arguments:	[More on optional operator arguments]

---

unlinkFromOriginal(scope[, linkTypeStr])

This function works exactly as the same as unlinkFrom(), *except* for one important difference that the link deleted is *always between two originals* even if either/both the evaluated source or destination are an alias.

For more detail of use, see unlinkFrom().

See also linkFromOriginal(), linkToOriginal(), unlinkToOriginal().

From v9.6.0, **unlinkFromOriginal()** no longer deletes prototype links.

unlinkTo(scope[, linkTypeStr])

Operator Type:	Function [other Function type actions]
Operator Scope of Action:	Group [operators of similar scope]
Operator Purpose:	Linking [other Linking operators]
Operator First Added:	Baseline
Operator Last Altered:	9.6.0
Operator Uses Scoped Arguments:	[More on scoped arguments in Action Code]
Operator Has Optional Arguments:	[More on optional operator arguments]

---

unlinkTo(scope[, linkTypeStr])

This removes *all* basic links to **scope** from the current note (i.e. outbound links).

The **scope** argument must be quoted unless an attribute reference, e.g. `"Some note"` vs. `$MyString`. Ways to define **scope**. Note that here, unlike other **scope** uses, *$Path can **not** be used*.

The **scope** may be **group** scoped including use of group designators and operators like find,() collect() and links().

linkTypeStr (string). Optionally, link deletion can be constrained to one or more types supplied in **linkTypeStr**. The argument allows simple regular expressions; "disagree|example" would delete links of type 'disagree' and 'example' but leav others untouched. If no **linkTypeStr** argument is supplied, only untitled (i.e. no link type) links are deleted. To delete all links of any type of none use "".

Both arguments are evaluated. This operator does not require a left-side argument, simply calling effects a result. No change occurs if the described link does not exist.

Unlinking a note "Some note":

all link types: `unlinkTo("Some note");`

only link type 'agree': `unlinkTo("Some note","agree");`

either of 2 link types: `unlinkTo("Some note", "agree|disagree");`

Matching the first child (via a designator):

```
unlinkTo(child);
unlinkTo(child,"agree");
```

See further below for group-scope references.

From v9.6.0, **unlinkTo()** no longer deletes prototype links.

Relevant similar operators: linkTo, linkFrom, unlinkFrom.

Use of this action does not shift note focus; in addition if **scope** contains operators (brackets, plus, minus, etc.) Tinderbox will first look for a match to the literal **scope** string and only if there is no match will the app try evaluating to operato and testing the resulting string. For example:

```
unlinkTo("Example 1 (a test)");
```

will unlink the note named 'Example 1 (a test)'. If no note matches this string, Tinderbox will attempt to evaluate the string. Thus for:

```
unlinkTo("2+2");
```

will unlink the note named '2+2' but if there is no match Tinderbox will look for a note named '4'.

This function can unlink an alias as opposed to an original (if the logical choice) and can also accept a group scope. An example of group scope is the following code that could be used in a rule or stamp to remove in/bound footnote links from one or more selected notes. Use of the 'all' designator removes the need to know the name of the notes for which the selection are footnotes:

```
unlinkFrom(all,"note");unlinkTo(all,"note+");
```

Use in agents

Beware that the action is working on an *alias* of the current note and note the current note itself. As originals and aliases support discrete basic links this function should not generally be used in an agent. The best way to use the function is by using a prototype and apply a $Rule to it thus running the code in all notes using the prototype.

An alternative unlinkToOriginal() code will ensure any link deleted is between two original notes regardless of whether an alias is the context of execution of the code.

unlinkToOriginal(scope[, linkTypeStr])

Operator Type:	Function [other Function type actions]
Operator Scope of Action:	Group [operators of similar scope]
Operator Purpose:	Linking [other Linking operators]
Operator First Added:	Baseline
Operator Last Altered:	9.6.0
Operator Uses Scoped Arguments:	[More on scoped arguments in Action Code]
Operator Has Optional Arguments:	[More on optional operator arguments]

---

unlinkToOriginal(scope[, linkTypeStr])

This function works exactly as the same as unlinkTo(), *except* for one important important difference that the link deleted is *always between two originals* even if either/both the evaluated source or destination are an alias.

For more detail of use, see unlinkTo().

See also linkFromOriginal(), linkToOriginal(), unlinkFromOriginal().

From v9.6.0, **unlinkToOriginal()** no longer deletes prototype links.

update(scope)

Operator Type: Function [other Function type actions]
Operator Scope of Action: List [operators of similar scope]
Operator Purpose: Data manipulation [other Data manipulation operators]
Operator First Added: Baseline
Operator Last Altered: As at baseline
Operator Uses Scoped Arguments: [More on scoped arguments in Action Code]

update(scope)

The action **update(scope)** asks Tinderbox to update *one or more* notes defined in the list **scope** by evaluating both their rule and edict of each listed note (defining **scope**). If the note has been evaluated recently, Tinderbox will not evaluate again.

update() returns a list of updated notes. This allow the user, to check, if necessary, which notes have actually been updated

For example, suppose there is a note 'INSTALL DISHWASHER', which has a rule saying that the dishwasher cannot start to be installed until the task of note 'FLOOR' is finished.

```
if($StartDate<$EndDate(FLOOR)) {
 $StartDate=$EndDate(FLOOR);
 $EndDate=$StartDate+$MyInterval;
}
```

This rule works best if the FLOOR is up to date. So, before doing anything else, the FLOOR note is updated:

```
update(FLOOR);
if($StartDate<$EndDate(FLOOR)) {
 $StartDate=$EndDate(FLOOR);
 $EndDate-$StartDate+$MyInterval;
}
```

Note that, if FLOOR has been updated "recently", it will not be updated again; the prevents Tinderbox from doing lots of unwanted and unnecessary work.

Note also that we can use update() at the start of an action to be sure we are ready to work, and we can also use **update()** after an action if we know some other note will want to respond to our new values.

On the whole, **update()** is envisaged an esoteric command for unusual cases, but may prove popular for some users with *demanding* applications. For light/occasional use, a need to use update() is likely an indication that some review of current code use is needed.

uppercase(dataStr)

Operator Type: Function [other Function type actions]
Operator Scope of Action: Item [operators of similar scope]
Operator Purpose: Formatting [other Formatting operators]
Operator First Added: Baseline
Operator Last Altered: As at baseline
Operator Has Newer Dot-Operator Variant: Yes

uppercase(dataStr)

The contents of **dataStr** are returned entirely in upper case.

If $MyString is "hello world":

```
$MyString = uppercase($MyString);
```

$MyString is set to "HELLO WORLD".

Functionally equivalent to String.uppercase().

urlEncode(dataStr)

Operator Type: Function [other Function type actions]
Operator Scope of Action: Item [operators of similar scope]
Operator Purpose: Formatting [other Formatting operators]
Operator First Added: Baseline
Operator Last Altered: As at baseline

urlEncode(dataStr)

urlEncode() converts a **dataStr** string to 'urlEncoded' form for for the web, in accordance with RFC1738. URLs should not contain spaces, control characters, or non-ASCII characters. Characters that are illegal in URLs are encoded as '%' followed by the corresponding hexadecimal character code, e.g. space = %20.

dataStr should be a quoted literal string or a reference to a string value.

values([scope,]attributeNameStr)

Operator Type: Function [other Function type actions]
Operator Scope of Action: Group [operators of similar scope]
Operator Purpose: Dictionary, Set & List operations [other Dictionary, Set & List operations operators]
Operator First Added: Baseline
Operator Last Altered: 9.5.2
Operator Uses Scoped Arguments: [More on scoped arguments in Action Code]
Operator Has Optional Arguments: [More on optional operator arguments]

values([scope,]attributeNameStr)

values(attributeNameStr)

This returns a Set of unique values for the attribute **attributeNameStr**. As the output is a of Set data type, the list of possible values is de-duped thus giving a list of unique values, and should be sorted. Any suggested values not actually us by at least one note are omitted from the list return by values().

If the named **attributeNameStr** is a set or list, values() returns a list of all the unique discrete list item values that occur. If the named attribute is a string, values() returns all the unique strings.

From v9.5.2, **attributeNameStr** is evaluated in case it is something like a variable holding the desired target attribute name. The change is that, if the argument is not an attribute name, and if the argument trimmed of its $ is still not an attribute name, we now see if the evaluated expression turns out to be an attribute name.

Important: unlike other operators, be aware that values($MyString) returns a list of values *found for $MyString, not a list of the values of the attribute whose name is stored in $MyString*. This is unique to values(), and arises because writing values($MyString) instead of values("MyString") was a very common mistake. If needing to pass a value holding an attribute name, use an action code variable and not as a value stored in another attribute.

The returned set is sorted in lexical order.

If a document has a user List-Type attribute $SomeList, then to get a a list of all the unique values for $SomeList in the whole document:

```
$MyList = values("SomeList");
```

The **attributeNameStr** argument is evaluated so may be:

- A quoted attribute name without $-prefix: `values("MyList")`.
- A string attribute name, whose current value is the name of an attribute (without a dollar prefix). If $MyString has the value "MyList", then `values($MyString)` will evaluate the unique values for $MyList (and output the same result as `values("MyList")` above).
- An expression resolving to an attribute name.

Thus if the document has a user List-Type attribute $SomeList and $MyString has the value "SomeList", then these are functionally equivalent:

```
$MyList = values("SomeList");
$MyList = values($MyString);
```

It is envisaged that the first method (the quoted, un-prefixed, attribute name) will be the most usual method of using values().

Sorting. The data is returned in case-sensitive lexical sort order (i.e. all capitals sort before lower case letters, and numbers sorting textually not numerically) so chaining .isort() may often be the desired 'default', or use .nsort() if the list is entirely composed of numbers. Assume, the intention is to get a note whose $Text has one value per line. It could be coded thus:

```
$Text=values("MyList").format("\n");
```

This is a very useful way of making a set of per-value notes. Use values() to collect the values, pass them to a note's $Text as a one-value-per-line string and then explode the $Text.

However, a value list of `[aardvark;amber;Ant]` would actually list in this order: `[Ant;aardvark;amber]`. That is due the Set's auto-sort using case-sensitive lexical sorting, whereas a case-insensitive sort would be more appropriate. Thus:

```
$Text=values("MyList").isort.format("\n");
```

Similarly the default sort would order 1/2/10 and 1/10/2, so a numerical sort would be more sensible:

```
$Text=values("MyList").isort.format("\n");
```

De-duplication. values() differs from collect() in that values() returns Set-type data and collect() returns List-type data. For a list $MyList, the following are functionally equivalent in output:

```
$MyList = collect(all,$SomeList).unique;
$MyList = values("SomeList");
$MyList = values($MyString);
```

(where $MyString has the value "SomeList")

values(scope, attributeNameStr)

If an optional first **scope** argument is provided, the value(s) returned are drawn only from notes in that **scope** (defining **scope**). If no first argument is supplied, as in the short form above, the default **scope**-defined group is assumed as 'all' a thus at whole document scope.The reference point for groups like 'children' or 'siblings' is *this* note. Thus:

```
$MyList = values(children,"Subtitle");
```

will return all the discrete subtitles (i.e. values of $Subtitle) for children of this note, i.e. the note in which context the action is being evaluated.

var

Operator Type:	Statement [other Statement type actions]
Operator Scope of Action:	Item [operators of similar scope]
Operator Purpose:	Data manipulation [other Data manipulation operators]
Operator First Added:	Baseline
Operator Last Altered:	As at baseline

---

var

var:type

The **var** statement declares the local variable. No trailing parentheses are used. Occasionally, an action may find it convenient to declare a local variable in which to save intermediate results. In the past, the only choice was to use a user attribute as a temporary store, with the attendant issues of deciding if it is necessary to reset the attribute at the end of the expression.

Depending on how the operator is being used it may be used without parentheses.

Try to use variable names that will not be mistaken for something else. Avoid using a 'var' prefix for variable names, such as 'varX', as this will confuse Tinderbox's parser. In such a case 'vX' might be a better choice and a 'v' prefix is the convention adopted in aTbRef (though there is *no requirement* to use such naming as it is used for explanatory reasons only).

A local variable acts in most ways like a user attribute. Local variables exist for the duration of the action or, when they are declared inside curly brackets { … }, e.g. an an . each() loop, their scope is the rest of the clause, i.e. the remaining individual statements within the {}.

Declaring a variable

So, the **var** statement declares the local variable:

`var x;` declares a variable 'x', but with no value

More sensible is to use a name indicating purpose:

`var vNum;` declares a variable 'vNum' with no value.

`var vNum(5);` declares a variable 'vNum' *and* gives it an initial value of 5.

`var vText("this is note "+$Name);` declares a variable 'vText' and gives it a calculated string value.

A variable can also be declared and assigned a value is a single expression:

```
var vCost = 5;
```

… defines a temporary variable 'vCost' and then assigns it the value '5'. It is equivalent to the following original syntax, note the explicit use of operator argument parentheses in this case:

```
var vCost(5);
```

In effect, the first form collapses the older two-step method of declaring an empty **var** *before* then giving it a value:

```
var vCost;
vCost=5;
```

Local variables *must* be declared before first use. If not explicitly initialised, their initial value is the empty string "". Within scope, a variable maybe reset in the same manner as an attribute. Thus, `YY=;` resets variable 'YY' to no value.

Setting a variable's data type

From v9.1.0, vars can optionally be given a data type on creation, by colon-appending the data-type to the var operator. Thus:

```
var:number vCost(5);
```

This generates a variable 'vCost' of (expected) data type Number and sets an initial value of '5'.

Available data types include

- `boolean`
- `color`
- `date`
- `dictionary`
- `interval`
- `number`
- `list`
- `set`
- `string`

Note that data type labels are *all-lowercase*. Other existing attribute data types that are not listed here will be provisioned as a `string`, as they are strings with a special, contextually different, form of use. **Important**: defining a date type without an initial value does not set that types value:

`var:number vNum;` → values is undefined (i.e. "")

`var:number vNum = 0;` → values is zero (`0`)

Providing an explicit type helps Tinderbox to understand the user's intent, especially if a default value is applied (though for string-based attributes it is moot). For example, the same numerical value number might react to subsequent operations in various ways:

`var:number vCost(5); vCost = vCost + 5; $MyString= vCost;` gives '10'

`var:list vCost((5); vCost =vCost + 5; $MyString = vCost;` gives a 2-member list, '5;5'

`var:string x(5); vCost =vCost + 5; $MyString = vCost;` gives '55'

Further usage Examples

```
$MyNumber = 0; var vNum = 2; $MyNumber = vNum;
$MyNumber = 0; var vNum; vNum = 2; $MyNumber = vNum;
$MyNumber = 0; var vNum(2); $MyNumber = vNum;
```

In all cases $MyNumber is set to 2. In the second example the variable is defined and given a value using a single code expression. Here is a scoped example within a single action the { and } marking the scope:

```
$MyNumber = 0;
{
 var vNum = 2;
 $MyNumber = vNum;
 vNum = 6;
};
$MyNumber = vNum;
```

$MyNumber is set to 0 (nothing) as 'vNum' has no meaning outside the code inside the braces {}. By analogy a variable created *inside* a loop or function, (i.e. within { } sections of code) cannot be read outside that scope, but variables create *outside* such scope can be read/set within a loop or function.

Using variable values declared outside loops or functions

A variable declared using var may be altered from within the scope of a List/Set.each() loop or a function. This includes nested loops or functions. The reverse does not hold. In-loop a new value would be to be stored in an attribute or variabl declared outside the loop. In a function, this value could be (part of) the return data.

Passing variables into export code

^value()^ can be used, *within the context a single template* , to insert a var() variable declared within an ^action()^ code. Note that the variable must be declared before use, i.e. before as in reading template code top to bottom. **Beware nam collisions** Assume for a moment you have an actual attribute $vNum, then:

```
$MyNumber = 0;
{
 var vNum(2);
 $MyNumber = vNum;
 vNum = 6;
```

```
};
vNum = 4;
$MyNumber = vNum;
```

$MyNumber is 4, rather than 0, because the `vNum = 4` is interpreted as the deprecated legacy syntax of `$vNum = 4`. So, be careful that a local variable's name does not match existing attributes, note names or string literals that you use in your project. Remember the local variable does **not** take a $ prefix. It *is* a value reference but *not* to an attribute.

version()

Operator Type: Property [other Property type actions]
Operator Scope of Action: Document [operators of similar scope]
Operator Purpose: Data manipulation [other Data manipulation operators]
Operator First Added: Baseline
Operator Last Altered: As at baseline

Syntax note: *Operators without any defined mandatory arguments may omit their empty closing parentheses*

---

version()

version

The action operator **version()** returns information about the version of Tinderbox. With no argument, is return a complete version string such as "8.7.2b472".

version("part") An optional single argument can be supplied. Possible values are: **major**, **minor**, **fix**, and **build**, each returning a specific part of the overall version.

For example, in Tinderbox 8.7.2b472:

version() returns: `8.7.2b472`

version(major) returns: `8`

version(minor) returns: `7`

version(fix) returns: `2`

version(build) returns: `b472`

These arguments may be used with or without enclosing quotes. Thus, usage:

```
$MyString = version(major);
```

or

```
$MyString = version("fix");
```

weeks(startDate, endDate)

Operator Type: Function [other Function type actions]
Operator Scope of Action: Item [operators of similar scope]
Operator Purpose: Date-time [other Date-time operators]
Operator First Added: Baseline
Operator Last Altered: As at baseline

---

weeks(startDate, endDate)

This returns the Number of (7-day weeks) weeks between two dates, **startDate** and **endDate**. If the dates are within seven days of each other, the result is zero.

If $DateA is 3 January 2016 and $DateB is 20 January 2016, then:

```
$MyNumber = weeks($DateA,$DateB);
```

sets $MyNumber to 2.

If **endDate** is before **startDate** the result is negative.

while(condition){}

Operator Type: Operator [other Operator type actions]
Operator Scope of Action: Conditional Group [operators of similar scope]
Operator Purpose: Data manipulation [other Data manipulation operators]
Operator First Added: Baseline
Operator Last Altered: As at baseline
Operator Has Conditional Arguments: [More on conditional operator arguments]

The operator **while(condition){...}** performs an action repeatedly until the **condition** expression evaluates as `false`. Put another way, the code in the curly brackets will continue to be run again and again, until **condition** expression evaluat as `false`. This operator addresses the scenario where:

- it is known a task needs to be done multiple times but,

_ the *exact number of times* to repeat the task is not known

Using **while()** it is possible to define the a test (**condition**) to ascertain when the task is done and the repeating/looping action can stop.

For those less used to coding, consider a generalised analogy—the task of filling a bucket with water filling a bucket. Before adding anything to the bucket we might sensibly ask if the bucket is full and only if it is not, do we add another scoop of water. But if the bucket is full, we stop adding water. It can be summarised thus:

- Is the bucket full?
  - No? We add another scoop of water and start over with the original check (i.e. continue the loop).
  - Yes? We're done—exit and finish.

Examples

Consider the case where it is desired to process a note's $Text, of unknown size, one sentence at a time. Thus (code commenting is for explanation only, and is not needed in actual code):

```
// make a variable holding $Text, so $Text itself is not affected
var:str vText = $Text;
// start the loop using vText
while(vText!=""){
 // get the first sentence in vText
 var:string vStr = $Text.sentence(0);
 // pass that sentence to function fProcessText() to do whatever task with it ...
 fProcessText(vStr);
 // remove the sentence vStr from vText *before* the loop runs again
 // so that vStr is different in the next loop, i.e. each sentence is used only once
 vText = vText.substr(vStr.size).trim();
};
```

This will read $Text and process it to call the user-defined function **fProcessText()** passing it one sentence at a time. Each loop removes the just-processed sentence, so the text considered in the next loop is one sentence shorter, until eventually vText is empty at which point the while() operator completes and the next action after it, if any, is read.

A programmer might write the same as above more tersely:

```
var vt=$Text;while(vt){var:string s=vt.sentence;vt=vt.substr(s.size).trim(); process(s);}
```

But, importantly, both do the same thing and the different code has no effect on Tinderbox performance. As long as the code is valid, users are free to choose their style.

Prior to this, it would be necessary to first count the number of discrete sentences and store that in a variable, then make a list of discrete sentences, then iterate this list using a loop counter and checking in each loop to see if the counter figure was below the stored count before taking any per-item action; while() wraps all that up into a simple operator.

Preventing against infinite loops

An infinite loop occurs when **condition** always remains `false` and thus the code loop never stops running. To guard against this unintended scenario **while()** loops are limited to 10,000 iterations, i.e. if the loop has run 10,000 time it automatically stops regardless of the state of **condition**.

word(dataStr)

Operator Type: Function [other Function type actions]
Operator Scope of Action: Item [operators of similar scope]
Operator Purpose: Query Boolean [other Query Boolean operators]
Operator First Added: Baseline
Operator Last Altered: As at baseline

---

word(dataStr)

Returns Boolean `true` if the string **dataStr** is contained in the current note's text. This operator is quicker for matching whole words than using a regular expression. The function matches all notes that contain a single word matching (case-insensitively) any complete word in:

- the title OR
- the text OR
- any user String data-type attribute(s). This does not include other data types based on strings (List, URL, Set, File, Action, etc.)

Note that word() is stricter in its matching than the .contains() function as it looks only for entire words and *does not match regular expressions*. Although word() examines several different fields, the result is that word() is much faster than .contains().

The word **dataStr** must be a quoted string containing:

- a single word (no hyphenated words)
- contain only upper or lower case letters, so no white space, digits, or punctuation
- 4 characters or more (values less than 4 characters or fewer are ignored by Tinderbox).

In the query creation pop-ups of agent and Find dialogs this function is listed as "contains word".

Legacy issues

This operator replaces the legacy #word query operator.

wordsRelatedTo(dataStr[, wordsNum])

Operator Type: Function [other Function type actions]
Operator Scope of Action: Item [operators of similar scope]
Operator Purpose: Data manipulation [other Data manipulation operators]
Operator First Added: Baseline
Operator Last Altered: As at baseline
Operator Has Optional Arguments: [More on optional operator arguments]

---

wordsRelatedTo(dataStr[, wordsNum])

NOTE: This feature needs macOS 10.15.0 or later. If used on older OSs, the function returns no data; i.e. not all users of v8.x maybe able to use this feature as it is macOS version dependent.

The operator tries to return a list (Set) of up to **wordsNum** (discrete) words related to its **dataStr** argument. Fewer words may be returned, or no words may be returned if the argument is unusual, or if the current language is not supported.

An optional second argument **wordsNum**, allows the returned number of related terms to be limited to a specific number of items. **wordsNum** is 1 or more (integer value), but Tinderbox will only return as many 'related' terms as it can find, i it may return *fewer* than **wordsNum** items.

Practically, an **wordsNum** value of more than 10 items is unlikely to render useful results. Be aware this draws on features still under development in the underlying OS, so there are no certainties.

```
$MySet = wordsRelatedTo("hubris");
```

or to get only 5 (or fewer) values:

```
$MySet = wordsRelatedTo("aspirational",5);
```

It is not certain, but likely, that results returned are are ordered by proximity of the relationship in the vector space of the **dataStr** (bear in mind this is a black-box OS procedure).

Though the product is essentially a Set, i.e. a list without duplicates, Sets *may* change the order of items during other processes whilst List attributes do not. So if concerned about retaining order exactly as originally returned by the function, i makes would pass the process to an explicit list

```
$MyList = wordsRelatedTo("expectation",8);
```

XML.each(pathStr){action}

Operator Type: Function [other Function type actions]
Operator Scope of Action: Item [operators of similar scope]
Operator Purpose: Stream parsing [other Stream parsing operators]
Operator First Added: Baseline
Operator Last Altered: As at baseline

XML.each(pathStr){ action(s) }

This operator locates the **xml** object for each object at the **pathStr** (within the XML), generating a loop upon it. The **.each(pathStr)** invokes the **action** block with the xml item bound in turn to each book element. On completion, it restores the XML object to its previous state.

Consider a note "Source note" with this $Text:

```
<shelf>
 <book price="9.95">War and Peace</book>
 <book price="4.95">No et Moi</book>
 <audio price="14.95">Born To Run</audio>
</shelf>
```

Example usage of **path**:

```
$MyList("test") = "";
$Text("Source note").xml.each("/shelf/book"){
 $MyList("test2")+=xml["book"]+"\n";
};
```

which sets $MyList of note "test" to a list of children of the `<shelf>` element that are `<book>` elements, [War and Peace;No et Moi].

```
$MyList("test2") = "";
$Text("Source note").xml.each("/shelf/book"){
 $MyList("test2")+=xml["book[2]"]+"\n";
};
```

which sets $MyList of note "test" to the value of child element of the `<shelf>` element that represents the second book, "No et Moi". Other iterated elements return a blank. *See 'Notes' below*.

```
$MyList("test2") = "";
$Text("Source note").xml.each("/shelf/book"){
 $MyList("test2")+=xml["book@price"]+"\n";
};
```

witch sets $MyList of note "test" to a list of the `price` attribute of each book element, [9.95;4.95].

Notes:

- Acceptable **path** content is a subset of the XPath standard (see https://www.w3.org/TR/2017/REC-xpath-31-20170321/).
- To accord with standard XML and XPath usage, the first child of an XML node is child [1], not [0].
- XML attributes are not related to Tinderbox attributes.

Consider this source XML string stream:

```
<shelf>
<book price="9.95">War and Peace</book>
<book price="4.95">No et Moi</book>
<audio price="14.95">Born To Run</book>
</shelf>
```

Examples

```
Stream.xml.each("/shelf/book") {action}
```

returns every 'book' object in the 'shelf' object and iterates over them, i.e. for each (**path**-matched) item it invokes the **action** block with the xml item bound in turn to each book element. In the example above this means the action is run 3 times, once for each of the 3 `<book>` elements under `<shelf>`.

Inside the **action** clause, action code can refer to the value of the iterated item as elementName[], `book[]`. Any attribute of that item can be addressed as elementName[@attribute] `book[@price]`, i.e. here the 'price' attribute of the XML object in focus.

On completion, it restores the **xml** object to its previous state.

XML.xml(pathStr)

Operator Type: Function [other Function type actions]
Operator Scope of Action: Item [operators of similar scope]
Operator Purpose: Stream parsing [other Stream parsing operators]
Operator First Added: Baseline
Operator Last Altered: As at baseline

XML.xml(pathStr)

If there is no current XML object, attempts to parse the string as XML and fails if unsuccessful. If there is a current XML object, it will be reused.

Returns a specific piece of data from the XML object, determined by the **pathStrathStr** should be an expression or a quoted string.

Consider a note "Source note" with this $Text:

```
<shelf>
 <book price="9.95">War and Peace</book>
 <book price="4.95">No et Moi</book>
 <audio price="14.95">Born To Run</audio>
</shelf>
```

Example usage of **path**:

```
$Text("test") = $Text("Source note").xml("/shelf/book");
```

sets $Text of note "test" to a list of children of the `<shelf>` element that are `<book>` elements. "War and Peace;No et Moi"

```
$Text("test") = $Text("Source note").$Text.xml("/shelf/book[2]")
```

sets $Text of note "test" to the value of child element of the `<shelf>` element that represents the second book, "No et Moi". *See 'Notes' below*.

```
$Text("test") = $Text("Source note").$Text.xml("/shelf/book@price")
```

sets $Text of note "test" to the `price` attribute of each book, "9.954.95"

Notes:

- Acceptable **path** content is a subset of the XPath standard (see https://www.w3.org/TR/2017/REC-xpath-31-20170321/).
- To accord with standard XML and XPath usage, the first child of an XML node is child [1], not [0].
- XML attributes are not related to Tinderbox attributes.

year(aDate[, yearsNum])

Operator Type: Function [other Function type actions]
Operator Scope of Action: Item [operators of similar scope]
Operator Purpose: Date-time [other Date-time operators]
Operator First Added: Baseline
Operator Last Altered: As at baseline
Operator Has Optional Arguments: [More on optional operator arguments]
Operator Has Newer Dot-Operator Variant: Yes

---

year(aDate[, yearsNum])

Alternatively, use Date.year.

year(aDate)

returns the year from the **theDate** date/time expression:

```
$MyNumber = year($MyDate);
```

year(aDate,value)

creates a new date based on the **theDate** date/time expression, but in which the year is **yearsNumDate** is not changed unless theDate is an attribute and the attribute is re-setting itself:

`$MyDateA = year($MyDate,2011);` $MyDate is unaltered

`$MyDate = year($MyDate,2011);` $MyDate is changed

Examples. If $MyDate is July 4,2009, then

```
$MyDate=hour($MyDate,2011);
```

will change $MyDate to July 4, 2011.

years(startDate, endDate)

Operator Type: Function [other Function type actions]
Operator Scope of Action: Item [operators of similar scope]
Operator Purpose: Date-time [other Date-time operators]
Operator First Added: Baseline
Operator Last Altered: As at baseline

---

years(startDate, endDate)

returns the Number of whole years that elapsed between **startDate** and **endDate**. If **endDate** is earlier than **startDate** then the result is negative.

If $DateA is 3 January 2016 and $DateB is 9 January 2018, then:

```
$MyNumber = years($DateA,$DateB);
```

sets $MyNumber to 2.

Action Operator Types

Here the actions operators are listed by functional type though these are not values set by Tinderbox, simply groupings of similar action codes:

- Assignment operators
- Color operators
- Composite operators
- Data manipulation operators
- Date-time operators
- Dictionary, Set & List operations operators
- Document configuration operators
- Formatting operators
- Linking operators
- Mathematical operators
- Non-query Boolean operators
- Query Boolean operators
- Stream parsing operators

Assignment operators

These operators the value of the right side of action codes be applied to the left side and are the means by which attribute values are altered via code (as opposed to manual edit). They include:

- &= (i.e. logical AND assignment)
- = (i.e. value assignment)
- |= (i.e. logical OR assignment)
- $AttributeName[(scope)]

For example

```
$Color = "bright red"
$Color = $AccentColor
$ResultText = $Label + " copy"
```

Color operators

These operators let rules and actions manipulate Color-type attribute values. It can often convenient to tie the colour of a note to its properties. For example, you can make overdue notes red, with a brightness proportional to how overdue they are.

Note that when setting a shade of a named colour in a string the shade value (lighter, darkest, etc.) comes before the colour, unlike listings in Get Info's attributes tab and the Inspector (e.g. Appearance Inspector, Interior sub-tab); thus "dark warm gray" not "warm gray dark", "light cyan" not "cyan light".

- Color.blue()
- Color.brightness()
- Color.green()
- Color.hue()
- Color.red()
- Color.saturation()
- rgb(redNum, greenNum, blueNum)

For example:

```
rgb($MyRed,255,255)
$Color = brightness(hue(RGB(255,0,0),120),75)
```

Composite operators

This is a family of operators that help with use of composites:

- compositeFor(nameStr)
- compositeFor(nameStr):count
- compositeFor(nameStr):kind
- compositeFor(nameStr):name
- compositeFor(nameStr):role(roleStr)
- compositeFor(nameStr):roles
- compositeWithName(compositeNameStr)

Data manipulation operators

This group of operators perform utility functions enabling manipulation of text and values:

- %matches (query back-references)
- $N (query back-reference)
- action([scope,]codeStr)
- changed([scope])
- count_if(scope, condition)
- count(scope)
- covid([stateStr, countryStr|zipCodeStr], aDate, keywordStr)
- delete(scope)
- distance(startItem, endItem)
- distanceTo(startItem, endItem)
- do(macroStr[,argumentsList])
- drivingTimeTo(item)
- eval([item], expressionStr)
- firstWord(dataStr)
- function
- hasLocalValue(attributeNameStr[, item])
- if(condition){actions}[else{actions}]
- isbn10(dataStr)
- isbn13(dataStr)
- lastWord(dataStr)
- List.isort([attributeRefStr])
- List.nsort([attributeRefStr])
- List.select()
- List.sort([attributeRefStr])
- List.unique()
- list(expressionList)
- List/Set.asString()
- List/Set.at(itemNum)
- List/Set.count_if(loopVar, condition)
- List/Set.count()
- List/Set.countOccurrencesOf(literalStr)
- List/Set.each(loopVar){actions}
- List/Set.intersect(aSet)
- List/Set.max()
- List/Set.min()
- List/Set.remove(matchValue)
- List/Set.replace(regexMatchStr, replacementStr)
- List/Set.reverse()
- List/Set.size()
- List/Set.tr(inStr, outStr)
- neighbors(scope, distanceNum[, linkTypeStr])
- neighbors2(scope, distanceNum[, linkTypeStr])
- neighbors2Within(scope, distanceNum[, linkTypeStr])
- neighborsWithin(scope, distanceNum[, linkTypeStr])
- notify(headlineStr[, detailsStr, deliveryDateTime])
- play(soundNameStr)
- return
- runCommand(commandStr[, inputsStr, dirStr])
- select()
- select(scope)
- show(msgString[, backgroundColor[,colorString]])
- stamp([scope,]stampName)
- String.beginsWith(matchStr)
- String.countOccurrencesOf(literalStr)
- String.deleteCharacters(characterSet)
- String.endsWith(matchStr)
- String.extract(regexStr[, caseInsensitiveBln])
- String.extractAll(regexStr[, caseInsensitiveBln])
- String.find(matchStr)
- String.following(matchStr)
- String.highlights([aColor])
- String.next()
- String.nounList()
- String.paragraph(paraNum)
- String.paragraphCount()
- String.paragraphList()
- String.paragraphs(parasNum)
- String.replace(regexMatchStr, replacementStr)
- String.reverse()
- String.sentence([sentenceNum])
- String.show([backgroundColor[,colorString]])
- String.size()
- String.speak([voiceNameStr])
- String.split(regexStr)
- String.substr(startNum[, lengthNum])
- String.toNumber()
- String.tr(inStr[, outStr])
- String.trim([filterStr])
- String.wordCount()
- String.wordList()
- String.words(wordsNum)
- StyledString.textColor(aColor)
- substr(dataStr, startNum[, lengthNum])
- twitter(usernameStr, statusStr)
- update(scope)
- var
- version()
- while(condition){}
- wordsRelatedTo(dataStr[, wordsNum])

Date-time operators

Date accessors and manipulator operators are available for rules and actions:

- Date.day()
- Date.hour()
- Date.minute()
- Date.month()
- Date.second()
- Date.week()
- Date.weekday()
- Date.year()
- date(dateStr)
- date(yearNum, monthNum, dayNum[, hourNum, minNum])
- day(aDate[, dayNum])
- days(firstDate, lastDate)
- hour(aDate[, hoursNum])
- hours(startDate, endDate)
- Interval.day()
- Interval.hour()
- Interval.minute()
- Interval.second()
- interval(dataStr)
- interval(startDate, endDate)
- locale([localeCodeStr])
- minute(aDate[, minutesNum])
- minutes(startDate, endDate)
- month(aDate[, monthsNum])
- months(startDate, endDate)
- seconds(startDate, endDate)
- time(aDate, hoursNum, minutesNum, secondsNum)
- time(aDate)
- weeks(startDate, endDate)
- year(aDate[, yearsNum])
- years(startDate, endDate)

The action code method to create a date string is to call the various date-related operators.

Date examples:

```
date(2004,7,23,16,45)
```
…is 23 July 2004 16:45

If $Date is July 4, 2009 then

```
day($Date)
```
…is 4

If $Date is July 4,2009, then

```
$Date=day($Date,5)
```
will change $Date to July 5, 2009.

If $Date is July 4, 2009 then

```
month($Date)
```
…is 7

If $Date is July 4,2009, then

```
$Date=month($Date,5)
```
will change $Date to May 4, 2009.

Note that using `format()` or `.format()` with Date or Interval data generates a *String*. Interval data is also expressed as a String. so if needing a duration in days or or mitures, consider the suite of difference measuring operators: years(), months(), weeks(), days(), hours(), minutes(), seconds(). Each of these takes two Date objects and returns the difference in the type of unit in the operator name. So hours() gives the Number of hours. The number is rounded *down* to give the count of whole hours, days—i.e. the calling time measure).

To get a similar unit-based measure of an Interval, use date("now") for the Date inputs but add the interval to one of them. the difference in the dates is now the interval and this can be tested with the above method. If $MyInterval is "`-05:`
and $MyDate is "`24/10/2022 12:00:00`" (midday on 24 October 2022):

```
$MyDate = $MyDate + $MyInterval;
```
results in $MyDate being `24/10/2022 11:54:30`.

```
$MyNumberOfMinutes = minutes($MyDate,$MyDate+$MyInterval);
```
By accepting negative intervals the same action code can deal with positive and negative durations. A more generalised approach to the last:

```
$MyNumberOfMinutes = minutes(date("now"),date("now")+$MyInterval);
```
Here minutes are used but the method could test days, hours or seconds (not bigger units as Interval data us usually a few days at biggest and generally less than a whole day).

Dictionary, Set & List operations operators

This is a family of operators that help create sets or lists:

- collect_if(scope, condition, expressionStr)
- collect(scope, expressionStr)
- Dictionary.add(itemDict)
- Dictionary.count()
- Dictionary.empty()
- Dictionary.extend(itemDict)
- Dictionary.keys()
- Dictionary.size()
- dictionary(dictionaryStr)
- Dictionary[keyStr]
- find(scope)
- links[(scope)].[directionStr].[linkTypeRegex].attributeNameRefStr
- List/Set.avg()
- List/Set.collect_if(loopVar, condition, expressionStr)
- List/Set.collect(loopVar, expressionStr)
- List/Set.first()
- List/Set.last()
- List/Set.lookup(keyStr)
- List/Set.randomItem()
- List/Set[itemNum]
- values([scope,]attributeNameStr)

Document configuration operators

This is a family of operators that help create or recover information about the current document's structure:

- attribute(attributeNameStr).keys
- attribute(attributeNameStr)[keyStr]
- create([containerStr,]nameStr)
- createAdornment([containerStr,] nameStr)
- createAgent([containerStr,]nameStr)
- createAttribute(nameStr[, dataType])
- document.keys()
- document()
- document[keyStr]
- fetch(urlStr,headersDict,attrNameStr,cmdStr[,httpMethod])
- require(featureName)
- type(attributeNameStr)

Formatting operators

This group of operators are primarily concerned with (re-)formatting values:

- attributeEncode(dataStr)
- capitalize(dataStr)
- Color.format()
- Date.format(formatStr)
- escapeHTML(dataStr)
- exportedString(item[, templateStr])
- format(dataStr, formatStr[, additionalArguments])
- idEncode(dataStr)
- Interval.format(formatStr)
- jsonEncode(dataStr)
- List/Set.format(formatStr)
- lowercase(dataStr)
- Number.format(decimalsNum[, widthNum, padStr]|formatStr)
- Number.precision(decimalsNum)
- String.capitalize()
- String.json()
- String.jsonEncode()
- String.lowercase()
- String.uppercase()
- StyledString.bold()
- StyledString.fontSize(pointSizeNum)
- StyledString.italic()
- StyledString.plain()
- StyledString.strike()
- uppercase(dataStr)
- urlEncode(dataStr)

These new functions will be useful in preparing data for use with other programs and web services via runCommand().

Linking operators

This group of operators create or remove internal Tinderbox links:

- createLink(sourceItem, destinationItem[, linkTypeStr])
- eachLink(loopVar[,scope]){actions}
- linkFrom(scope[, linkTypeStr])
- linkFromOriginal(scope[, linkTypeStr])
- linkPath(pathNameStr[, startStr, endStr])
- linkTo(scope[, linkTypeStr])
- linkToOriginal(scope[, linkTypeStr])
- unlinkFrom(scope[, linkTypeStr])
- unlinkFromOriginal(scope[, linkTypeStr])
- unlinkTo(scope[, linkTypeStr])
- unlinkToOriginal(scope[, linkTypeStr])

Mathematical operators

Actions and rules may use certain numeric operators. In all cases the argument may be a number or an expression that evaluates as a number.

- - (i.e. subtraction)
- -= (i.e. decrement)
- ... (i.e. range)
- * (i.e. multiplication)
- / (i.e. division)
- + (i.e. addition)
- + (i.e. string concatenation)
- += (i.e. increment)
- abs(sourceNum)
- atan(radiansNum)
- avg_if(scope, condition, expressionStr)
- avg(scope, expressionStr)
- ceil(sourceNum)
- cos(radiansNum)
- degrees(radiansNum)
- exp(powerNum)
- floor(sourceNum)
- List/Set.sum_if(loopVar, condition[, expressionStr])
- List/Set.sum()
- log(sourceNum)
- max(numberList)
- min(numberList)
- mod(sourceNum, modulusNum)
- Number.ceil()
- Number.floor()
- Number.round()
- pow(sourceNum, powerNum)
- radians(degreesNum)
- rand([maxNumber])
- round(sourceNum)
- sin(sourceNum)
- sqrt(sourceNum)
- sum_if(scope, condition, expressionStr)
- sum(scope, expressionStr)
- tan(radiansNum)

For example:

```
$Width=5+log($WordCount);
$Datum=2*cos(radians($Angle));
```

Comparisons and equality tests are described separately under Basic Comparison Operators.

Non-query Boolean operators

These operators return a Boolean `true/false`:

- any(scope, condition)
- every(scope, condition)
- isDuplicateName(item)
- List/Set.empty()
- String.empty()

Query Boolean operators

These operators return a Boolean `true/false`. This group of operators replace the old #-prefixed special query codes:

- != (i.e. value inequality)
- (!$AttributeName) (i.e. a short form test for no value)
- & (i.e. query logical AND join)
- < (i.e. less than)
- <= (i.e. less than or equal to)
- == (i.e. value equality)
- > (i.e. greater than)
- >= (i.e. greater than or equal to)
- | (i.e. query logical OR join)
- $AttributeName (i.e. a short form test for value)
- between(valueNum, minNum, maxNum)
- contains(item)
- descendedFrom(item)
- Dictionary.contains(keyStr)
- Dictionary.icontains(keyStr)
- first(item[, childrenNum])
- indented(depthNum[, item])
- inheritsFrom([item,]prototypeStr)
- inside(item)
- last(item[, childrenNum])
- linkedFrom(scope[, linkTypeStr])
- linkedTo(scope[, linkTypeStr])
- List/Set.any(loopVar, expressionStr)
- List/Set.contains(matchStr)
- List/Set.every(loopVar, expressionStr)
- List/Set.icontains(matchStr)
- originalLinkedFrom(scope[, linkTypeStr])
- originalLinkedTo(scope[, linkTypeStr])
- similarTo(item[, notesNum])
- String.contains(regexStr)
- String.containsAnyOf(regexList)
- String.icontains(regexStr)
- String.icontainsAnyOf(regexList)
- word(dataStr)

Stream parsing operators

A range of operators used for stream parsing of textual/numerical data—most usually a not's $Text):
Such operators are:

- fail()
- JSON.each([pathStr]){actions}
- JSON.json[itemNum]
- JSON.json[keyStr]
- JSON.jsonValue(pathStr)
- String.captureJSON()
- String.captureLine([targetAttributeStr])
- String.captureNumber([targetAttributeStr])
- String.captureRest([targetAttributeStr])
- String.captureTo(matchStr[, targetAttributeStr])
- String.captureToken([targetAttributeStr])
- String.captureWord([targetAttributeStr])
- String.captureXML()
- String.eachLine(loopVar[:condition]){actions}
- String.expect(matchStr)
- String.expectNumber()
- String.expectWhitespace()
- String.expectWord()
- String.failed()
- String.skip(charsNum)
- String.skipLine()
- String.skipTo(matchStr)
- String.skipToNumber()
- String.skipWhitespace()
- String.try{actions}[.thenTry{actions}]
- XML.each(pathStr){action}
- XML.xml(pathStr)

Action Operator Scope

Action codes can also be grouped via their scope of action:

- Composite-based operators
- Conditional Group-based operators
- Document-based operators
- Group-based operators
- Item-based operators
- Link type honouring operators
- List-based operators
- Query-based operators

Composite-based operators

These operators use a scope based on a composite. Such operators are:

- compositeFor(nameStr)
- compositeFor(nameStr):count
- compositeFor(nameStr):kind
- compositeFor(nameStr):name
- compositeFor(nameStr):role(roleStr)
- compositeFor(nameStr):roles
- compositeWithName(compositeNameStr)

Conditional Group-based operators

These operators use a scope based on a group. They can work with sets or lists, retaining data type, or may use expressions evaluating as lists or sets.
The group may be any designator from:

- any designator from {children,descendants,sibling,ancestor,all}
- a find()
- a list-based group designator

Such operators are:

- avg_if(scope, condition, expressionStr)
- collect_if(scope, condition, expressionStr)
- if(condition){actions}[else{actions}]
- links[(scope)].[directionStr].[linkTypeRegex].attributeNameRefStr
- sum_if(scope, condition, expressionStr)
- while(condition){}

Document-based operators

These operators use a scope based on the whole document which could be consider 'all' group-scope.
Such operators are:

- require(featureName)
- document[keyStr]
- attribute(attributeNameStr).keys
- document.keys()
- return
- function
- dictionary(dictionaryStr)
- attribute(attributeNameStr)[keyStr]
- document()
- type(attributeNameStr)
- version()
- locale([localeCodeStr])

Group-based operators

These operators use a scope based on a group. They can work with sets or lists, retaining data type, or may use expressions evaluating as lists or sets.
The group may be:

- any designator from {children,descendants,sibling,ancestor,all}
- a find()
- a list-based group designator

Such operators are:

- any(scope, condition)
- avg(scope, expressionStr)
- collect(scope, expressionStr)
- count_if(scope, condition)
- delete(scope)
- every(scope, condition)
- linkFrom(scope[, linkTypeStr])
- linkFromOriginal(scope[, linkTypeStr])
- linkTo(scope[, linkTypeStr])
- linkToOriginal(scope[, linkTypeStr])
- List.select()
- List/Set.count_if(loopVar, condition)
- List/Set.sum_if(loopVar, condition[, expressionStr])
- neighbors(scope, distanceNum[, linkTypeStr])
- neighbors2(scope, distanceNum[, linkTypeStr])
- neighbors2Within(scope, distanceNum[, linkTypeStr])
- neighborsWithin(scope, distanceNum[, linkTypeStr])
- select()
- select(scope)
- stamp([scope,]stampName)
- sum(scope, expressionStr)
- unlinkFrom(scope[, linkTypeStr])
- unlinkFromOriginal(scope[, linkTypeStr])
- unlinkTo(scope[, linkTypeStr])
- unlinkToOriginal(scope[, linkTypeStr])
- values([scope,]attributeNameStr)

Item-based operators

These operators use a scope based on a N-item (note/path) goup or item designator:

- - (i.e. subtraction)
- -= (i.e. decrement)
- ... (i.e. range)
- (!$AttributeName) (i.e. a short form test for no value)
- * (i.e. multiplication)
- / (i.e. division)
- &= (i.e. logical AND assignment)
- + (i.e. addition)
- + (i.e. string concatenation)
- += (i.e. increment)
- = (i.e. value assignment)
- |= (i.e. logical OR assignment)
- $AttributeName (i.e. a short form test for value)
- $AttributeName[(scope)]
- abs(sourceNum)
- action([scope,]codeStr)
- atan(radiansNum)
- attributeEncode(dataStr)
- between(valueNum, minNum, maxNum)
- capitalize(dataStr)
- ceil(sourceNum)
- changed([scope])
- Color.blue()
- Color.brightness()
- Color.format()
- Color.green()
- Color.hue()
- Color.red()
- Color.saturation()
- contains(item)
- cos(radiansNum)
- covid([stateStr, countryStr|zipCodeStr], aDate, keywordStr)
- create([containerStr,]nameStr)
- createAdornment([containerStr,] nameStr)
- createAgent([containerStr,]nameStr)
- createAttribute(nameStr[, dataType])
- createLink(sourceItem, destinationItem[, linkTypeStr])
- Date.day()
- Date.format(formatStr)
- Date.hour()
- Date.minute()
- Date.month()
- Date.second()
- Date.week()
- Date.weekday()
- Date.year()
- date(dateStr)
- date(yearNum, monthNum, dayNum[, hourNum, minNum])
- day(aDate[, dayNum])
- days(firstDate, lastDate)
- degrees(radiansNum)
- descendedFrom(item)
- Dictionary.add(itemDict)
- Dictionary.contains(keyStr)
- Dictionary.count()
- Dictionary.empty()
- Dictionary.extend(itemDict)
- Dictionary.icontains(keyStr)
- Dictionary.keys()
- Dictionary.size()
- Dictionary[keyStr]
- distance(startItem, endItem)
- distanceTo(startItem, endItem)
- do(macroStr[,argumentsList])
- drivingTimeTo(item)

- eachLink(loopVar[,scope]){actions}
- escapeHTML(dataStr)
- eval([item], expressionStr)
- exp(powerNum)
- exportedString(item[, templateStr])
- fail()
- fetch(urlStr,headersDict,attrNameStr,cmdStr[,httpMethod])
- first(item[, childrenNum])
- firstWord(dataStr)
- floor(sourceNum)
- format(dataStr, formatStr[, additionalArguments])
- hasLocalValue(attributeNameStr[, item])
- hour(aDate[, hoursNum])
- hours(startDate, endDate)
- idEncode(dataStr)
- indented(depthNum[, item])
- inheritsFrom([item,]prototypeStr)
- inside(item)
- Interval.day()
- Interval.format(formatStr)
- Interval.hour()
- Interval.minute()
- Interval.second()
- interval(dataStr)
- interval(startDate, endDate)
- isbn10(dataStr)
- isbn13(dataStr)
- isDuplicateName(item)
- JSON.json[itemNum]
- JSON.json[keyStr]
- JSON.jsonValue(pathStr)
- jsonEncode(dataStr)
- last(item[, childrenNum])
- lastWord(dataStr)
- linkedFrom(scope[, linkTypeStr])
- linkedTo(scope[, linkTypeStr])
- linkPath(pathNameStr[, startStr, endStr])
- List.isort([attributeRefStr])
- List.nsort([attributeRefStr])
- List.sort([attributeRefStr])
- List.unique()
- list(expressionList)
- List/Set.asString()
- List/Set.at(itemNum)
- List/Set.contains(matchStr)
- List/Set.each(loopVar){actions}
- List/Set.empty()
- List/Set.format(formatStr)
- List/Set.icontains(matchStr)
- List/Set.replace(regexMatchStr, replacementStr)
- List/Set.reverse()
- List/Set.size()
- List/Set.sum()
- List/Set.tr(inStr, outStr)
- log(sourceNum)
- lowercase(dataStr)
- minute(aDate[, minutesNum])
- minutes(startDate, endDate)
- mod(sourceNum, modulusNum)
- month(aDate[, monthsNum])
- months(startDate, endDate)
- notify(headlineStr[, detailsStr, deliveryDateTime])
- Number.ceil()
- Number.floor()
- Number.format(decimalsNum[, widthNum, padStr]formatStr)
- Number.precision(decimalsNum)
- Number.round()
- originalLinkedFrom(scope[, linkTypeStr])
- originalLinkedTo(scope[, linkTypeStr])
- play(soundNameStr)
- pow(sourceNum, powerNum)
- radians(degreesNum)
- rand([maxNumber])
- rgb(redNum, greenNum, blueNum)
- round(sourceNum)
- runCommand(commandStr[, inputsStr, dirStr])
- seconds(startDate, endDate)
- show(msgString[, backgroundColor[,colorString]])
- similarTo(item[, notesNum])
- sin(sourceNum)
- sqrt(sourceNum)
- String.beginsWith(matchStr)
- String.capitalize()
- String.captureJSON()
- String.captureLine([targetAttributeStr])
- String.captureNumber([targetAttributeStr])
- String.captureRest([targetAttributeStr])
- String.captureTo(matchStr[, targetAttributeStr])
- String.captureToken([targetAttributeStr])
- String.captureWord([targetAttributeStr])
- String.captureXML()
- String.contains(regexStr)
- String.countOccurrencesOf(literalStr)
- String.deleteCharacters(characterSet)
- String.empty()

- String.endsWith(matchStr)
- String.expect(matchStr)
- String.expectNumber()
- String.expectWhitespace()
- String.expectWord()
- String.extract(regexStr[, caseInsensitiveBln])
- String.extractAll(regexStr[, caseInsensitiveBln])
- String.failed()
- String.find(matchStr)
- String.following(matchStr)
- String.highlights([aColor])
- String.icontains(regexStr)
- String.json()
- String.jsonEncode()
- String.lowercase()
- String.next()
- String.nounList()
- String.paragraph(paraNum)
- String.paragraphCount()
- String.paragraphList()
- String.paragraphs(parasNum)
- String.replace(regexMatchStr, replacementStr)
- String.reverse()
- String.sentence([sentenceNum])
- String.show([backgroundColor[,colorString]])
- String.size()
- String.skip(charsNum)
- String.skipLine()
- String.skipTo(matchStr)
- String.skipToNumber()
- String.skipWhitespace()
- String.speak([voiceNameStr])
- String.split(regexStr)
- String.substr(startNum[, lengthNum])
- String.toNumber()
- String.tr(inStr[, outStr])
- String.trim([filterStr])
- String.try{actions}[.thenTry{actions}]
- String.uppercase()
- String.wordCount()
- String.wordList()
- String.words(wordsNum)
- StyledString.bold()
- StyledString.fontSize(pointSizeNum)
- StyledString.italic()
- StyledString.plain()
- StyledString.strike()
- StyledString.textColor(aColor)
- substr(dataStr, startNum[, lengthNum])
- tan(radiansNum)
- time(aDate, hoursNum, minutesNum, secondsNum)
- time(aDate)
- twitter(usernameStr, statusStr)
- uppercase(dataStr)
- urlEncode(dataStr)
- var
- weeks(startDate, endDate)
- word(dataStr)
- wordsRelatedTo(dataStr[, wordsNum])
- XML.each(pathStr){action}
- XML.xml(pathStr)
- year(aDate[, yearsNum])
- years(startDate, endDate)

Link type honouring operators

These operators use a scope that may be modified based on a link type based filter:

- createLink(sourceItem, destinationItem[, linkTypeStr])
- linkedFrom(scope[, linkTypeStr])
- linkedTo(scope[, linkTypeStr])
- linkFrom(scope[, linkTypeStr])
- linkFromOriginal(scope[, linkTypeStr])
- links[(scope)].[directionStr].[linkTypeRegex].attributeNameRefStr
- linkTo(scope[, linkTypeStr])
- linkToOriginal(scope[, linkTypeStr])
- neighbors(scope, distanceNum[, linkTypeStr])
- neighbors2(scope, distanceNum[, linkTypeStr])
- neighbors2Within(scope, distanceNum[, linkTypeStr])
- neighborsWithin(scope, distanceNum[, linkTypeStr])
- originalLinkedFrom(scope[, linkTypeStr])
- originalLinkedTo(scope[, linkTypeStr])
- unlinkFrom(scope[, linkTypeStr])
- unlinkFromOriginal(scope[, linkTypeStr])
- unlinkTo(scope[, linkTypeStr])
- unlinkToOriginal(scope[, linkTypeStr])

List-based operators

A range of operators use a List or Set as their scope. They can work with either attribute type, retaining data type, and may use list-based group designators or find().
The group may be any designator from:

- any designator from {children,descendants,sibling,ancestor,all}
- a find()
- a list-based group designator

Such operators are:

- count(scope)
- JSON.each([pathStr]){actions}
- List/Set.any(loopVar, expressionStr)
- List/Set.avg()
- List/Set.collect_if(loopVar, condition, expressionStr)
- List/Set.collect(loopVar, expressionStr)
- List/Set.count()
- List/Set.countOccurrencesOf(literalStr)
- List/Set.every(loopVar, expressionStr)
- List/Set.first()
- List/Set.intersect(aSet)
- List/Set.last()
- List/Set.lookup(keyStr)
- List/Set.max()
- List/Set.min()
- List/Set.randomItem()
- List/Set.remove(matchValue)
- List/Set[itemNum]
- max(numberList)
- min(numberList)
- String.containsAnyOf(regexList)
- String.eachLine(loopVar[:condition]){actions}
- String.icontainsAnyOf(regexList)
- update(scope)

Query-based operators

A range of operators use a query as their scope, returning a List-type list of (unique) $Path values.
Such operators are:

- %matches (query back-references)
- $N (query back-reference)
- find(scope)
- | (i.e. query logical OR join)
- < (i.e. less than)
- == (i.e. value equality)
- > (i.e. greater than)
- <= (i.e. less than or equal to)
- >= (i.e. greater than or equal to)
- != (i.e. value inequality)
- & (i.e. query logical AND join)

Action Operator Functional Types

Action codes can also be grouped via their functional type:

- Function actions
- Operator actions
- Property actions
- Statement actions

Function actions

These action codes operate like functions. They take one or more inputs and returns an output. Normally the latter is passed to an attribute but in a few cases, e.g. action() or runCommand(), the function may simply be called.
Such operators are:

- abs(sourceNum)
- action([scope,]codeStr)
- any(scope, condition)
- atan(radiansNum)
- attribute(attributeNameStr).keys
- attribute(attributeNameStr)[keyStr]
- attributeEncode(dataStr)
- avg_if(scope, condition, expressionStr)
- avg(scope, expressionStr)
- between(valueNum, minNum, maxNum)
- capitalize(dataStr)
- ceil(sourceNum)
- collect_if(scope, condition, expressionStr)
- collect(scope, expressionStr)
- Color.format()
- compositeFor(nameStr)
- compositeWithName(compositeNameStr)
- contains(item)
- cos(radiansNum)
- count_if(scope, condition)
- count(scope)
- covid([stateStr, countryStr|zipCodeStr], aDate, keywordStr)
- create([containerStr,]nameStr)
- createAdornment([containerStr,] nameStr)
- createAgent([containerStr,]nameStr)
- createAttribute(nameStr[, dataType])
- createLink(sourceItem, destinationItem[, linkTypeStr])
- Date.format(formatStr)
- date(dateStr)
- date(yearNum, monthNum, dayNum[, hourNum, minNum])
- day(aDate[, dayNum])
- days(firstDate, lastDate)
- degrees(radiansNum)
- delete(scope)
- descendedFrom(item)
- Dictionary.add(itemDict)
- Dictionary.contains(keyStr)
- Dictionary.extend(itemDict)
- Dictionary.icontains(keyStr)
- dictionary(dictionaryStr)
- Dictionary[keyStr]

- distance(startItem, endItem)
- distanceTo(startItem, endItem)
- do(macroStr[,argumentsList])
- document.keys()
- document()
- document[keyStr]
- drivingTimeTo(item)
- eachLink(loopVar[,scope]){actions}
- escapeHTML(dataStr)
- eval([item], expressionStr)
- every(scope, condition)
- exp(powerNum)
- exportedString(item[, templateStr])
- fail()
- fetch(urlStr,headersDict,attrNameStr,cmdStr[,httpMethod])
- find(scope)
- first(item[, childrenNum])
- firstWord(dataStr)
- floor(sourceNum)
- format(dataStr, formatStr[, additionalArguments])
- hasLocalValue(attributeNameStr[, item])
- hour(aDate[, hoursNum])
- hours(startDate, endDate)
- idEncode(dataStr)
- if(condition){actions}[else{actions}]
- indented(depthNum[, item])
- inheritsFrom([item,]prototypeStr)
- inside(item)
- Interval.format(formatStr)
- interval(dataStr)
- interval(startDate, endDate)
- isbn10(dataStr)
- isbn13(dataStr)
- JSON.each([pathStr]){actions}
- JSON.json[itemNum]
- JSON.json[keyStr]
- JSON.jsonValue(pathStr)
- jsonEncode(dataStr)
- last(item[, childrenNum])
- lastWord(dataStr)
- linkedFrom(scope[, linkTypeStr])
- linkedTo(scope[, linkTypeStr])
- linkFrom(scope[, linkTypeStr])
- linkFromOriginal(scope[, linkTypeStr])
- linkPath(pathNameStr[, startStr, endStr])
- links[(scope)].[directionStr].[linkTypeRegex].attributeNameRefStr
- linkTo(scope[, linkTypeStr])
- linkToOriginal(scope[, linkTypeStr])
- List.isort([attributeRefStr])
- List.nsort([attributeRefStr])
- List.select()
- List.sort([attributeRefStr])
- List.unique()
- list(expressionList)
- List/Set.any(loopVar, expressionStr)
- List/Set.asString()
- List/Set.at(itemNum)
- List/Set.avg()
- List/Set.collect_if(loopVar, condition, expressionStr)
- List/Set.collect(loopVar, expressionStr)
- List/Set.contains(matchStr)
- List/Set.count_if(loopVar, condition)
- List/Set.countOccurrencesOf(literalStr)
- List/Set.each(loopVar){actions}
- List/Set.every(loopVar, expressionStr)
- List/Set.format(formatStr)
- List/Set.icontains(matchStr)
- List/Set.intersect(aSet)
- List/Set.lookup(keyStr)
- List/Set.randomItem()
- List/Set.remove(matchValue)
- List/Set.replace(regexMatchStr, replacementStr)
- List/Set.reverse()
- List/Set.sum_if(loopVar, condition[, expressionStr])
- List/Set.sum()
- List/Set.tr(inStr, outStr)
- List/Set[itemNum]
- locale([localeCodeStr])
- log(sourceNum)
- lowercase(dataStr)
- max(numberList)
- min(numberList)
- minute(aDate[, minutesNum])
- minutes(startDate, endDate)
- mod(sourceNum, modulusNum)
- month(aDate[, monthsNum])
- months(startDate, endDate)
- neighbors(scope, distanceNum[, linkTypeStr])
- neighbors2(scope, distanceNum[, linkTypeStr])
- neighbors2Within(scope, distanceNum[, linkTypeStr])
- neighborsWithin(scope, distanceNum[, linkTypeStr])
- notify(headlineStr[, detailsStr, deliveryDateTime])
- Number.ceil()
- Number.floor()
- Number.format(decimalsNum[, widthNum, padStr]|formatStr)

- Number.precision(decimalsNum)
- Number.round()
- originalLinkedFrom(scope[, linkTypeStr])
- originalLinkedTo(scope[, linkTypeStr])
- play(soundNameStr)
- pow(sourceNum, powerNum)
- radians(degreesNum)
- rand([maxNumber])
- require(featureName)
- rgb(redNum, greenNum, blueNum)
- round(sourceNum)
- runCommand(commandStr[, inputsStr, dirStr])
- seconds(startDate, endDate)
- select()
- select(scope)
- show(msgString[, backgroundColor[,colorString]])
- similarTo(item[, notesNum])
- sin(sourceNum)
- sqrt(sourceNum)
- stamp([scope,]stampName)
- String.beginsWith(matchStr)
- String.capitalize()
- String.captureJSON()
- String.captureLine([targetAttributeStr])
- String.captureNumber([targetAttributeStr])
- String.captureRest([targetAttributeStr])
- String.captureTo(matchStr[, targetAttributeStr])
- String.captureToken([targetAttributeStr])
- String.captureWord([targetAttributeStr])
- String.captureXML()
- String.contains(regexStr)
- String.containsAnyOf(regexList)
- String.countOccurrencesOf(literalStr)
- String.deleteCharacters(characterSet)
- String.eachLine(loopVar[:condition]){actions}
- String.endsWith(matchStr)
- String.expect(matchStr)
- String.expectNumber()
- String.expectWhitespace()
- String.expectWord()
- String.extract(regexStr[, caseInsensitiveBln])
- String.extractAll(regexStr[, caseInsensitiveBln])
- String.failed()
- String.find(matchStr)
- String.following(matchStr)
- String.icontains(regexStr)
- String.icontainsAnyOf(regexList)
- String.json()
- String.jsonEncode()
- String.lowercase()
- String.next()
- String.paragraph(paraNum)
- String.paragraphs(parasNum)
- String.replace(regexMatchStr, replacementStr)
- String.reverse()
- String.sentence([sentenceNum])
- String.show([backgroundColor[,colorString]])
- String.skip(charsNum)
- String.skipLine()
- String.skipTo(matchStr)
- String.skipToNumber()
- String.skipWhitespace()
- String.speak([voiceNameStr])
- String.split(regexStr)
- String.substr(startNum[, lengthNum])
- String.toNumber()
- String.tr(inStr[, outStr])
- String.trim([filterStr])
- String.try{actions}[.thenTry{actions}]
- String.uppercase()
- String.words(wordsNum)
- StyledString.bold()
- StyledString.fontSize(pointSizeNum)
- StyledString.italic()
- StyledString.plain()
- StyledString.strike()
- StyledString.textColor(aColor)
- substr(dataStr, startNum[, lengthNum])
- sum_if(scope, condition, expressionStr)
- sum(scope, expressionStr)
- tan(radiansNum)
- time(aDate, hoursNum, minutesNum, secondsNum)
- time(aDate)
- twitter(usernameStr, statusStr)
- type(attributeNameStr)
- unlinkFrom(scope[, linkTypeStr])
- unlinkFromOriginal(scope[, linkTypeStr])
- unlinkTo(scope[, linkTypeStr])
- unlinkToOriginal(scope[, linkTypeStr])
- update(scope)
- uppercase(dataStr)
- urlEncode(dataStr)
- values([scope,]attributeNameStr)
- weeks(startDate, endDate)
- word(dataStr)

- wordsRelatedTo(dataStr[, wordsNum])
- XML.each(pathStr){action}
- XML.xml(pathStr)
- year(aDate[, yearsNum])
- years(startDate, endDate)

Operator actions

These action codes are effectively mathematical, logical-test or assignment operators.
Such operators are:

- - (i.e. subtraction)
- -= (i.e. decrement)
- != (i.e. value inequality)
- ... (i.e. range)
- (!$AttributeName) (i.e. a short form test for no value)
- * (i.e. multiplication)
- / (i.e. division)
- & (i.e. query logical AND join)
- &= (i.e. logical AND assignment)
- + (i.e. addition)
- + (i.e. string concatenation)
- += (i.e. increment)
- < (i.e. less than)
- <= (i.e. less than or equal to)
- = (i.e. value assignment)
- == (i.e. value equality)
- > (i.e. greater than)
- >= (i.e. greater than or equal to)
- | (i.e. query logical OR join)
- |= (i.e. logical OR assignment)
- $AttributeName (i.e. a short form test for value)
- changed([scope])
- while(condition){}

Property actions

These actions either set, or fetch (get) the value of an attribute.
Such operators are:

- %matches (query back-references)
- $AttributeName[(scope)]
- $N (query back-reference)
- Color.blue()
- Color.brightness()
- Color.green()
- Color.hue()
- Color.red()
- Color.saturation()
- compositeFor(nameStr):count
- compositeFor(nameStr):kind
- compositeFor(nameStr):name
- compositeFor(nameStr):role(roleStr)
- compositeFor(nameStr):roles
- Date.day()
- Date.hour()
- Date.minute()
- Date.month()
- Date.second()
- Date.week()
- Date.weekday()
- Date.year()
- Dictionary.count()
- Dictionary.empty()
- Dictionary.keys()
- Dictionary.size()
- Interval.day()
- Interval.hour()
- Interval.minute()
- Interval.second()
- isDuplicateName(item)
- List/Set.count()
- List/Set.empty()
- List/Set.first()
- List/Set.last()
- List/Set.max()
- List/Set.min()
- List/Set.size()
- String.empty()
- String.highlights([aColor])
- String.nounList()
- String.paragraphCount()
- String.paragraphList()
- String.size()
- String.wordCount()
- String.wordList()
- version()

Statement actions

These actions are a small sub-set of functions where there are no arguments. Instead the operator acts as a keyword indicating code of a particular purpose will follow.
Such operators are:

- function
- return
- var

Listing of dot-operators

'dot' operators in action code can be recognised because they:

- start with a period character
- are always dot-joined to an attribute or literal value (String or Number)
- apply to a specified range of data types

Be aware that not all dot operators work meaningfully with all data types. For instance, property operators for Color-type data (`Color.red`) will not return data if attached to other data types.

The majority of dot operators relate to use of text-type data, i.e. String, List, or Set. In some cases, whilst the operator applies to more than one attribute data type, it works differently depending on the data type. A prime example is **.forma** which has many data-type dependent behaviours.

As a result where a dot operator behaves differently per chained data type it is documented separately. Thus an operator chained to a particular type, for example **Date.day**, will not return data if chained to other types, *unless documented otherwise*.

An odd exception, that mimics dot operator behaviour but isn't a dot operator is links(). The latter using dot chained arguments, a usage that pre-dates the introduction of the dot operator behaviour.

In aTbRef listings, dot operators will be referred to with the data type before the dot, e.g. Color.blue, so that inline dots in text do not get misread as punctuation. In practice, Color.blue would be used with any Color-type attribute:

```
$MyColor.red = "#ff"
```

A Date.minute would attach to a date attribute:

```
$MyNumber = $MyDate.minute
```

With text-based dot operators a wider range of use is possible:

```
$MyNumber = $MyString.size()
$MyNumber = "Firewood, iron-ware, and cheap tin trays.".size()
$MyNumber = $MyList.size()
$MyNumber = $MySet.size()
$MyNumber = "Euryalus;Galatea;Hermione".size()
```

Multiple dot operators can be ' chained' to connect a series of tasks.

The listing below shows the existing dot operators. As each is listed after the data type(s) with which it may be chained, this means some operators may be listed several times, once for each appropriate data type. The operators are:

- changed([scope])
- Color.blue()
- Color.brightness()
- Color.format()
- Color.green()
- Color.hue()
- Color.red()
- Color.saturation()
- Date.day()
- Date.format(formatStr)
- Date.hour()
- Date.minute()
- Date.month()
- Date.second()
- Date.week()
- Date.weekday()
- Date.year()
- Dictionary.add(itemDict)
- Dictionary.contains(keyStr)
- Dictionary.count()
- Dictionary.empty()
- Dictionary.extend(itemDict)
- Dictionary.icontains(keyStr)
- Dictionary.keys()
- Dictionary.size()
- Interval.day()
- Interval.format(formatStr)
- Interval.hour()
- Interval.minute()
- Interval.second()
- JSON.each([pathStr]){actions}
- JSON.json[itemNum]
- JSON.json[keyStr]
- JSON.jsonValue(pathStr)
- List.isort([attributeRefStr])
- List.nsort([attributeRefStr])
- List.select()
- List.sort([attributeRefStr])
- List.unique()
- List/Set.any(loopVar, expressionStr)
- List/Set.asString()
- List/Set.at(itemNum)
- List/Set.avg()
- List/Set.collect_if(loopVar, condition, expressionStr)
- List/Set.collect(loopVar, expressionStr)
- List/Set.contains(matchStr)
- List/Set.count()
- List/Set.countOccurrencesOf(literalStr)
- List/Set.each(loopVar){actions}
- List/Set.empty()
- List/Set.every(loopVar, expressionStr)
- List/Set.first()
- List/Set.format(formatStr)
- List/Set.icontains(matchStr)
- List/Set.intersect(aSet)
- List/Set.last()
- List/Set.lookup(keyStr)
- List/Set.max()
- List/Set.min()
- List/Set.randomItem()
- List/Set.remove(matchValue)
- List/Set.replace(regexMatchStr, replacementStr)
- List/Set.reverse()
- List/Set.size()
- List/Set.sum()
- List/Set.tr(inStr, outStr)
- Number.ceil()
- Number.floor()
- Number.format(decimalsNum[, widthNum, padStr]|formatStr)
- Number.precision(decimalsNum)

- Number.round()
- String.beginsWith(matchStr)
- String.capitalize()
- String.captureJSON()
- String.captureLine([targetAttributeStr])
- String.captureNumber([targetAttributeStr])
- String.captureRest([targetAttributeStr])
- String.captureTo(matchStr[, targetAttributeStr])
- String.captureToken([targetAttributeStr])
- String.captureWord([targetAttributeStr])
- String.captureXML()
- String.contains(regexStr)
- String.containsAnyOf(regexList)
- String.countOccurrencesOf(literalStr)
- String.deleteCharacters(characterSet)
- String.eachLine(loopVar[:condition]){actions}
- String.empty()
- String.endsWith(matchStr)
- String.expect(matchStr)
- String.expectNumber()
- String.expectWhitespace()
- String.expectWord()
- String.extract(regexStr[, caseInsensitiveBln])
- String.extractAll(regexStr[, caseInsensitiveBln])
- String.failed()
- String.find(matchStr)
- String.following(matchStr)
- String.highlights([aColor])
- String.icontains(regexStr)
- String.icontainsAnyOf(regexList)
- String.json()
- String.jsonEncode()
- String.lowercase()
- String.next()
- String.nounList()
- String.paragraph(paraNum)
- String.paragraphCount()
- String.paragraphList()
- String.paragraphs(parasNum)
- String.replace(regexMatchStr, replacementStr)
- String.reverse()
- String.sentence([sentenceNum])
- String.show([backgroundColor[,colorString]])
- String.size()
- String.skip(charsNum)
- String.skipLine()
- String.skipTo(matchStr)
- String.skipToNumber()
- String.skipWhitespace()
- String.speak([voiceNameStr])
- String.split(regexStr)
- String.substr(startNum[, lengthNum])
- String.toNumber()
- String.tr(inStr[, outStr])
- String.trim([filterStr])
- String.try{actions}[.thenTry{actions}]
- String.uppercase()
- String.wordList()
- String.words(wordsNum)
- StyledString.bold()
- StyledString.fontSize(pointSizeNum)
- StyledString.italic()
- StyledString.plain()
- StyledString.strike()
- StyledString.textColor(aColor)
- XML.each(pathStr){action}
- XML.xml(pathStr)

Listing of non dot-operators with dot-operator versions

The listing below shows the existing *non*-dot operators. Many of these now also have a dot-operator version, which may prove more useful for action code than the original version. The operators are:

- avg(scope, expressionStr)
- capitalize(dataStr)
- ceil(sourceNum)
- collect_if(scope, condition, expressionStr)
- collect(scope, expressionStr)
- contains(item)
- count_if(scope, condition)
- count(scope)
- first(item[, childrenNum])
- floor(sourceNum)
- format(dataStr, formatStr[, additionalArguments])
- hour(aDate[, hoursNum])
- jsonEncode(dataStr)
- last(item[, childrenNum])
- lowercase(dataStr)
- max(numberList)
- min(numberList)
- minute(aDate[, minutesNum])
- month(aDate[, monthsNum])
- round(sourceNum)
- select(scope)
- show(msgString[, backgroundColor[,colorString]])
- substr(dataStr, startNum[, lengthNum])
- sum_if(scope, condition, expressionStr)
- sum(scope, expressionStr)
- uppercase(dataStr)
- year(aDate[, yearsNum])

Listing of operators with link type filters

The listing below shows those operators which are aware of link types:

- createLink(sourceItem, destinationItem[, linkTypeStr])
- linkedFrom(scope[, linkTypeStr])
- linkedTo(scope[, linkTypeStr])
- linkFrom(scope[, linkTypeStr])
- linkFromOriginal(scope[, linkTypeStr])
- links[(scope)].[directionStr].[linkTypeRegex].attributeNameRefStr
- linkTo(scope[, linkTypeStr])
- linkToOriginal(scope[, linkTypeStr])
- neighbors(scope, distanceNum[, linkTypeStr])
- neighbors2(scope, distanceNum[, linkTypeStr])
- neighbors2Within(scope, distanceNum[, linkTypeStr])
- neighborsWithin(scope, distanceNum[, linkTypeStr])
- originalLinkedFrom(scope[, linkTypeStr])
- originalLinkedTo(scope[, linkTypeStr])
- unlinkFrom(scope[, linkTypeStr])
- unlinkFromOriginal(scope[, linkTypeStr])
- unlinkTo(scope[, linkTypeStr])
- unlinkToOriginal(scope[, linkTypeStr])

Listing of operators emitting styled text

The listing below shows those operators that emit styled, not plain text (see explanation):

- StyledString.bold()
- StyledString.fontSize(pointSizeNum)
- StyledString.italic()
- StyledString.plain()
- StyledString.strike()
- StyledString.textColor(aColor)

Listing of operators for Stream Parsing

The listing below shows those operators that can be used in Stream Parsing. Note that whilst most are for use with Strings, a few are specially for use only with strings containing either JSON or XML data (in whole or part).
List of such operators:

- fail()
- JSON.each([pathStr]){actions}
- JSON.json[itemNum]
- JSON.json[keyStr]
- String.captureJSON()
- String.captureLine([targetAttributeStr])
- String.captureNumber([targetAttributeStr])
- String.captureRest([targetAttributeStr])
- String.captureTo(matchStr[, targetAttributeStr])
- String.captureToken([targetAttributeStr])
- String.captureWord([targetAttributeStr])
- String.captureXML()
- String.eachLine(loopVar[:condition]){actions}
- String.expect(matchStr)
- String.expectNumber()
- String.expectWhitespace()
- String.expectWord()
- String.failed()
- String.skip(charsNum)
- String.skipLine()
- String.skipTo(matchStr)
- String.skipToNumber()
- String.skipWhitespace()
- String.try{actions}[.thenTry{actions}]
- XML.each(pathStr){action}
- XML.xml(pathStr)

Listing of operators that can use regular expressions

The listing below shows those operators that are able to use regular expressions ('regex'). Not all require use of regex, but rather regex are allowed in some contexts. For instance, a .replace() operator's **regex** argument that defines what to match could be a literal string such a a particular word. Or, it can be a regex.
For link and unlink commands the regex use only applies to selection of link types. For runCommand() is it relates to parsing of the input string.

For query efficiency in larger documents, it is generally beneficial if a first—or previous—term(s) are used to reduce the number of matches before regex operations are applied. Regex operations are potentially more cpu-intensive task (i.e. tak longer than simpler tasks); very complex regex will also take more time that simple ones. The burden is individually small, but in a large document with many agents running, the sum might become noticeable. By writing efficient queries, e.g. putting regex-based operator as the last term—where the query allows, it is possible to pre-empt any such slow-down occurring.

Regardless, the latter issues applies to large and or heavily queried documents. A new user is very unlikely to need to worry about such issue in their initial exploration of use of agents.

- any(scope, condition)
- collect_if(scope, condition, expressionStr)
- contains(item)
- count_if(scope, condition)
- every(scope, condition)
- if(condition){actions}[else{actions}]
- links[(scope)].[directionStr].[linkTypeRegex].attributeNameRefStr
- List/Set.icontains(matchStr)
- List/Set.replace(regexMatchStr, replacementStr)
- runCommand(commandStr[, inputsStr, dirStr])
- String.captureTo(matchStr[, targetAttributeStr])
- String.contains(regexStr)
- String.containsAnyOf(regexList)
- String.extract(regexStr[, caseInsensitiveBln])
- String.extractAll(regexStr[, caseInsensitiveBln])
- String.icontains(regexStr)
- String.icontainsAnyOf(regexList)
- String.replace(regexMatchStr, replacementStr)
- String.skipTo(matchStr)
- String.split(regexStr)
- sum_if(scope, condition, expressionStr)

Listing of operators with scoping arguments

The listing below shows those operators where one or more arguments are scoping in nature. This means the argument is trying to pass one or more note item(s) using $Name or $Path data (Lees usual, but supported, is $ID)) or a list of items The aim of a scoping argument is to set the scope of the action, i.e. limit as to which note(s) are affected by the action.

Some operators support deeper evaluation than others—think of that in terms of how many nested levels of offset address arguments are included, or the number of implicit nested levels of evaluation. The variation might be from an explicit literal string for $Name or $Path, through data stored in an attribute, data stored in an attribute is a different note, to complex action code expressions. This aspect of operators is not well documented. The more complex the intended argument value, the better it is to do a small rest file first to ensure the argument is evaluated correctly before using the same in a large document.

List of such operators:

- $AttributeName[(scope)]
- any(scope, condition)
- avg_if(scope, condition, expressionStr)
- avg(scope, expressionStr)
- changed([scope])
- collect_if(scope, condition, expressionStr)
- collect(scope, expressionStr)
- count_if(scope, condition)
- count(scope)
- createLink(sourceItem, destinationItem[, linkTypeStr])
- delete(scope)
- eachLink(loopVar[,scope]){actions}
- every(scope, condition)
- find(scope)
- linkedFrom(scope[, linkTypeStr])
- linkedTo(scope[, linkTypeStr])
- linkFrom(scope[, linkTypeStr])
- linkFromOriginal(scope[, linkTypeStr])
- links[(scope)].[directionStr].[linkTypeRegex].attributeNameRefStr
- linkTo(scope[, linkTypeStr])
- linkToOriginal(scope[, linkTypeStr])
- List.select()
- neighbors(scope, distanceNum[, linkTypeStr])
- neighbors2(scope, distanceNum[, linkTypeStr])
- neighbors2Within(scope, distanceNum[, linkTypeStr])
- neighborsWithin(scope, distanceNum[, linkTypeStr])
- originalLinkedFrom(scope[, linkTypeStr])
- originalLinkedTo(scope[, linkTypeStr])
- select()
- select(scope)
- sum_if(scope, condition, expressionStr)
- sum(scope, expressionStr)
- unlinkFrom(scope[, linkTypeStr])
- unlinkFromOriginal(scope[, linkTypeStr])
- unlinkTo(scope[, linkTypeStr])
- unlinkToOriginal(scope[, linkTypeStr])
- update(scope)
- values([scope,]attributeNameStr)

Listing of operators with optional arguments

The listing below shows those operators where one or more arguments are optional. This means the these argument are not *required* for the operator to execute correctly. Without the optional argument(s) the operator will function correctly b will assume some defaults. These defaults may be altered by providing the optional argumet (s).

Some operators support deeper evaluation than others—think of that in terms of how many nested levels of offset address arguments are included, or the number of implicit nested levels of evaluation. The variation might be from an explicit literal string for $Name or $Path, through data stored in an attribute, data stored in an attribute is a different note, to complex action code expressions. This aspect of operators is not well documented. The more complex the intended argument value, the better it is to do a small rest file first to ensure the argument is evaluated correctly before using the same in a large document.

List of such operators:

- action([scope,]codeStr)
- changed([scope])
- covid([stateStr, countryStr|zipCodeStr], aDate, keywordStr)
- create([containerStr,]nameStr)
- createAdornment([containerStr,] nameStr)
- createAgent([containerStr,]nameStr)
- createAttribute(nameStr[, dataType])
- createLink(sourceItem, destinationItem[, linkTypeStr])
- date(yearNum, monthNum, dayNum[, hourNum, minNum])
- day(aDate[, dayNum])
- do(macroStr[,argumentsList])
- document()
- document[keyStr]
- eachLink(loopVar[,scope]){actions}
- eval([item], expressionStr)
- first(item[, childrenNum])
- format(dataStr, formatStr[, additionalArguments])
- hasLocalValue(attributeNameStr[, item])
- hour(aDate[, hoursNum])
- if(condition){actions}[else{actions}]
- indented(depthNum[, item])
- inheritsFrom([item,]prototypeStr)
- last(item[, childrenNum])
- linkedFrom(scope[, linkTypeStr])
- linkedTo(scope[, linkTypeStr])
- linkFrom(scope[, linkTypeStr])
- linkFromOriginal(scope[, linkTypeStr])
- linkPath(pathNameStr[, startStr, endStr])
- links[(scope)].[directionStr].[linkTypeRegex].attributeNameRefStr
- linkTo(scope[, linkTypeStr])
- linkToOriginal(scope[, linkTypeStr])
- List.isort([attributeRefStr])
- List.nsort([attributeRefStr])
- List.sort([attributeRefStr])
- List/Set.sum_if(loopVar, condition[, expressionStr])
- minute(aDate[, minutesNum])
- month(aDate[, monthsNum])
- neighbors(scope, distanceNum[, linkTypeStr])
- neighbors2(scope, distanceNum[, linkTypeStr])
- neighbors2Within(scope, distanceNum[, linkTypeStr])
- neighborsWithin(scope, distanceNum[, linkTypeStr])
- notify(headlineStr[, detailsStr, deliveryDateTime])
- Number.format(decimalsNum[, widthNum, padStr]|formatStr)
- originalLinkedFrom(scope[, linkTypeStr])
- originalLinkedTo(scope[, linkTypeStr])
- rand([maxNumber])
- runCommand(commandStr[, inputsStr, dirStr])
- similarTo(item[, notesNum])
- stamp([scope,]stampName)
- String.captureLine([targetAttributeStr])
- String.captureNumber([targetAttributeStr])
- String.captureRest([targetAttributeStr])
- String.captureTo(matchStr[, targetAttributeStr])
- String.captureToken([targetAttributeStr])
- String.captureWord([targetAttributeStr])
- String.eachLine(loopVar[:condition]){actions}
- String.highlights([aColor])
- String.sentence([sentenceNum])
- String.speak([voiceNameStr])
- String.substr(startNum[, lengthNum])
- String.tr(inStr[, outStr])
- String.trim([filterStr])
- String.try{actions}[.thenTry{actions}]
- substr(dataStr, startNum[, lengthNum])
- unlinkFrom(scope[, linkTypeStr])
- unlinkFromOriginal(scope[, linkTypeStr])
- unlinkTo(scope[, linkTypeStr])
- unlinkToOriginal(scope[, linkTypeStr])
- values([scope,]attributeNameStr)
- wordsRelatedTo(dataStr[, wordsNum])
- year(aDate[, yearsNum])

Listing of operators with conditional arguments

The listing below shows those operators where one or more arguments are conditional. This means the argument , apart for the case of if(), acts as a further filter to a scoping argument, as items evaluating `false` are dropped from the the source listing (in the case of dot-operators) or scope-derived list of item processed by the action.

Why the need for such a two-part approach of scope+condition? Early in the life of the app, the only scoping values allowed were designators which at that time did not include the ad hoc find(query) designator. Thus conditional tests allow fine tuning of a designator's scope. In many cases a find(query) used as the scope argument may obviate the need to use a conditional qualifier, but often the two part approach can be easier to write; either way the outcome can be the same.

Some operators support deeper evaluation than others—think of that in terms of how many nested levels of offset address arguments are included, or the number of implicit nested levels of evaluation. The variation might be from an explicit literal string for $Name or $Path, through data stored in an attribute, data stored in an attribute is a different note, to complex action code expressions. This aspect of operators is not well documented. The more complex the intended argument value, the better it is to do a small rest file first to ensure the argument is evaluated correctly before using the same in a large document.

List of such operators:

- any(scope, condition)
- avg_if(scope, condition, expressionStr)
- collect_if(scope, condition, expressionStr)
- count_if(scope, condition)
- every(scope, condition)
- if(condition){actions}[else{actions}]
- List/Set.collect_if(loopVar, condition, expressionStr)
- List/Set.count_if(loopVar, condition)
- List/Set.sum_if(loopVar, condition[, expressionStr])
- String.eachLine(loopVar[:condition]){actions}
- sum_if(scope, condition, expressionStr)
- while(condition){}

Listing of operators with loop variables

The listing below shows those operators that employ a user-named loop variable or loopVar. This is an in-loop placeholder for the literal value of the source list item currently being iterated upon. This when iterating through item #2 of a list, the loopVar's value is the literal value of item #2 in the source list.

Functions are included here as, in-function, the calling arguments are accessible in the same manner as seen with a loop variable.

Some operators support deeper evaluation than others—think of that in terms of how many nested levels of offset address arguments are included, or the number of implicit nested levels of evaluation. The variation might be from an explicit literal string for $Name or $Path, through data stored in an attribute, data stored in an attribute is a different note, to complex action code expressions. This aspect of operators is not well documented. The more complex the intended argument value, the better it is to do a small rest file first to ensure the argument is evaluated correctly before using the same in a large document.

List of such operators:

- eachLink(loopVar[,scope]){actions}
- function
- List/Set.any(loopVar, expressionStr)
- List/Set.collect_if(loopVar, condition, expressionStr)
- List/Set.collect(loopVar, expressionStr)
- List/Set.count_if(loopVar, condition)
- List/Set.each(loopVar){actions}
- List/Set.every(loopVar, expressionStr)
- List/Set.sum_if(loopVar, condition[, expressionStr])
- String.eachLine(loopVar[:condition]){actions}

Problematic Characters for Action code in $Name and $Path

If intending to do action code based creation/deletion/traversal of links in the document, it is advisable to avoid the following characters when titling notes (N.B. this list is *not* exhaustive):

- **/** forward slash. Used as the folder delimiter in paths.
- **** backward slash. Used as an escape character in strings.
- **(** opening and **)** closing parentheses. Indicates nested expressions in action code.
- **:** colon. Key:Value delimiter in look-up tables and dictionaries
- **;** semi-colon. The item delimiter in lists.

Using the above in note titles has no effect on non-action code activities.

If a path contains slashes or parentheses, Tinderbox checks for note names that exactly match the path. For example, if the input path value is `S/Z`, this matches either the note `Z` inside container `S` or the note named `S/Z`.

Paths containing Parentheses

Paths containing $Names including parentheses, e.g. "Fred Smith (Jr.)", will handle correctly during concatenation. In older pre-v6 versions, parentheses caused the new string to terminate at the first parenthesis. For a note with $Name "Fred Smith (Jr.)" setting a new path:

```
"/Some/Path/"+ $Name
```

…gives "/Some/Path/Fred Smith (Jr.)" unlike the old (and incorrect) "/Some/Path/Fred Smith ".

Improved parsing

From v9.1.0, the action parser behaves better when handling unquoted paths, as in

```
create(/Sources/People/Jefferson);
```

Previously, Tinderbox attempted to parse the path, which could lead to unexpected results. Now, a path with an initial "/" is recognised and treated as if it were a literal string. However, as literal paths containing commas, semicolons, or parentheses can confuse the parser.

Therefore if it is necessary to include commas and semicolons and parentheses in your path names, enclose the path in quotes or store the name in a string attribute.

```
create("/Sources/Martin Luther King, Jr./Speeches");
```

More normally paths are **not** quoted:

```
create(/Sources/Journals/Nature);
```

The same holds for attribute references

```
create($MyString);
```

Use of # and @

From v9.5.0, by default, Tinderbox scans all note name titles on creation/edit for these characters used as shorthand mark-up for assigning prototype and location data. This behaviour is controlled by a toggle in Document Settings ▸ Genera and so may be disabled if it is needed.

Conditional Actions (if clauses)

This note has been updated and moved to the action code operator list: see if().

Counting characters in strings

String.size and $TextLength (i.e. $Text.size) report the length of the UTF-16 representation of the text of the notes. Unicode can represent more than a million distinct characters. Most such characters count as a single character, i.e. count of +1 in String.size and $TextLength. These characters in the 'base multilingual plane 'which include most modern languages and many common symbols. They include 'invisible' characters such as spaces, line returns and tabs. This also includes:

- Western or Eastern European languages based on Latin characters.
- Most Central European languages and Turkish.
- Russian (and related Cyrillic scripts), Greek, Thai, Arabic, and Hebrew.

However, some other literal characters need more data to describe them, so-called 'double-byte' characters. Each of these counts as *two characters* for the purposes of String.size and $TextLength. Examples are:

- Emoji.
- Asian language scripts such as Chinese, Japanese, and Korean.

Getting and setting attribute values and inheritance

There are some things to bear in mind when setting attribute values , relating to inheritance of values:

- Setting an attribute to (no value) via code
- Setting an attribute to re-enable inheritance via code
- Short Boolean form for testing attribute values

Setting an attribute to (no value) via code

You can remove an attribute value by setting it to *nothing* (a longer explanation is below)

```
$MyAttr=;
```

Note the semicolon is not quoted, even for attribute type like strings. You can also use a empty string for string types:

```
$MyString="";
```

*Only the **first** of the syntaxes above* also allows attributes that use prototype or preference settings to re-inherit these again. So, do not use an empty string as a reset value unless that is your definite need. See about resetting inheritance fo more detail.

To explain the syntax further, you are effectively doing this to 'reset' an attribute to default inherited value:

```
$MyAttr=
```

…and then explicitly then adding a semi-colon to close the expression, thus…

```
$MyAttr=;
```

Note that the semi-colon is not part of the reset expression. So the last-but-one code above, even if used as a rule on its own, will work but is not recommended. The additional semi-colon makes sure Tinderbox does not have to guess your intent and also avoids you having to later remember to add a semi-colon if you add more code on the end of the original unclosed expression.

Setting an attribute to re-enable inheritance via code

Once an attribute has been explicitly set (i.e. not via inheritance) to a non-default value, Tinderbox inheritance no longer occurs. This is deliberate, the rationale being that the user wants that note's attribute to retain the value; maintaining inheritance might cause the user-set value to be unintentionally over-written.

This can catch out the new user when prototypes are in use. One of the strengths of prototypes is that a change to the prototype is instantly imposed on all notes using that prototype *except where a note's attribute has been explicitly changed*.

A very simplistic example might be where a prototype sets note $Color, e.g. 'blue'. If the user sets a note using the prototype to $Color 'green', if the prototype's $Color is then changed to 'red' all the other note's using the prototype will go red, but not the one explicitly changed to green. Many new users would expect them all to go red.

So, fine if the user wanted to take an attribute out of the inheritance stream. But, what if it was a mistake? In the above scenario, if the user reset the changed note's $Color back to 'blue' would inheritance re-occur? No. That is because whils the inherited colour is 'green', an explicitly set value is still seen as an explicitly set (i.e. non-inherited) value *even if the same* as the notional inherited value.

An easy way to confirm if an attribute value for a particular note is inherited or not is to view the attribute either as a Displayed Attribute or in the note's Get Info/Attributes tab (Cmd+Opt+i). Attributes being inherited are listed in normal weight text and those locally set in the note are shown in bold text.

How is inheritance re-enabled?

The trick is to set the attribute value to no value, something you cannot do by typing into a value input box (e.g. as in displayed Displayed Attributes). There are several approaches that can be used and which may appeal to different users' styles of work. The resetting process can be done in a number of ways:

- Use the Quickstamp Inspector. Select the note(s) and use Method 1 below.
- Use Get Info's (Cmd+Opt+i). Select the note and use Method 2 below.
- In an action or rule, using method 3 below.
- Menu 'normal' option in some pop-ups, see method 4 below.

Method 1: Use the Inspector (works for all currently selected items)

To change a single attribute for all selected note(s):

- Open the Properties Inspector, Quickstamp tab
- Do either of:
  - Choose the Attribute Group from the top pop-up and then the desired attribute from the second-pop-up.
  - Use the Search box and select the desired attribute from this of matches.
- Click the left button above the value box.

To change a multiple attributes, repeat the above process for each attribute in turn.

Method 2: Use Info view (works for current single item only)

To change a single attribute for a single note:

- Open Get Info, attributes tab (Cmd+Opt+i)
- Do either of:
  - Choose the Attribute Group from the top pop-up and then the desired attribute from the second-pop-up.
  - Use the Search box and select the desired attribute from this of matches.
- Select the line of the table for the desired attribute.
- Right-click and select the 'Use Inherited Value' menu item. The attribute name and value will revert from bold text to normal text.

To change a multiple attributes, repeat the above process for each attribute in turn.

Method 3: Use Code (in an attribute or stamp)

Code can be run as a $Rule or $AgentAction or within a stamp.

To set *any* data-type of attribute to its inherited value:

```
$MyAttribute=;
```

Note that no quotes are used around the semicolon. The above is not the same as simply setting a string to empty or a number to zero. Why? Because the latter is simply setting that value locally and thus still blocking inheritance.

If needing to reset a long list of attributes, consider using a list and loop.

Method 4: Use menu 'normal' options

A number of menus, pop-ups and context menus for setting values have a 'normal' item at the top of their value listing. For example, this can be seen with menus for setting $Color or $Badge (see example). In any of these, clicking the 'normal' option will reset the local attribute to re-inherit values.

Short Boolean form for testing attribute values

It is possible to cite attribute tests in both a long form:

```
if($MyBoolean == true)
if($MyBoolean != true)
```

or a simple short form:

```
if($MyBoolean)
if(!$MyBoolean)
```

Note in the negative version the '!' moves in front of the attribute name (and the $-prefix). Thus `!$MyString` means $MyString has no value set whilst `!$Name(Test)` causes a reverse query where all note's whose title does not include the string "Test" are matched.

Though the short form is primarily for Boolean attributes, where another data type can be coerced to a `true/false` equivalent the short form may be tried. The coercion being tested depends on the attribute type's default value. Any value other than the default, results in a coercion of `true`. $ChildCount is a number type attribute so the default is zero. Both the code examples below equate to the same test but the latter is shorter, thus useful in longer, more complex expressions.

```
if($ChildCount > 0) {… if($ChildCount) {…
```

The converse would be:

```
if($ChildCount < 1) {… if(!$ChildCount) {…
```

Note that the short Boolean form, whilst coercing non-Boolean type values to a `true/false` occurs it is not testing whether the default value is inherited or was explicitly set; see also use of the |= operator.

Delaying code execution in prototypes and notes using prototypes

When using pre-planned structural prototypes, i.e. those stored apart from actual content and with no purpose other than as a means to configure other content notes, consideration should be given to whether action code such as found in in Action-type attributes is run in the prototype itself, or should be suppressed there and only run in a note inheriting from the prototype. This scenario affects code placed in the Action-type attributes (e.g. via their create/rename dialog boxes):

- $OnAdd
- $Rule
- $AgentAction (agents only)
- $DisplayExpression
- $TableExpression

In large or complex documents, it is possible to end up with many notes using the same prototype. If the later has a complex $Rule or $DisplayExpression, this can place quite a loading on the speed with which the cycle of running action code is completed. These two action attributes attributes have additional special system attributes specifically to suppress prototype-based executions a result the attributes $RuleDisabled or $DisplayExpressionDisabled are intrinsic: setting them in the prototype has no effect as these attributes are inherent and not inherited by notes using the prototype.

Controlling use of inherited complex actions

Thus, in some cases it is desirable to delay code execution as the code is written with the intention of running in an inherited context, using actual data from the inheriting note, but only when intended. This scenario requires a different approach. One method is to make the action in inheriting notes check the disablement state *in their prototype*. This example is for a rule:

```
if($RuleDisabled($Prototype)){
    if($MyNum>0){
        $Color="bright red"
    }else{
        $Color=;
    };
}
```

However, be aware the rule still exists and the outer if() condition is still evaluated, so this approach does not remove the rule entirely. For actions with no disablement attribute, consider the same but using a user Boolean attribute set in the prototype.

Another example:

```
if(!$IsPrototype){
    $AgentQuery="Topics("+$Name+")";
}
```

The above ensures that, in a prototype agent, the agent's query is not set within the prototype whereas in all agents inheriting from the prototype the query will be set. The need for a wrapping if() conditional test becomes clearer with this rule:

```
if(!$IsPrototype){
    $AgentQuery="Topics("+$Name+")";
    $Rule="";
}
```

Without the if() test, the latter rule would run and delete itself within the prototype and never set anything in the inheriting agents.

Note too, that if using self-deleting rules via a prototype the act of clearing the rule breaks inheritance for the $Rule attribute.

Using multiple prototypes

Another method for switching complex actions on/off is to use prototypes for the existing prototype. For instance, in the example of a complex rule, keep the existing prototype but cut the rule code and move it to a new, otherwise un-customised prototype. Now, to turn the rule on, set the $Prototype of the existing prototype to the name of the new one; this causes it to inherit the rule. Resetting the $Prototype value to the default (nothing) causes the main prototype to inherit the default $Rule, i.e. nothing. Be aware that this method only really works for turning one attribute (i.e. rule or edict) on or off, or if multiple actions (e.g. both rule *and* edict) are to always enabled/disabled at the same time.

Stream Processing and parsing

Tinderbox's string processing operators are intended to help extract information from structured and semi-structured text. Such text may be hand-typed, for copied from sources like email. Often, it may be imported from other programs or

downloaded from web services into a Tinderbox attribute. The need is to extract needed information from this text.

Tinderbox does not handle true streams, i.e. continuously reading an API output in real time. But in its 'stream processing' Tinderbox is behaving in the same manner. A key point is the stream is read by moving forward, never backwards—the process essentially forgets the already precessed parts of the stream. While the overall process can be run again on the same source, e.g. a note's $Text, *within* a running stream parse, it cannot move/look backwards in the source ('stream').

'Stream' parsing uses 'lines' of text. Importantly this isn't a line as seen on screen but content between the start, each line break and end. In lay terms a paragraph of text is a 'line'. Understanding this is important to following how stream parsing goes about its task.

Regular expressions. Be aware that stream processing operators do not use regular expressions (regex), except in clearly marked exceptions. If regex are needed to complete the task, either use ordinary String processing operators or insert appropriate delimiters into the text before processing.

Processing a text as if it were a stream

Broadly speaking, the parsing approach is to begin at the start of the string and proceed, step by step, following a recipe (of chained dot-operators). For example, such a 'recipe' might say:

- Read until you find a line that begins with "To:", "From:", or "Subject:"
- If you find a "To:", copy everything character following that up to the first space character encountered and save the copy in at the current note's $Email.
- If you find a "From:", copy everything character following that up to the first space character encountered and save the copy in at the current note's $EmailFrom.
- If you find a "Subject:", get the rest of the current line and use that for the $Name of the current note.
- Having found a "Subject:, delete all the headers you have processed and leave the rest of the text.
- If you never find a "Subject:", do not delete anything.

All functional string processing operators accept a string, in this documentation called the *stream*, of text being processed. In the the majority of cases, but not all, this is likely to be a note's $Text, or an attribute/variable value based on some $Text.

Stream processors act in some way on the stream possibly saving some data into an attribute or simply moving further forward (left-to-right) and returning the unprocessed remainder (right-most portion of the stream) which may be passed to another operators such as further chained dot-operators. For example:

```
$MyString.skip(22).captureNumber("MyNumber");
```

takes the value of MyString, skips exactly 22 characters, and extracts a number to be stored in $MyNumber. For instance MyString holds string "We think there may be 1234 items":

```
$MyString.skip(22).captureNumber("MyNumber");
```

$MyNumber is 1234. But if MyString holds string "We think there may be 1,234 items" then $MyNumber is 1 as a comma follows the first number (after the skip operator consumes the first 22 characters.).

The parsing operators can best be understood as a series of discrete roles:

- Expect operators
- Skip operators
- Capture operators
- Functional control structure operators
- XML processing
- JSON processing

Expect operators

The expect family of operators looks at the current place in the text stream to ensure that the text is what you expect it to be.

String.expect("matchString")

Tests that the next characters in the stream are are a literal string "matchString", and advances the stream beyond "matchString". Otherwise, a failure occurs and parsing ceases.

String.expectWhitespace

Tests that the next characters in the stream are whitespace, such as spaces, tabs, and carriage returns (i.e. 'whitespace' as in regular expression terms). It advances the stream to the first non-whitespace character. A failure occurs if the next character is not whitespace.

String.expectNumber

Tests that the next characters in the stream can be interpreted as a number. Numbers include "0", "5.7", and "-13". Skips over the number. A failure occurs if the next character is not part of a number.

String.expectWord

Tests that the next character is not whitespace, punctuation, or a digit. Skips to the next whitespace, punctuation, or digit. Otherwise, a failure occurs and parsing ceases.

Skip operators

The skip operators advance the current position of the stream.

String.skip(N)

Skips the stream forward exactly **n** characters. A failure occurs if **N** reaches past the end of the stream.

String.skipTo("match string")

Advances the stream to the first occurrence of the target **match string** (which is **not** a regular expression regex), skipping over the target string so the stream parsing position now sits *after* the match. A failure occurs if the target is not found. An alternative, that supports regex is String.extract().

String.skipWhitespace

Advances the stream to the first character that is not whitespace. A failure occurs if the stream is exhausted.

String.skipToNumber

Advances the stream to the next number (i.e. one of more continuous number characters). A failure occurs if the stream is exhausted.

String.skipLine

New to v9.5.0, skips forward to the next carriage return or to the end of the stream.

Capture operators

The capture family of operators extract a chunk of information from the stream and store them in a target attribute. The target attribute *may* be the source of the stream, in which case the parse will cease at the first capture.

String.captureLine[("targetAttribute")]

Captures text from the current stream position, up to the next line ending or the end of the stream. The text is stored in the designated **targetAttribute**, and the stream advances to the character following the line break.

String.captureNumber[("targetAttribute")]

Captures a number from the current steam position, up to the next character that cannot be considered part of a number; the capture is passed **targetAttribute**. Advances the stream to that character. A failure occurs if the stream cannot be interpreted as a number.

String.captureWord[("targetAttribute")]

Captures text from the current stream position up to the next character that is a digit, whitespace, or punctuation; the capture is passed **targetAttribute**. Fails if the stream is empty or begins with a digit, whitespace, or punctuation.

String.captureTo[("string","targetAttribute")]

Stores the string up to but not including the designated literal **string** (it is not a regular expression) in the designated attribute, and returns the string that follows **string**. An alternative, that supports regex is String.extract().

String.captureToken[("targetAttribute")]

If the string contains a token specified by a prior parsing operator, the token itself is passed to **targetAttribute**. The stream is not advanced and any chained parsing continues from the same point.

String.captureRest[("targetAttribute")]

Captures the rest of the stream in as the value of **targetAttribute**, and returns the empty string.

String.captureXML

Attempts to parse the string as fully/partially XML and fails if unsuccessful. The parsed XML is saved as the current XML (stream) object. Essentially, this re-scopes the current stream so that it contains only the contents of the first section of XML detected within the original stream. The focus of the stream parsing is set to the beginning of the extracted XML code. Only one XML object may be current at any time. If the source stream contains multiple discrete XML code sections, only the first is detected/used.

String.captureJSON

Attempts to parse the string as fully/partially JSON and fails if unsuccessful. The parsed JSON is saved as the current JSON (stream) object. Essentially, this re-scopes the current stream so that it contains only the contents of the first section of JSON detected within the original stream. The focus of the stream parsing is set to the beginning of the extracted JSON code. Only one JSON object may be current at any time. If the source stream contains multiple discrete JSON code sections, only the first is detected/used.

Functional control structure operators

Functional control structure

These control structures are used with functional stream processing.

String.eachLine(x[:condition]){action}

The .eachLine(){} operator allows iteration through each line of a string, performing an action on each. For attributes like $Text, a line equates to a paragraph as the delimiter is a line break character.

String.try{action}[.thenTry{action}]

Saves the value of an attribute and attempts an action. If the action fails because one of its operators fails or the fail() operator is performed, the value of the attribute used as a stream source

A .try{} may be followed by one or more .thenTry{} clauses. If the original .try{} succeeded, all subsequent .thenTry{} clauses are ignored. If the original .try{} failed, then each .thenTry{} is attempted in turn. Once a .thenTry{} clause succeeds, subsequent .thenTry{} clauses are ignored.

String.failed()

tests for a failed stream processing action, e.g. using Stream.try{}. Its use include an explicit fail() call.

XML processing

Tinderbox can process XML stored in any attribute, including $Text. XML element attributes are not related to Tinderbox attributes. XML stream operators are expecting the whole contents of the stream to be a valid XML object.

XML.xml(path)

Returns the value of the object at **path**.

XML.each(path){action}

Iterates the child objects at the **path** using an **action**.

String.captureXML

Attempts to parse the Stream's string and re-scope it it as only the (first) XML code detected within the text. It fails if unsuccessful.

JSON processing

Tinderbox can process JSON stored in any attribute, including $Text. JSON stream operators are expecting the whole contents of the stream to be a valid JSON object.

JSON.json[key]

If there is no current JSON object, attempts to parse the string as JSON and fails if unsuccessful. If there is a current JSON object, that object will be reused. If the top-level element is an object, JSON.json[key] returns a dictionary for that object.

JSON.json[N]

If there is no current JSON object, attempts to parse the string as JSON and fails if unsuccessful. If there is a current JSON object, that object will be reused. If the top-level element is an array, JSON.json[N] returns the **N**th object.

JSON.json.each{action}

If the top-level element is an array, rebinds the JSON object in turn to each array element. After calling the action block for each element, the JSON object is restored.

String.captureJSON

Attempts to parse the Stream's string and re-scope it it as only the (first) JSON code detected within the text. It fails if unsuccessful.

JSON property chaining

JSON doesn't have a formal pathing method like XPath in XML, but JSON properties can be chained to allow navigation to nested objects. For example, consider:

```
{
  "person": { "firstName": "Thomas", lastName: "Roe"},
  "coordinates" : [-90,41]
}
```

Thomas's latitude can be addressed via

```
$Text.json["coordinates"][0]
```

And his family name via

```
$Text.json["person.lastName"]
```

Using attributes as global variables or constants

It may be useful to store a snippet of information, such as the local sales tax percentage or the name of a certain project in a manner such as lets any note read or set that value. In programming this method is often known a a constant or global variable.

This is slightly different to setting an attribute default value whereby all attributes start with the same value. Using the local sales tax example above, it might be necessary to have a value of 0.05 (5%) that can then be used by any action code trying to figure a true sales total.

This simplest method to do this is to simply place the value in an attribute (of suitable data type) in a note outside the main content. Because it is an attribute you can both fetch the value and (re-)set it. Things to consider:

- place such value placeholder notes away from your main content so they do not accidentally get edited, e.g. in a separate branch of the root.
- for map users, place variable notes off away on the side of the map.
- give the notes unique names, allowing them to referred to by $Name alone. Otherwise, you will need to cite their $Path instead. In the case of the latter, consider keeping the path short
- a note can hold one 'variable' or several, assuming each is kept in a separate attribute. Which makes most sense is mainly down to personal choice.
- you do not need to make a $SalesTax attribute just to hold the sales tax variable. Again whether you do so is a matter of personal style; there is no right or wrong way.
- if you have, say *a lot of* number variables, it probably makes less sense to gives them each an attribute. Rather, use something like $NumberVariable, and store each variable in that attribute within a suitably named notes, allowing you to reference things in a manner like $NumberVariable(varLocal_Sales_Tax)
- if you are worried about user accident or code error changing an important constant value, set the note's $ReadOnly attribute to `true` as this stops the note's attributes changing.

Chaining 'dot' operators

Where pertinent, 'dot' operators (operators prefaced with a period— see listing), like **String.split()**, can be chained together. If an operator's parentheses are normally optional, they be omitted if that operator has another operator chained to its right. Thus:

`$MyList.sort().reverse()` GOOD
`$MyList.sort().reverse` GOOD
`$MyList.sort.reverse()` GOOD
`$MyList.sort.reverse` GOOD

The observation "where pertinent" points to the fact that the dot operators used must act on *compatible* data types:

`$MyList.sort().reverse()` GOOD
`$MyColor.red().empty()` BAD

It is most likely that opportunities for chaining only occur in relation to text data functions though do be aware that some text dot operators are only for lists/sets and some only for string even though others cover both types of data.

Chaining to parentheses

It is possible to chain to parentheses and indeed is sometimes necessary. The parentheses ensure Tinderbox evaluates contents of parentheses *before* the dot-chained actions are applied. In this example, three strings are concatenated to a single string before its size (number of characters) is calculated:

```
(Hello" +" "+"world").size
```

The above example is trivial but shows the general principle. In practice, the technique can become useful is the expression is complex and requires Tinderbox to fully evaluate it before processing the chained function. For instance, the links() operator already uses dot-chained arguments and so to find the number of links found using .size, it is necessary to place the whole links() expression in parentheses:

```
(links(children).inbound..$StartDate).size
```

Look-up tables

(NOTE: The prefix 'list' may be used interchangeably here with 'table'. Thus, 'look-up list' and 'look-up table' are describing the same thing.)

Look-up tables, are listings of `key:value` pair items allowing simple value arrays (more formally called a one-dimensional array).

See also: Dictionary-type attributes

Since look-up tables were added, Tinderbox added the Dictionary data type. It too holds key:value pairs but Dictionary-type attribute offer extra affordances for creating/editing data via action code.

In many cases, if starting from new, a Dictionary may well be a better choice than a List for this purpose.

Look-up table functional example

For those without a coding background, this is more easily understood by use of a simple example. Thus, suppose some notes record the US state (in attribute $State) in which each customer resides. Each state is also assigned to on of a number of a regions and it is useful to know the sales region ($Region) for given customer's home $State. Before support for look-up tables, a usual approach—to avoid hard coding each note's $Region—would be via use of conditional if() statements:

```
if($State=="AL") { $Region = "South";}
```

However, this is a tedious process if there are many states as it needs a lot of if() statements, at least 50 in this case!

To use a look-up table for mapping $State to $Region, we make just *one* listing of colon-joined `key:value` pairs, separated by colons, thus: Key1:Value1;Key2:Value2. For example:

```
$RegionList=[AL:South;AK:NorthWest]
```

Several data types can be used to store look-up tables, and that aspect of use is discussed further below

Limitations on key and value definition

A *key* is:

- case-sensitive
- **must** be unique to the listing
- ought not to consist of numbers only as this *may* confuse Tinderbox (is it a key value or the item number in the list?)
    - should there be a need to use numbered keys, e.g. if generating the look-up key based on a computed value, place a letter before the number. So, use 'x1' rather than just '1'.

A *value* may be the same as values for different keys within the current listing.

Accessing look-up table values

Look-up tables are interrogated using the .lookup("key") operator like so:

```
$RegionListing=[AL:South;AK:NorthWest];
$Region=$RegionListing.lookup("AL");
```

sets a $Region value of "South".

Optional default value for missing keys

If the look-up table does not contain the supplied *key* value, **.lookup()** then looks for an *optional* special default key (`default`, case-sensitive) and returns that key's value or if none is found, an empty string is returned (if the 'default' key exi
but has no value, an empty string is returned). So:

```
$RegionListing=[AL:South;AK:NorthWest];
$Region=$RegionList.lookup("CT");
```

gives a nil value of "" as there is no matching key 'CT'. If the option 'default' key is added to the look-up table:

```
$RegionList=[AL:South;AK:NorthWest;default:Unknown];
```

Then

```
$Region=$RegionList.lookup("CT");
```

sets $Region to "Unknown".

The *key* input argument can be evaluated. Thus if attribute $State has a value "AL", then:

```
$Region=$RegionList.lookup($State);
```

sets a $Region value of "South".

Alternate keys, same value

Lookup tables are able to specify several keys that map to a common value. This is done by separating each key with the pipe character '|'. For example:

```
[Oliver|Micawber|Pip:Dickens;Tess:Hardy;Palliser|Finn:Trollope;default:anon]
```

A structural representation of the above example is as follows, with each key:value pairing on a separate line for clarity.

```
Key1-1|Key1-2|Key1-3:Value1;
Key2-1:Value2;
Key3-1|Key3-2:Value3;
default:anon
```

The number of keys in each Key/Value set is discrete from other sets in the list: 3 keys map to Value1, one to Value2, two to Value 3 and default has one value..

Keys are matched on exact, *case-sensitive*, strings. In the above example "Tess" will return "Hardy" but "tess", "Tes" or "Tessa" would return "anon". If no default had been defined, an empty string "" would be returned instead for the non-matching values.

Lexical and numerical ranges for keys, same value

Lookup tables also permit ranges (based on case-sensitive lexical sort based matching). For example, suppose the value of $MyList is:

```
[Alaska-Connecticut: 1; Delaware-Nebraska: 2; default: 3]
```

Then `$MyList.lookup("Alabama")` would return string "1", and `$MyList.at("Illinois")` would return "2". However, `$MyList.lookup("alabama")` returns "3" due to the way Tinderbox lexical sort works, as would. Note too, tha
in this case `$MyList.lookup("Manitoba")` would return "2" as "Manitoba" lexically sorts between "Delaware" and "Nebraska": Tinderbox has no way to know Manitoba is a Canadian province rather than a US state.

Lookup tables can also work with numeric ranges. If $MyList is:

```
[0-10: red; 10-20: blue; default: green]
```

Then `$MyList.lookup(5.0)` returns 'red'. Note that `$MyList.lookup(5)` (N.B. an integer *without* a decimal point matches the key "5". Thus if the actual key is "5" use "5.0" to trigger the first behaviour.

What data type to use for storing look-up tables?

When the feature of look-up tables was first added, the Dictionary data type did not exist, so a List or Set was suggested. Sets, make more sense than lists are they avoid unintentional duplicates. Legacy: not that prev9.5.0 using List.unique
to de-duplicate a list re-sorts that list. As a result either a Set or Dictionary type attribute is suggested for holding a look-up table. List type may also be used, but attention should be given to accidental key duplication.

For Dictionary type, as long as used with **.lookup()** rather than the normal **Dictionary["key"]** access methods, a Dictionary supports all the above advanced look-up table behaviours (alternate keys, key ranges).

Storing the look-up table in one location

In the first above example, the look-up table listing is stored in the note being tested, for simplicity of explanation and learning. But, in reality, this same list may need to be used in hundreds of different notes. Luckily this is easily done by defining the look-up list as a global value, once, in a single note and then referenced from all others by an offset attribute reference, i.e. using $AttributeName(name) or $AttributeName(path). If using this technique chose a unique note $Name, you will need to use the full $Path on the offset.

Thus these examples are more representative of real-world use:

```
$Region=$RegionListing("config").lookup("AL")
$Region=$RegionListing("config").lookup($State)
```

Noting the point about the name of the note, it may make more sense to use a deliberately unique name that also implies the note's purpose. So:

```
$Region=$RegionList("lt_States").lookup("AL")
```

A note called 'lt_States' ('lt' for 'look-up table'), will stand out in action code and imply a look-up table is being referenced and when revisited months later the note's $Name may remind us as to its purpose.

Actions, Stamps and Quickstamps

A Tinderbox document has a number of ways to change attribute values. First it is useful to draw a distinction between the following:

- a literal value. This is the actual stored value of an attribute and as seen when displayed in contexts like a note's Displayed Attributes table.
- an evaluated value based on action code. Here it is code that is stored and then evaluated when applied so as to render a literal (above) stored value.

Attribute value changes can happen in various contexts:

- via action code (also allows literal value input)
  - container-based:
    - OnAdd action
    - OnRemove
    - Agent action.
  - selection-based (note: selected items do not need to be in the same container):
    - Stamp.
- via literal value of an attribute:
  - selection-based (note: selected items do not need to be in the same container):
    - Quickstamp.
  - single item only:
    - Get Info/attributes.
    - Displayed Attributes.

OnAdd

This an action applied *once* to each note that is added to, or created in, a container note. The OnAdd action also takes effect when the note containing the on-add code is converted into a container: that is, when another note (or notes) is dragged onto it, creating new child notes. In the latter case the OnAdd is applied to the newly added/created note after it has been moved in situ; this is important so code evaluating things like the parent note's title are resolved correctly.

If a selection of multiple notes are moved into a container, the same action is run once on each discrete note. Removing a note from a container does not trigger this action but see the OnRemove action below. Map adornments also have an OnAdd that is run when note(s) are moved onto an adornment.

The OnAdd action is stored in a note's $OnAdd attribute. The action can be manually set via the Action Inspector's Action tab, Get Info's attributes/General tab, or by displaying $OnAdd as a displayed attribute. Otherwise, the attribute can set via any action code method, such as stamps or quickstamp.

OnRemove

The action is the mirror opposite of OnAdd and is an action applied once to each note that is removed from, or deleted from, a container. Map adornments also have an OnRemove action.

Agents also have an OnRemove action but note that it is applied to an alias of a note, so pay attention if altering intrinsic attributes.

Note: *As this action is a much more recent feature than OnAdd, older articles & tutorials on actions my (now) incorrectly state that the is no remove action.*

The OnRemove action is stored in a note's $OnRemove attribute. The action can be manually set via the Action Inspector's Remove tab, Get Info's attributes/General tab, or by displaying $OnRemove as a displayed attribute. Otherwise, the
attribute can be set via any action code method, such as stamps or quickstamp.

OnJoin

This action is evaluated when a note is first added to a composite. If an existing composite note is touched by the drag and it has $OnJoin code, it is run upon the newly joined note.

OnVisit

The $OnVisit action is primarily for Storyspace compatibility but may be used as an 'Upon Selection' event trigger.

Agent action

Agents have no OnAdd action, but instead an agent 'action' and this also functions slightly differently. The agent action is run on each discrete child alias in an agent, but is run *every* cycle of the agent update.

The agent action is stored in a note's $AgentAction attribute. The action can be manually set via the Action Inspector's Action tab, the AgentAction box of Get Info's agent1 tab Get Info's attributes/Agent tab, or by displaying $AgentAction a
a Displayed Attributes. Otherwise, the attribute can be set via any action code method, such as stamps or quickstamp.

Note: *although an agent can ostensibly have an $OnAdd attribute value, it is never used. If a selection includes both notes and agents and the Action Inspector's Action tab has code added, Tinderbox will apply the code to $OnAdd or $AgentAction as appropriate.*

Stamp

A stamp is one or more complete action code expressions (i.e. complete actions) stored for ad hoc use. Stamps are created and saved via the Document Inspector's Stamps. Once created, a stamp can be applied from this Inspector, by clicking the stamp's name in the Stamps menu, or can be called via action code. All the stamps for a document are stored in the document's TBX file

When used a stamp is applied *once* only to each selected item. This can be a useful alternative to an agent if the code is needed only occasionally. By applying a stamp to an alias inside an agent, the stamp can be used to simulate/test a possible agent action whilst acting on a single item.

Quickstamp

A Quickstamp is a fast method to (re-)set the value *once* for a single attribute for every note currently selected. Unlike the methods above, Quickstamp can only change the literal value of an attribute, i.e. the input cannot be action code but must be the actual final value itself. Also, unlike a stamp, the applied value is not saved, e.g. for later re-use, and cannot act on more than one selected attribute at a time. Quickstamp can only be used via the Properties Inspector's

Quickstamp tab.

One small exception to the literal value rule in Quickstamp is that if the selected attribute is Date-type, date designators *are* evaluated when entered (i.e. 'now' will become a date/time string in the Quickstamp value input box when return is entered, the latter string then being applied when the Apply button is pressed. However, this is a niche exception to the general fact that Quickstamp is intended for inserting literal values.

Get Info/attributes & Displayed Attributes

These offer different UI routes to the same interaction by changing the literal value of a single attribute in a single selected item. In each case the user selects the desired attribute and manually edits the actual value. UI: Get Info/ attributes, Displayed Attributes.

Unlike Quickstamp, the context menu in these input UIs does offer an 'evaluate' method which will evaluate action code in the input box and replace it with a literal value. Although this allows use of (simple) code on an ad hoc basis, the primary edit means of these UIs is attribute literal values.

Self-Cancelling Rules & Actions

By setting an action or rule to nothing as the last of a set of chained tasks, it can be self-cancelling. This can be done as part of the rule/action itself and can help to avoid unwanted repetition/looping. It is more normally needed with rules thar actions as rules run every agent update cycle.

Consider this rule:

```
$Color="blue";$Rule="";
```

When the rule is run, the note's $Color is set to 'blue' and the Rule is set to nothing. Thus the $Color assignment is run once-only before the rule ceases to exist. However, the above only sets the Rule to no value, it does not re-enable inheritance. For that a slightly different syntax is needed:

```
$Color="blue";$Rule=;
```

Using regular expression back-references

Using parentheses within a regular expression regex, it is possible to set up to nine back-references from within the overall regex. These discrete sub-matches can then be used in the action connected with the query. The most normal means of setting a regex with back-references is by using the operators String.contains() or String.icontains().

Back-references can be used in actions in several contexts:

- Most obvious is in an agent's action (referencing the agent's query).
- Within general action code:
  - with the if(query){actions} conditional action, back-references for the conditional query can be used within any action(s) enclosed within the operators' {} brackets—in both the 'if' and 'else' branches.
  - although not formally queries, the action String.replace(regex,replacement) can set back-references in the first (*regex*) input argument. The regex can be a literal string or a regular expression and, if so, then the back-references it creates can then be used within operator's second (*replacement*) argument.

Exceptions:

- Although Macros use the same back-reference *style* of notation for inserting content, in that case the values are drawn from the macro's input arguments rather than from a regex.
- Whilst the find() action operator uses queries, the operator uses this to return the paths of matching notes and therefore does not support regex back-references.

From v9.5.0, if(){…} and if(){…} else {…} now restore regular expression back-references ($1, $2…) to their state prior to the next if() statement.

IMPORTANT: The examples below are not intended to teach how regex work but simply to illustrate how back-references are used once created.

Referring to a back-reference

The method of referring to a back-reference is via a $-prefixed number, $0 through $9. The back-reference $0 always refers to the the *whole* matched string (or sub-string) for the stated query regex, i.e. it may match all or part of the target string. $1 to $9 refer to any defined back-references *within* the overall regex, as discussed in examples below.

Back-references are returned (i.e. number-referenced) in the order created. The order is usually left-to right in order the parentheses open (note this allows for nesting) but to understand that process better, read up on regular expression back-references.

Do back-references need quoting? No, if $MyString is "This or that", *all the following* result in a value of "This and that":

```
$MyString=$MyString.replace("(^.+)or(.+$)","$1and$2");
$MyString=$MyString.replace("(^.+)or(.+$)","$1"+"and"+"$2");
$MyString=$MyString.replace("(^.+)or(.+$)", $1+"and"+$2);
```

Back-references 1: in an agent context

This is an example of an agent query designed to create back-references that can then be used in the agent's query:

query: `$Text.contains("email: (\w+([,| |-]*\w*)*)\<([^>]+)\>, on (\d+/\d+/\d+)")`

action: `$FullName=$1; $Email=$3`

The action will set the value of attributes $FullName and $Email using the back-references to regex found in the currently focused notes $Text (well strictly, the note's alias as this is an agent). So, for a worked example, if the $Text was:

```
Project X
Brief discussion to finalise resources allocation
Source email: John Doe<johndoe@example.com>, on 24/03/2010
Follow up actions: Bob, Mary.
```

…then the above query would give the following back-references:

- $0: `email: John Doe, on 24/03/2010` (i.e. the full matched *sub-string* within the source text—i.e. not necessarily all of that text).
- $1: `John Doe` i.e. the contents of the *first* parentheses-delimited code `\w+([,| |-]*\w*)*` (note the nested parentheses to deal with names of two or more words.
- $2: Empty, nested inside $1 it serves to capture second and subsequent words in $1. See below for more on nesting back-reference groups.
- $3: `johndoe@example.com` i.e. the contents of the *third* parentheses-delimited section of code `[^>]+`
- $4: `24/03/2010` i.e. the contents of the *fourth* parentheses-delimited section of code `\d+/\d+/\d+`
- ($5 through $9: returns nothing, as they have no source match defined.)

Back-references 2: using if(){}

Using the same examples as above, an if() usage might look like this (the line breaks are not significant and only for clarity of reading here):

```
if($Text.contains("Emailed by: (\w+([,| |-]*\w*)*)<([^>]+)>, on (\d+/\d+/\d+)")){
 $MyString=$0;
 $FullName=$1;
 $Email=$3;
 $StartDate=date($4);
};
```

In this method the if() operator holds the query and generates the back-references. These can be used anywhere within, but only within the operators { } curly braces enclosing the action code. The back-reference could be used in the `els` { } branch, but the nature of the overall usage (i.e. for back-reference generation) means this is unlikely.

See if() for further back-reference usage examples.

Back-references 3: using string.replace()

The use of string.replace() is to replace part of an existing current string attribute value. The operator can be thought of in terms of $SourceDataString("query","return string") where the "return string" might be one or more back references forr the query and may include string literals.

For example, assume $MyString has the value of "AABBCC", from which it is desired to make a value of "BB". Essentially this means deleting all the non-'B' characters. This can be done by capturing the 'B's in a back-reference and using that to replace the original value. Thus to replace the original $MyString value:

```
$MyString = $MyString.replace(".*(BB).*","$1");
```

Note the $1 back-reference must be *inside* quotes for the second argument to work. Alternatively, the altered string can be saved to a different attribute, leaving $MyString unchanged

```
$AnotherString = $MyString.replace(".*(BB).*","$1");
```

The back-references created here cannot be used *except in the second input* ('replacement') argument. Clearly, the applications for using string.replace() are far more limited than when using an if() statement.

See String.replace() for further examples of use of back-references within an action context.

Nesting back-references

Back-references may be nested is side one another (as seen in the opening example above):

Query: `$Name.contains("(a(ard))v(ark)")`

Action: `$MyString =$1; $MyStringA = $2; $MyStringB = $3;`

For the matched note the 3 attributes set by the action will hold, in order, "aard", "ard" and "ark". This shows back-references are numbered in the order encountered running left to right and not by some other system such as the level of nesting.

Literal parentheses

Literal parentheses in regexs must be escaped by a backslash. To match "this (that) other", use:

```
$Text.contains("this \(that\) other")
```

To capture "(that)" as back-reference $1:

```
$Text.contains("this (\(that\)) other")
```

Sometimes parentheses are needed, e.g. in the agent example shown earlier above, in order to achieve the right match, but which do not match anything meaningful to back-reference use. Do not worry about that, you do not need to use every back-reference created.

What is the role of $0?

$0 is always the whole matched (sub-)string for the stated attribute value but if the regex regex creates additional back-references within the query then $1 through $9 may be used to access those additional match sub-strings.

In this case above, $0 is not all of the current note's $Text, the overall source for the query, but rather it is all the text matched within $Text by the regex code in the '.contains(" *regex*")' operator's *regex*.

Often, the *regex* matches the entire source so $0 returns the whole source text. The structure of the example above is deliberate, so as show that $0 attaches to the regex's match rather than simply being the entire text being passed to the regular expression.

Do not worry too much about getting the right number. If new to this sort of work and using a regex with several back references, you are strongly advised to try it in a small test file first. This makes it easier to make sure:

- that the overall regex matches the right notes
- that the back references return the right content
- which $-number refers to which extracted content

Fetching all back-references

From v9.6.0, the %matches operator returns a list of all populated references in order $0 to $9. Thus if a query populates 3 back-references within the overall match, then %matches returns a List of $0, $1, $2, and $3.

Returning the match offset position (dot operators only)

If the regular expression regex used with the contains() family of dot-operators (e.g. String.contains()) is found the function returns the match's offset+1, where offset is the distance from the start of the string to the start of the matched regex. Formerly, .contains() returned `true` if the regex was found. The '+1' modifier ensures that a match at position zero returns a number higher than zero which would otherwise coerce to `false`. Since the offset+1 is always `true`, no changes a required in existing documents but the function also gives usable offset information, albeit requiring adjustment for use with zero-based indices such as List.at() or String.substr().

Using long sections of code—code notes

NOTE: this concept pre-dates newer features like functions, and stamp notes in Built-in Hints. The concept of a note for holding code, such as export boilerplate code—i.e. not formal templates but used in generating export code, still holds true.

In code input boxes, using opt+Return will insert a line break into the code whereas Return executes the change.

The 'code note' concept

If long pieces of action code are needed, such as do not easily fit into a dialog's input box, then another note's $Text can be used to store the code. This approach is generally described as a 'code note' but note this is not a formal Tinderbox concpet. A new note is created and its text is used to just hold the code. This offers:

- More overall editing space for code.
- Using the affordance of some built-in prototypes (see below).
- Line (paragraph) breaks are ignored when code is run so each statement can be on a different line.
- They can be re-used by numerous notes.

Use the 'Code' or 'Action' prototype

If using code notes, first add the ' Code' prototype from the built-in prototypes to the TBX and then assign the prototype to your code note(s). If using several such notes, make a codes container and set the "code" prototype via its $OnAdd. The code note sets a monospace font, removes paragraph (line break) spacing, turns off auto-lists, etc. All this makes setting and testing code much easier.

An important consideration with these 'code repository' notes is that generally it is not required to export such notes or have them appear in the content part of a document (e.g. in agents). If using a top-level split of content and utility notes, these code repository notes should definitely go into the utility.

From v9.6.0, if the code being stored is only action code, then use of the ' Action' built-in prototype may be more useful as this employs syntax colouring of action code and detection of unbalanced brackets.

Using a code note's code

The best way to deploy code in the code note is to use its rule (or edict if preferred) to set the rule/OnAdd/etc. of another note. Therefore a self-focussed action, i.e. rule or edict, in the note will set an action in another note, for example:

```
$Rule("Some note")=$Text;
```

Assuming the $Name "some note" is unique, this will set that's note's $Rule's code to be the current note's $Text. If the $Name is not unique, consider using the $Path:

```
$Rule("/Stuff/examples/Some note")=$Text;
```

This approach allows use of the $RuleEnabled (or $EdictEnabled) to act as a cut-off and stop propagation of code if the code in the note's $Text is being worked on, e.g. to extend or correct existing code. Once the edit is complete, re-enable and the new code will be passed to the relevant note(s). Although, it is more instinctive to use a rule in this role, unless the code will change often, an edict is a more sensible choice as it runs less often (edict execution). If using these 'enable' attributes, consider adding either/both as additional Displayed Attributes to the Code prototype.

If the reference is to more than one note, a list of offsets is possible:

```
$Rule("Some note;Another note")=$Text;
```

But, if more than two or three matches are desired, a better approach is to use a prototype name as the match. Then, all notes using the prototype are updated.

Storing stamps

Older methods were rendered obsolete by the v9+ feature of the *optional* Hints container. If this is feature used every stamp (including older pre-existing stamps) is represented by a discrete notes using the ' Action' prototype which offers syntax colouring and $Text customisations pertinent writing action code.

A further useful affordance of using Hints is per-stamp notes can be opened as stand-alone note windows with action code syntax and parenthesis completion warnings.

Storing stamps - without a Hints container (legacy only)

Consider using the hints container (above), before using this legacy approach.

If a stamp's code is long, it may be more convenient to write the code in a code note and then make the actual stamp an action() task that executes the $Text of the code note. Thus if a code note 'Test-stamp' held the action code `$Colour="red";`, then a stamp with the code:

```
action($Text("Test-stamp"));
```

when run would result in the stamped note(s) turning red. The example is trivial but shows the technique. Note the offset addressing in the stamp to the code note is case sensitive and should use a unique $Name (or else cite the full, unquoted, $Path to the code note). Local attribute references, i.e. $Color or $ChildCount, are bound to the note being stamped: *it is not possible to reference values in the code note using a designator* .

Usage Tips

Avoid the temptation to make the code note affect itself, e.g. set its own $Rule by copying pasting code from the note's $Text into a $Rule displayed as a Displayed Attribute. Use the code note simply to store/edit the code, then use its $rule t set an attribute in a different note.

The limitation of the above is the result is to overwrite the note's own $Rule, or other action attribute, with the code from the referenced note's $Text. However, if the latter is updated, the changed code is not reflected as the note needing the code is no longer referencing it, as the $Rule has a literal copy of the old code. The fix is to reverse the reference, so the note with the code sets the attribute of note needing to use it. If note "Complex Test" needs to use the code in $Text of "code sample 1", then in "code sample 1" this $Rule is added:

```
$Rule("Complex Test") = $Text
```

The "Complex Test" note's $Rule, if one, makes no reference to "code sample 1". If the code-holding note's $Text is edited, on the next update cycle the $Rule of "Complex Test" is updated. Another variant on this is to use the same technique to set a complex conditional $Rule or $DisplayExpression for a prototype note. Editing the 'source' code will update the prototype and the new code will be inherited by all notes using that prototype.

The context of this process does not just have to be a rule. Other examples:

```
$OnAdd(some note)=$Text(code sample 2);
$AgentQuery(some agent)=$Text(code sample 3);
```

Using eval() and action()

The eval() action can be used to further extend the complexity of what may be done, where an operator does not exist for the task at hand. For example, assume a code note of the above type has written, into another note's $TempString, a literal string value of:

```
"collect_if(all,$MyDate== '"+$MyDate+"',$Name)"
```

See how the attribute value, though a string, includes enclosing quotes. This means that it is possible to delay the evaluation of $MyDate. In the note using the code, a single eval() call on $TempString, `eval($TempString)`, returns the sa string in different form. The literal string value of the current note's $MyDate is inserted into it (this example is on a system using day/month date order) and stripped of enclosing quotes:

```
collect_if(all,$MyDate== '23/7/2011 10:02',$Name)
```

The latter is important as the value of eval($TempString) becomes a string of valid action code. To actually get the result of the collect_if(), it is still necessary to evaluate that code. Thus by nesting a second eval() call, the output of the first call is evaluated afresh. Using `eval(eval($TempString))` gives:

```
Note A;Note C;Note_X
```

which is the output of the collect_if(). As a side note, observe how in the initial string it was necessary to added enclosing single quotes around where the code reference to $MyDate occurred, ensuring the literal value, when inserted, was a properly quoted string literal. Single quotes were used as the overall initial string already used double quotes.

Sometimes the Tinderbox code parser is unclear as to user intent and over-evaluates returning a boolean true/false value (N.B. this underlies queries which under the hood resolve query terms to a boolean value). In such cases, the action() operator generally works but note a *significant difference* between action() and eval(): action() evaluates a complete expression—both left *and* right sides of the '=' assignment—as a mix of concatenated strings and and literal values. By comparison, eval() evaluates *part of* an expression.

Using .each() for loops

The List/Set.each() function can be used to iterate lists. It works like a 'for each' method commonly used in programming and scripting whereby each and every item in the source list is processed using the in-loop code.

Using a list item more than once per loop

Assume your data in a list or set. In this case $MySet contains the values cow/dog/eel. Assume it is necessary to turn that into an HTML select list like this:

```
<select>
<option value="cow">cow</option>
<option value="dog">dog</option>
<option value="eel">eel</option>
</select>
```

Notice how in the output above each source list item is used *more than once*. However, using format() or List.format() you can only wrap each list item once. So other than doing iterative list formatting, which can get complex, .each() offers a way around this. Use the following $Rule, line breaks and indentation are just for clarity:

```
$MyList =;
$MySet.each(X) {
 $MyString = '<option value="'+X+'">'+X+'</option>';
 $MyList = $MyList + $MyString;
};
```

```
$Text = "<select>\n"+$MyList.format("\n")+"\n</select>";
```

Shorter version, without the in-loop caching of the concatenated per-item string (see bullet #3 below for the rationale):

```
$MyList =;
$MySet.each(X) {
 $MyList = $MyList + ("<option>"+X+"</option>");
};
$Text = "<select>\n"+$MyList.format("\n")+"\n</select>";
```

Notes:

- $MyList must be reset (to an empty list) *before* the loop otherwise each time the Rule fire the loop adds to existing $MyList data rather than build a new set.
- Everything between the { } is runs as the loop. Here the closing } has a semi-colon after it as another discrete section of code follows it.
- Ordinarily, one could simply put parentheses around the string literals to concatenate them before passing a single string into $MyList. This works as long as the string literals to concatenate do not contain single or double quotes, or, if double quoted enclosed strings contain single quotes. But, an edge case occurs where single-quote-enclosed strings contain any double-quotes; in such cases an 'interim' string attribute is needed to ensure the sub-strings are correctly concatenated before the data is passed to $MyList.
- As a string attribute only contains one value, adding a new one replaces the old. Thus $MyString does not need to be reset before the loop unlike a set or list such as $MyList.
- Note that for the final concatenation .format() can be used as each list item is used only once in the resulting output. The closing select tag must be preceded with a '\n' line break as the .format()'s join only adds the concatenation string between each item in $MyList and a lone break is needed after the last option close tag.

Detecting first or last loop items

See List/Set.first and List/Set.last.

Adding a loop counter

By default .each() processes every list item. But what if you only want do something with the Nth item or perhaps only even numbered ones? In that case it is necessary to create a loop counter, and iterate in-loop. For instance, the following sums the value of odd-numbered items of $MyList onto the existing value of $MyNumber:

```
$MyCountNum = 0;
$MyList.each(ListItem) {
 if(mod($MyCountNum,2)==0) {
  $MyNumber = MyNumber + ListItem;
 };
 $MyCountNum = $MyCountNum + 1;
};
```

Notes:

- Set the counter ($MyCountNum) to zero (or just reset the attribute) before starting the loop.
- Tinderbox list values number upwards from zero (e.g. as seen if accessing data via the List.at() function). Here, think of list item #1 being `1`, then set $MyCountNum to `1` before the loop and test for `mod($MyCountNum,2)!=0`.
- Note that list.at() is *read-only* so you cannot use code like `$MyList.at($MyCountNum) = ListItem;`

An alternate method is to use the range operator (see example).

Using a path variable

The loop variable can itself be a path and this be used as a variable designator allowing attribute offset addressing inside the loop:

```
$Text=""; collect($Overdue,$Path).each(x){
 $Text = $Text+":"+$Text(x);
};
```

In the above, the 'x' variable is a $Path value and is being used to provide the offset address argument in the loop.

Using a variable within a loop

A variable created by var can be altered from within an .each() loop, as shown here.

Creating an $AttributeReference within a loop

Described here.

Constructing $Attribute references in loops

Tinderbox's loop mechanism using .each() is helpful but one task can be problematic. This where the task is to loop a list of attribute title values and then act upon each attribute in turn.

The basic problem is that if the loop variable is, for example 'X' and current loop value is 'MyString', how to construct the reference $MyString, so as to act on the MyString attribute.

The answer is to use a nested action() call inside the loop, even if that seems counter-intuitive at first sight. For example, consider the task of resetting every attribute in a note's displayed attributes table. The contents of the table are stored the Set-type system attribute $DisplayedAttributes, which can be iterated with .each():

```
$DisplayedAttributes.each(X){...};
```

Now the task is to set X, in each loop, to address the attribute named in the current value using action():

```
$DisplayedAttributes.each(X){
 action("$"+X+"=;")
};
```

In other words, if X is value 'MyString' the action() is *both* concatenating some strings:

```
"$"+"MyString"+"=;"
```

…*and* then evaluating the resulting expression:

```
$MyString=;
```

Note that:

- eval() will not work in this context.
- the contents of the action() must concatenate to form a complete action code expression.

It may be necessary to nest quotes. Consider a list of String-type values. To set each one to a value of 'test':

```
$MyList.each(X){
   action("$"+X+"='test';");
};
```

Note that the target value needs to be enclosed in quotes. As the main expression string already uses double quotes, the string needs to use single quotes. It does not matter if you reverse the nesting. The action would work just as well in the form:

```
action('$'+X+'="test";');
```

But, do ensure the sets of single/double quotes nest correctly.

The action can use loop variable more than once. Take the last scenario, but this time where all the attributes are Set or List type. Here the code is:

```
$DisplayedAttributes.each(X){
 action("$"+X+"=$"+X+"+'test';");
};
```

…i.e. creating an action expression, in-loop, like this for X value 'MyList':

```
$MyList=$MyList+'test';
```

White space in the expression does not matter, as in normal action expressions, but is omitted above for brevity/clarity. The expression could also be:

```
$MyList = $MyList + 'test';
```

…and still work.

Constructing references to values in other notes

The task gets more complex when one or both of the left/right side attributes is not `this` note. Consider this task:

```
$MyString("Some note")=$MyString("Other note");
```

… but in a loop where 'MyString ' is a loop variable value. `action()` is still the approach. However, to avoid confusing the action code parser with too many sets of quotes—whether on one or both sides of the expression—is to pass the offset agument via a variable. This avoids normal consideration of quoting (paths) or not quoting (anything other than paths) the offset values.

For example, consider the task of iterating Displayed Attributes to set the value of each such attribute in "Some note" into those of "Other note":

```
var:string vSource = "Source Note";
var:string vDest = "Destination Note";
var:string vCmd;
$DisplayedAttributes(vSource).each(anAttrib){
 vCmd = '$'+anAttrib+'('+vSource+')=$'+anAttrib+'('+vDest+')');
 action(vCmd);
};
```

Or, in function form:

```
function copyDA(iSource:string,iDest:string) {
 var:string vCmd;
 $DisplayedAttributes(source).each(anAttrib) {
  if(dest==""){
   vCmd ="$"+ anAttrib +"="+ anAttrib +"("+iSource+");";
  }else{
   vCmd ="$"+anAttrib+"("+iDest+")="+anAttrib+"("+iSource+");";
  };
  action(vCmd);
 }
};
// called via code
copyDA("/Notes/Source Note","/Notes/Destination Note");
// or
copyDA("/Notes/Source Note","");
```

In the code calling the function, the source/destination string might themselves be attribute values or variables:

```
copyDA($Name(parent),"");
```

Testing constructed code strings

If encountering errors, it may be useful to incorporate a show() command, which can then be commented out or deleted when all is working. For example:

```
var:string vSource = "Source Note";
var:string vDest = "Destination Note";
var:string vCmd;
$DisplayedAttributes(vSource).each(anAttrib){
 vCmd = '$'+anAttrib+'('+vSource+')=$'+anAttrib+'('+vDest+')');
 action(vCmd);
 show(vCmd);
};
```

That will display, in turn, the literal value of vCmd string for each attribute in Displayed Attributes in turn.

A similar but different approach is to log the data by writing it to the $Text of another note, in this case a note 'log' at root level:

```
var:string vSource = "Source Note";
var:string vDest = "Destination Note";
var:string vCmd;
// clear the log of old data
$Text(/log) =;
$DisplayedAttributes(vSource).each(anAttrib){
 vCmd = '$'+anAttrib+'('+vSource+')=$'+anAttrib+'('+vDest+')');
 action(vCmd);
 $Text(/log) += vCmd;
};
```

Concatenation versus addition

In Tinderbox, the plus symbol (+) is used both to concatenate (join) strings of text *and* to sum figures. Thus

```
$MyString = "My favourite colour is " + $FavouriteColour
$MyNumber = 4 + 3
```

Nothing untoward there, not least as the resulting data type (as implied here by the left side attribute name) helps Tinderbox guess the user's intent. But what about:

```
$MyString = "6" + 4
```

Is it "64", 64, "10" or 10? As the stated attribute is String-type, the result is a string. As one of the right side arguments is a string, then a concatenation is used. If both (all) arguments are numbers, then an addition occurs before being saved as a string:

`$MyString = "6" + 4` gives "64"

`$MyString = 6 + 4` gives "10"

`$MyString = 6 + "4"` gives "64"

`$MyString = "6" + "4"` gives "64"

In earlier versions of Tinderbox, there was no need to quote strings and Tinderbox guessed string/number by context. Since then, as the action code has got richer, it has become harder to divine the user's intent, thus the move to expecting text to be placed in double quotes. Incidentally it also makes it easier to indicate intent with regard to leading/trailing white space.

Thus, when using the '+' operator try to ensure Tinderbox is not having to guess too hard as to the desired outcome.

If necessary use parentheses to help indicate intended operator precedence to Tinderbox if it is getting things wrong.

Dates and Strings

Expressions in which a date is added to a string. In the following the value of $Name is "Created"

`$String=$Name + $Created` <- a string "Created08/07/2010 08:27"

`$String=$Name + " " + $Created` <- a string "Created 08/07/2010 08:27"

In the above example the date time format is the user's host OS short date/time format so will vary by locale. The date shown equates to format($Date,"I t"), i.e. local short date space local time (see format(), date formats). Use format() with date attribute if some other date/time formatting is required.

Note also above how a joining space needs to be provided. Note also that adding a string *to* a date continues to return a date, adding a date *to* a string returns a string:

`$Date+"3 days"` <- a *date*: 3/29/10

`"3 Days "+$Date` <- a *string*: 3 Days 3/26/10

`"3 Days"+" "+$Date` <- a *string*: 3 Days 3/26/10

The first above assumes the string being added contains logical date-related content.

Parentheses as a guide to code execution

Although the order in which code is executed might seem intuitive to the user but this may not necessarily be how Tinderbox sees and interprets it. In such situations, parentheses (normal brackets) can be used to indicate 'sub-expressions'. Execution order of things like this

```
$AttrA($Num1) = $AttrB($Num1) + 1;
```

can be less obvious to Tinderbox than to the user writing them. This is especially true where, like above, attribute expressions are being used as these require evaluating themselves before the main expression is evaluated.

In such circumstances, especially if tests appear to give the wrong answer, consider adding parenthesis as a guide to Tinderbox as to the order of the process. For instance:

```
$AttrA($Num1) = ($AttrB($Num1) + 1);
```

tells Tinderbox to evaluate the whole of the right side before attempting the overall expression.

Parentheses can also be useful in stopping mis-interpretation of multi-term queries including the short-form no-value attribute test, e.g. !$AttributeName. Using `(!$AttributeName)` instead of `!$AttributeName)` can help.

Beware too of data types and lack of clarity of whether a '+' is a (string) concatenation or a (number) addition; see more.

Updates and cascading actions

A configuration can arise where one or more agents look at (query) another agent. Alternatively, an action might trigger a note to be moved, in turn triggering an $OnAdd and so on.

When using 'Update now' from the File menu, all agents are fired once (in $OutlineOrder sequence) as are any action events dependent on the update cycle. In a cascading scenario, it may thus be necessary to call 'Update now' several time to flush an action through to completion, especially if using linked agents where the order of dependence does not follow $OutlineOrder. For instance, if agent C looks at agent B which looks at agent A, a change in A will need 2 updates to flush through to C.

Therefore, if using documents with cascading effects, be aware that a single forced update may not always 'complete' all actions as the user might intuit would occur from a single invocation.

Forced invocation of updates runs all running agents, i.e. regardless of priority setting unless the agent is turned off.

Optimising code for performance

In small TBX files there is rarely a need to consider code optimisation, but as your files grow in note count and complexity and acquire more agents you may see the time-to update increase, i.e. how quickly code changes are effected. In such circumstances, or just as best practice, it is worth reviewing and applying a few optimisation techniques.

General practices

These do not directly affect code execution (other than by bad syntax) but help make everything else easier.

Prototypes

Using prototypes to set up and/or locate discrete sets of note(s) is a big productivity gain. Editing code (or code notes) once can then affect many notes. Consider using $RuleDisabled and $DisplayExpressionDisabled in the prototype so as t suppress execution of such code in the prototype. If applying prototypes as part of incremental formalisation, do not forget to reset the inheritance of any prototype-using note's attributes that may have previously had a local value set.

Check your syntax

Make sure your code does not include deprecated syntax. At some point in the future the code may not work at all and in the meantime you're making Tinderbox guess your real intent. Tinderbox does not warn about the existence deprecate code (it can't know what things are that the parser doesn't 'understand'), so updating code is an exercise left to the user.

Act on the right subject: originals vs. aliases

Be aware that if using actions in agents, you are acting on the original, so values of intrinsic attributes

Optimisation

These techniques can help with tuning performance once the size and complexity of your TBX grows:

- **Narrow the scope of agent search**. Review and rewrite queries so each successive query terms tests fewer items. The fewer tests, in aggregate that Tinderbox has to make the faster the results. This is especially true if the (final) test is based on a complex regular expression such as in .contains(). If the desired matching group of notes is not easily defined in narrow scope, consider giving them all the same prototype.
- **Querying existing agents**. Rather than have numerous agent all querying the same overall scope, consider making one agent to to that first selection and then have other agents query that agent's results using different queries. So rather than have 5 agents querying descendedFrom("X"), make a single agent, e.g. 'agent A', do the latter and the the other use inside("agent A") as their first query term. Do note though that this means it may take several agent cycles for a change to cycle through.
- **Agent Priority ($AgentPriority)**. Not every agent needs to run every cycle. The agent's priority can be set via the agent's Action Inspector's Query sub-tab or directly in code via $AgentPriority. Also consider an agent to find all agents allowing you to easily adjust their $AgentPriority via its action. Or make a stamp to toggle $AgentPriority on/off.
- **Re-use agents as part of queries**. Queries can match aliases in other agents. If using multiple agents all with the same query start and differing final filter, it is more efficient to use one agent to find the initial query scope and then mak the other agents query that one agent for the extra term. Consider an 10 agents testing the whole TBX of 1,000 notes, to match the same 100 notes then testing those to get the actual full matches. That is 1,000+100 x 10, i.e. 11,000 test in total. Now if one agent does the first match (1,000 tests) and 10 agents test its 100 aliases, we get 1,000 + (100 x 10) to give 2,000 test—just 18% of the original number of tests per agent cycle. This isn't apparent in small files, but the effects pile up as a document grows especially if the user is keen on using agents for everything. If employing this chaining, be aware of the possibility of cascade effects for which to allow.
- **Caching expression results in a user attribute**. Expressions like Display Expressions and Hover Expressions are, slower to run than rules or agent actions; this is especially so if the expression is anything but a simple attribute value call. As a result, if such expressions are complex, it is better to use a rule or agent action to run the code and store the expression's result in a user attribute and then have the display/hover expression simply read that stored value. Adding an extra attribute has negligible impact on the TBX.
- **Use $Searchable to exclude notes**. The $Searchable attribute provides an ad hoc way to exclude some notes from all searches.
- **Make the action remove acted on notes from query scope**. Consider making the action's outcome make changes such that the altered note no longer meets the initial query. This is a good method for making an agent effectively ac once only on a note. If it is not possible to alter an attribute value being matched, e.g. it is $Text that must not be changed, consider either moving the note (via $Container), or making a user Boolean as a 'guard' field to be tested in the query for the action to set (e.g. an $IsCorrected).

Checking and setting Time correctly in Date data

Tinderbox Date-type data *always* includes a time element (hours:minutes:seconds), even if not explicitly set by the user. If not supplied when first set, the current OS system clock time is used for the item element, even if the date element is fo a different day.

Use of seconds in time. Seconds are set (previously, in older releases, seconds were not set); if not set explicitly seconds are assumed to be '00'. Even if seconds are not of direct interest (but consider rounding to minutes) and not normally displayed in Displayed Attributes, it is it advisable to set a seconds value of '00' when defining time for Date-type data: that should avoid unexpected effects.

The time element of dates makes *some* date comparisons work in unexpected ways.

Methods to see/check a Date's time:

- Visually in a Displayed Attribute displayed in a text window
- Exported as a string using a date formatting string that includes the time.
- Via time-specific action codes:
  - time(theDate)
  - Date.hour
  - Date.minute
  - Date.second

To set or change the time element of a date:

- Manually edit it a Displayed Attribute displayed in a text window
- Set a new Date/time via date(string)
- Set a new Date/time via date(year,month,day,hour,min)
- Via time-specific action codes:
  - set just the hours element of time via Date.hour
  - set just the minutes element of time via Date.minute
  - set just the seconds element of time via Date.second
  - set the whole time element via time(theDate,hours,minutes,seconds)

When working with date designators, and explicit (24-hour clock) time can be added stated. At 14:23:00 on 28 November 2012:

`date("today")` gives 28 November 2012 14:23:00

`date("today - 2 days")` gives 26 November 2012 14:23:00

But

`date("today 23:59:00")` gives 28 November 2012 23:59:00

`date("today - 2 days 08:00:00")` gives 26 November 2012 08:00:00

If wanting to use greater than or less than date comparisons, it can be helpful to explicitly set a time value like 00:00:00 or 23:59:00 if wanting to test for a full day midnight-to-midnight.

Another context where (re-)setting a fixed time can be useful is with timelines and historical dates, for instance setting all times to 08:00:00 or some other common time, if only to avoid effectively random times set automatically when creating dates by supplying on the date and no time. Tidying up existing dates is not hard. Make an agent to find the notes needing tidying, and that add an action like:

```
$StartDate = time($StartDate,08,00)
```

Note that as the code force-changes just the hours/minutes the actual date cannot be affected. For existing data, you can run the above and delete the agent. However, if you have processes creating new events it is worth checking date designator offsets—like the examples above—also use an explicit time before you delete or turn off the agent.

Debugging user-written Action code

Remember that output of any action code is normally to set/alter one or more attributes. Here are a few things learned from testing complex action & export code…

Tiny steps *save time in the end—especially when things are not going as they ought.*

If an initial attempt at a piece of code does not work then consider the following:

- Use small deliberate test documents
- Break the code into discrete steps
- Be aware of context of execution and addressing
- Re-check documentation
- Using code notes for large code sections.
- Store interim code values in variables or attributes
- Store constant values in a note
- Viewing interim values via placards
- Using a log file to review interim code outputs
- Comment code and attribute descriptions
- Annotate your process if not simple
- Use the Tinderbox Inspector's tabs
- Abstract re-used patterns to functions
- Is it really a bug?
- There was a crash or hang
- Export code

Use small deliberate test documents

If adding code into a current working document (i.e. with lots of content) doesn't work as expected, and having checked for obvious errors like typos or wrong literal path values, a sensible approach is to make test document.

If the code features being tried are new it is a good idea to make as simple a test as possible to give confidence. Don't overlook using methods available to show interim values. Use as little test data as possible to start with and validate the basic method works. Then, especially if agent queries or find() are involved, add extra content to test false positive/negatives.

Once confident the basic process works, code can be moved back into the min working document with confidence that it ought to work. If it still fails, this would suggest code/processes in the existing document may be interfering. The Tinderbox Inspector's tabs ca help get an overview as to how many active rules/edicts there are and as to running. It can help to turn off or disable some of these whilst looking for a cause of interference.

In a large mature document with a lot of code, a seeming failure may not be the code as such but the fact it runs less quickly/often than expected. In such a case the work shifts from de-bugging to tuning performance.

Tuning performance

This is not de-bugging as such but involves getting a clear view of how much code is running, where, and with what priority. consider that actions can be turned off (they retain their last used child aliases) and rules/edicts can be disabled. Rule code can move to edicts, that run less frequently, or even moved into a stamp.

Again, it may help to make a small test document of your general workflow just to look for possible optimisations.

Be ready to delete test documents

When done, it is often a good idea to delete the test document straight away or soon after the code works in the main document. In other words, make test documents with the intent of deleting them once the problem is resolved unless you have a clear need. If you do keep a test document it is worth adding a note explaining what it was for and what parts of the bigger context it did/did not test.

Break the code into discrete steps

An action can be as simple as setting a new $Color value, but in reality it may involve multiple expressions (think discrete actions in the same stored bit of code. Also remember that dot-operators condense a whole series of tasks into one expression.

Test as the smallest scope you can. Once confident the values resulting are as expected, all the parts can be joined back together. A common mis-step when debugging a failing action is not to tear down far enough and in so doing miss the step where a failure is occurring.

Chained operators

So, an expression might get a list of data via a collect using a custom find() query, .sort() it and de-duplicate it via .unique() and then format it as a custom string via .format(). Essentially, there are five activities there any of which can be tested separately:

- the find()'s query. This could be tested using an agent. Does the agent find the expected items.
- what collect() is collecting. This could be tested via an agent action.
- the remaining 3 tasks can be split into new expressions. By using a variable or attribute to inspect interim values (or displaying them) the other tasks can be broken into discrete action expressions.

Agents

If using agents with an agent action, it is a good idea first to check that the agent is finding (aliasing) the correct notes before then adding the action code. Also consider turning the agent off whilst actively editing the action/rule/edict.

Rules and edicts

If editing code, disable rules and edicts so that half-written code is not run. Important to note is that if editing a prototype's rule or edict, it is better to remove all the code, edit it in a code note and re-insert when done. This is because $RuleDisabled and $EdictDisabled are intrinsic and **not** inherited. with long/complex actions especially, this increases the chances of large numbers of notes trying to run incomplete code.

A common issue with prototype use is that during test/tune of a rule/edict the code gets edited in an inheriting note (wrong!) rather than in the source prototype. A query for use of the prototype (in $Prototype) & hasLocalValue() can help. For notes using prototype "Person" this query will match all notes with that prototype which have locally set $Rule value:

```
$Prototype=="Person" & hasLocalValue("Rule")
```

This ought to match no notes but if it does match any, those notes need their rule reset (`$Rule=;`). Edicts—indeed, any attribute—can be checked for broken inheritance.

Be aware of context of execution and addressing

Often, as a document grows, action code moves beyond code acting on the current note and it either/both calls or sets attribute values in other notes, via offset addressing.

When addressing an offset by $Name be aware that the title might not be unique or even use the same names as a designator or operator. Using $Path addressing is better in most simple cases, with advanced users benefitting for using $IDString (replacing previous use of $ID) for unambiguous addressing—even if $IDString is hard to use by 'eye' when debugging. But, if using logging, it is easy to log an $IDString and its associated $Path.

As agent actions act on aliases, it is important to under stand the differing contextual meaning of parent vs. agent or adornment. Similarly, be aware of the notion of intrinsic attribute values when reading/setting vales.

When using queries in find() operations be aware of the differing implications of this vs that.

Re-check documentation

Be aware not all operators evaluate all arguments, or to the same depth—i.e. if the arguments included nested evaluations. Argument evaluation is not formally annotated for all operators, the full operator listing probably has the most usefu collection of knowledge on this aspect.

If evaluation is an issue, consider tear-down testing starting with literal values and slowly changing to evaluated arguments.

Using code notes for large code sections.

For long section of code, consider storing the code in a code note, so that it is easier to review and work upon.

Store interim code values in variables or attributes

Historically, this was done by storing data in attributes. That works but if, for instance, this involves storing lots of really large lists, it is easy for unneeded stored data to accumulate unless the attribute values are reset.

Better is to use variables unless there is a genuine need to persistently store such data. Very occasionally, and less encountered today, it may prove necessary to 'crystallise' a calculated value before using it in another expression. Typed variables generally remove that older requirement.

Variables have he advantage of clearing themselves up. They are created when needed during a a complex action, where there may be all sorts of different interim values being created. At the end pf the code running the variables are destroyed.

To record a variable's value for review after running the code, logging is the best approach.

Store constant values in a note

If there are values, e.g. the path to a particular note or a lookup list, that code in multiple notes may use, consider storing such values in attributes of a single note. Some users refer to this a a configuration or 'config' note. Its only purpose is to be a consistent place (path) to store information likely needed for regular reference by code.

Viewing interim values via placards

An alternative to using a log is to show logging massages in the Tinderbox UI, is via message placards. As each message shows for a few seconds and all are shown sequentially, if using a lot of logging, saving the messages to a log note makes more sense. Both methods can be mixed. For instance a placard might report passing key stages in the process whilst logging might capture for more fine-grained detail.

Using a log file to review interim code outputs

Tinderbox has no fixed logging system but this is easily set up in a way that suits the user simply by writing code values to the $Text of a log, i.e. another note. Thus

Make a note. 'log' is a simple default name. Ideally put is somewhere easy to find and avoid a complex name and/or pat that might make addressing the note difficult. Placing it at root of the outline is a good initial choice, i.e. $Name 'log', $Path '/log'.

In the code to be tested write to the log by altering $Text(/log):

```
$Text(/log) = "Current value of vPath is: "+vPath+"\n";
```

Be aware using '=' overwrites existing log text, so an increment '+=' makes more sense, as does an opening call at top of the code to 'reset' the log removing old content. so at start of the code being tested:

```
$Text(/log) =;
```

Then later in the code:

```
$Text(/log) = "Current value of vPath is: "+vPath+"\n";
```

In this case, after the code is run, the log should record the value of variable 'vPath'. A mix of literal string and a value is useful for indicating the code ran. For instance, you may see the literal text but no value indicating the code at least executed up to the logging point but the value assumed to be in the variable vPath has in fact not been set.

In a long-lived document is it easy to go a stage further and make user functions to reset to log or to write to it. For writing , that takes care of remembering to forma the input, add a line break, etc.

If using logging in code it is a good idea to comment out the log call or remove it when done and to check the log file and delete accidentally accrued content.

Comment code and attribute descriptions

Don't overlook that action code supports comments, as what makes sense today might not do so tomorrow or in a year's time.

Similarly all user attributes allow a free text description in the Inspector to record the purpose of attribute. Note that existing controls show the data type, default value, etc. so the description is more useful for capturing *why* the attribute was needed—even if only for ad hoc testing purposes.

For user attribute, take a moment to give them names that are self-explanatory. Clarity of naming is subjective. There is no single right way, so that actual names are a choice for the individual.

Annotate your process if not simple

As a document grows over time and in scale, code accumulates, as does process (such as import form other apps).

Whilst actual action code can hold comments, it is also worthwhile making some notes about overall process. Ideally store these notes away from the document working content, and capture enough detail to inform future self or collaborators as the the overall intent if the automation in the process: why code/functions were added, the steps to do certain processes, etc.

These notes will repay in any latter debugging in the note.

Do *update* the notes as the document ensures, so that they are not out of date when/if needed for future reference. It is much easier to capture this information at the time the process is implemented than trying to remember detail later.

Use the Tinderbox Inspector's tabs

Most useful is the second tab, Agents & Rules. This is particularly helpful when looking at performance-like issues

Abstract re-used patterns to functions

More for advanced users, but be open to using user functions. for instance is a code, or codes in several different notes (or prototypes) are calling the same sequence of actions, consider abstracting the task into a function.

By using a function, the process is written and tested once, with the only varying element being the inputs passed to the function. The more steps to the common process, the more the pay-off as any changes only need to be made tested in the function code and not in all the notes using the function (or the code in the note that function replaced).

Is it really a bug?

If trying to move code tested in a small file into a mature file and hitting issues, it can be useful to restart the application or just close and re-open the current document. Odd as it may seem it pays not to overlook the 'turn it off and on again'

advice.

An easy hole into which to fall is to appear to have a repeatable bug or glitch that occurs consistently in the current document, only for a document re-open to magically clear the issue. The latter is not at all common but a point to bear in minc before writing in for support.

If after testing the code there is still an unexpected outcome, next try to replicate it in a new file with only as much content, if any, as it needed to re-create the criteria to produce the outcome. If still reproducible, please email Tinderbox Tech Support (tinderbox@eastgate.com) with:

- A description of the problem.
- Exact steps to recreate the problem.
- What was expected.
- What actually occurred.
- If possible a test document that can be tested using the described steps to show the issue. Note that TBXs zip-compress well for email transfer.

There was a crash or hang

Crashes are self-evident—the app closes immediately and a crash log is recorded by the OS (occasionally, and extremely unusually, no log may get made). An alternate scenario is the app hangs and is 'unresponsive'. After waiting a bit the general resource is to Force Quit that app, which generates a h'ang' log. Inconveniently, macOS stores the two types of logs in different places: how to find crash and hang logs .

Crashes are not expected and remarkably rare in Tinderbox. However, once having tripped over a crash condition, repeating the cause of the crash will reliably repeat the crash. Importantly, repeating the scenario does not mean the app is now 'more crashy' but the same crash condition is being reliably completed.

Given the large toolbox that is Tinderbox, it is remarkably stable. Given the possible number of combinations of tool, task and context, it is vey possible that a user is the first person to use that combination. The fact a problem is repeatable after it is found does not imply the context is common, even if intuition makes us think otherwise.

Encountering a crash, after restart take a back-up of your active TBX. Normally a crash, if it occurs, the document might loose the last few seconds of unsaved input but generally there is no widespread harm to data. So either use the current document as a back-up or restore a recent file as a last, good reference. Do this before testing further.

At this point either pass the crash log to Tinderbox technical support (email tinderbox@eastgate.com) or continue with diagnosis.

Note that the user-to-user forum cannot diagnose crashes. Given that crashes are contextual, the fact that other users my not see the same problem is not unusual.

Useful things to investigate—if sufficient time/expertise:

- is the problem repeatable:
  - does it re-occur if the app/doc are restarted?
  - does it occur in all documents or only some, or just one?
  - describe the steps to recreate he problem (either in a new/text document or in the affected document(s).
  - if possible share a copy of the/an affected document (TBX files compress well if zipped).

Meanwhile…

Avoid doing whatever causes the crash (if the trigger is clear). Depending on the feasibility of doing so, and if the crash arises from a new release, consider rolling back to a previous version. Providing clear information to Eastgate will increase the chances of early resolution.

Export code

With regard to Export code:

- as with action code be aware of context. From which note's attributes is the code reading?
- includes (`^include()^`) cannot draw data from a note that is set to not export itself. Learn to understand the nuances of $HTMLDontExport and $HTMLExportChildren.
- links in exported pages cannot be created to the content of notes exported as an inline include within another note (i.e. a composite()).

Getting section numbering via Action code

Export code includes a code ^sectionNumber^ which gives output like this `2.6.2.3.42`. It is effectively a chaining all the $SiblingOrder values for ancestors and that of the current note.

This can be achieved in action code but is somewhat inelegant as root level notes require a different handling and action code conditionals require a full expression inside the branch. This code sets $MyString to the section number of the current object:

```
$MyString = if($OutlineDepth >1){
   collect(ancestor,$SiblingOrder).reverse.format(".")+"."+$SiblingOrder;
}else{
   $SiblingOrder
};
```

Or, in one-line form:

```
$MyString = if($OutlineDepth >1){collect(ancestor,$SiblingOrder).reverse.format(".")+"."+$SiblingOrder;}else{$SiblingOrder};
```

Other approaches

There are several ways to number (parts of) a document whether internally or (only) for export. These techniques use one or both of two user attributes:

The technique uses two user attributes:

- $Section. String-type.
- $IsTop. Boolean-type. Regardless of where in the outline the 'top' or root note of the numbered sections needs to have $IsTop set to `true`.

The techniques are:

- Bottom-up
- Top-down
- Just in time

Bottom-up

In this action based approach, notes look upward for seeding section numbering.

Code

```
if($IsTop) {
     $Section="";
} else {     $Section = $Section(parent);     if ($Section) {$Section+=". ";}
    $Section += $SiblingOrder;
};
```

Logic

- If you are the top, you do not have a $Section number.
- Otherwise:
  - Start by getting the parent's $Section value
  - If that value is not empty, add a period
  - Then add your sibling number and set as $Section

Pros and Cons

- GOOD:
  - Stays up to date as the TBX is reorganised
  - Simple
  - Individual notes can choose to behave differently
- BAD:
  - Can fall a few seconds behind setting $Section in big/complex TBXs.
  - Might be a bad choice for huge outlines (thousands of elements)
  - If individual notes behave differently today, that might not be wanted tomorrow.

Top-down

A recursing function using no user attributes. Start at the top, then name all children and so on down.

Code

```
    function fSectionName(iNote, iSectionValue){
        $Section(iNote) = iSectionValue;
        var:string vName = iSectionValue;
```

```
        if (vName != ""){
            vName = vName + ":";
        };
        $Path(children(iNote)).each(aNote){
            fSectionName(aNote, (vName+$SiblingOrder(aNote)));
        };
    };
```

Logic

- If the top, section value is already known.
- Then, tell each child:
  - This is your name.
  - Now name your own children

Pros and Cons

- GOOD:
  - Parent may know best how to name its children
- BAD:
  - Then again, maybe not.

Just in time

A recursing function. Figure out the section name only when actually needed.

Code

```
function fAsk(iName){
    if($IsTop(iName)){
        return "";
    };
    if($IsTop(parent(iName))){
        return $SiblingOrder;
    }
return fAsk($Path(parent(iName)))+"."+$SiblingOrder;
};
```

Logic

- If the top, $IsTop should be `true`
- If parent is the top, return $SiblingOrder
- Otherwise,
  - ask the parent for their name,
  - and then add "." and own $SiblingOrder.

Pros and Cons

- GOOD:
  - Only computes the section numbers when needed
  - Recursion is elegant
- BAD:
  - Recursion can be tricky

Moving notes by setting $Container

$Container holds the '/' terminated path to the current note. The attribute can be set by the user and in doing so the note is moved to a new parent container. Such an approach is useful if processing notes that then can be archived away fron still-current content.

Using $Container with agent actions. Care needs to be taken as agents act on aliases. Therefore an agent action seeking to move a note must act on the *original* of the note, i.e. via $Container(original).

Tinderbox is forgiving if a $Container is set with no trailing "/". For example, this agent action will work:

```
$Container="/work/projects";
```

However, this is the correct way:

```
$Container="/work/projects/";
```

The error (in the first example above) if the last container in the target path is a variable. For example:

```
$Container="/work/"+$MyString(agent);
```

Luckily that does work but should really be :

```
$Container="/work/projects/"+$MyString(agent)+"/";
```

Strings vs. StyledStrings in operator documentation

Many dot-operators are intended to work on a literal string, the value of a String-type attribute or an expression resolving to a string. There are a small number of exceptions to this where the operator applies rich text (RTF) styles to its output.

Closing styles

If this code is used:

```
$Text = "This is " + "some".strike + " text.";
```

The result is "This is **some text**." This occurs because as with normal typing in $Text if a style is turned on the user must also turn it off. The the end of $Text is bolded, any new text added is also bolded. The way to avoid run in it to apply '.plain' to the next added segment.

```
$Text = "This is " + "some".strike + " text.".plain;
```

The result is "This is **some** text." There is currently no way to add a plain 'closure' without applying it to at least one character. But adding tot to punctuation, e.g. `".".plain`, or invisible characters , e.g. `"\n".plain` is possible.

Passing styled text

At present, $Text is the only attribute capable of holding such styled text. String attributes and action code variables, i.e. var, cannot store styled text. If passed styled text the latter will store the un-styled version of the text being passed.

Thus, if we create a styled string and pass it direct to a RTF-capable String type, it works:

```
$Text = "This is " + "some".strike + " text".plain;
```

The result is a $Text value of a "This is ~~some~~ text". Also if $MyString is "some":

```
$Text = "This is " + $MyString.strike + " text".plain;
```

But if we try this:

```
$MyString = "This is " + "some".strike + " text".plain; $Text = $MyString;
```

…the result is a $Text value of an *un-styled* "This is some text". So, a String-type attribute cannot hold styled text. The same is true if a variable is use for hold the result:

```
var vStr = "This is " + "some".strike + " text".plain; $Text = vStr;
```

…again, the result is a $Text value of an *un-styled* "This is some text".

Type coercion, strings and numbers

From v9.5.0, when attempting to reason about the type of the result of an expression in ambiguous situations, Tinderbox examines the values passed to binary operators like "+". If a value is interpretable as a number, Tinderbox prefers the numeric interpretation to a string. The rules for "interpretable as a number" were, however, too lax, and expressions such as

```
"1883 – " + "1964"
```

were treated as numeric because (the opening) part of the first string is a number.

Now, the string is regarded as a numeric value only if the entire string can be parsed as a number.

If ambiguity is obvious to the user, it is preferable to use intermediate attributes or typed variables in order to avoid the ambiguity entirely and to signal user intent clearly to Tinderbox.

Links

Links connect two notes or a note with an external (web or local) resource. There are two discrete forms of link origin, basic links (linked at note level) and text links (based on a selected section of text) which are described in more detail in th notes linked to below (scroll page to see these links).

External links. As well as internal links, links may be created outside the file to web URLs, RSS feeds, etc. Although these are termed web links—see more on Web links—they can actually link to local resource too, such as DEVONthink or Bookends items.

Inbound links into Tinderbox documents can be made using the Tinderbox URL schema. Note that these will only work if the calling app, Tinderbox and the destination Tinderbox are all on the same computer.

Most Tinderbox objects can use linking:

- Notes: yes
- Container notes: yes
- Separator notes: yes (this was not original intent but can be used)
- Aliases: yes (see extra notes below)
- Agents: yes
- Adornments: no
- Picture adornments: no
- Map backgrounds: no (though the parent container note they represent can be linked to, by using another view window at different scope)

Creating links via the UI

See various methods. Do not forget that it is also possible to use action code to generate (or remove) links.

Aliases

Basic links to and from aliases are supported. This is useful for hierarchical documents like aTbRef that make use of aliases as it enables each alias to be exported as and linked to as a discrete note in its correct location in the document hierarchy. However, be mindful that aliases inherit the original's text and web links and these cannot be added to except by altering the original. More on 'Linking & aliases'. Note below, behaviour of aliases and links during export

Links and duplication

When duplicating a linked object, there are slightly different behaviours for inbound and outbound links:

- **Outbound**. All links (basic, text and web) are all duplicated.
- **Inbound**. Inbound links are not copied. Thus all inbound links remain pointing to the original object. If necessary, they must be re-created for the duplicate object.

Properties of Links

Links are *uni-directional*, from source to destination. Links do not have note-type attributes. Agent queries can interrogate data based on specified link types. Likewise action code can act on the existence of links between notes with optional filtering based on link type.

Every link has 'link type' property. More on link types.

Self-linking

A limitation on a note linking to itself has been removed in order to support some more esoteric uses. In principle, for general use, notes should not be linked to themselves.

Other Aspects of Links:

- Basic Links
- Text Links
- Web Links
- Smart Links and URLs
- Link Types
- Link Comments
- Link Actions
- Link Groups
- Linking & aliases
- Linking to target $Text
- Prototype links
- Visualising links
- Links and Export
- Self-links

Basic Links

A note-to-note link at note level, i.e. not from within the note's text ($Text). It links the entire note with another note, or a text selection in another note. Basic links occur only within the current document, i.e. they can only begin or end at a nc in the current TBX.

Originals and aliases export their own basic links (i.e. aliases can differ), but if an alias has no in/outbound basic links it will export those of the original.

Tinderbox provides many ways to generate basic links:

- The link widget on or next to the selected item in most View pane views.
- The link widget at left of the text pane title bar, but only when there is no selection in $Text (in which case a text link is created).
- Dragging a parked link from the link park widget at the left end of the tab bar of a document window.

Basic links can point to a location in the destination note's $Text.

Prototype assignments are effected as a form of link— see more.

Basic links can be used to self-link.

Text Links

This describes links from a text selection, within the $Text of a note, to something else:

- another note
- a URL (q.v. Web Links)

The text selected forms the link 'anchor' in the originating note and is coloured according to the colours defined in the Document Settings Text pane's Text link color setting.

For the currently selected note, any text (and web) link anchors can be highlighted by pressing the Cmd+Opt keys: link anchors are temporarily 'double-underscore' underlined as the keys are pressed and the underlining is removed when the keys are released. The latter feature can be disabled via the Underline on ⌘⌥ option in Document Settings Text pane.

For text links where the anchor text is the name of the destination note, the ' ziplinks' method allows rapid creation of text links via the keyboard input (i.e. typing) alone. This replaces the older Quicklinks method allows rapid addition of links.

If text is added or removed within the original anchor the anchor expands or contracts accordingly. If all anchor text deleted the link is deleted completely.

If the anchor is at the end of a line or paragraph, typing more text may extend the anchor when this is not required. In such circumstances, press Opt+spacebar. The closes the anchor and adds a space after the anchor. If a space is not required immediately following the anchor, after adding the ⌥+Spacebar add the desired character, then left arrow key and backspace (delete) key to remove the space and thus allow the anchor to stop in the middle of a word or just before punctuation.

Links can point to a location in the destination note's $Text.

When renaming notes, Tinderbox finds and updates any text links that link to the renamed note and are anchored to the note's old name.

Text links can be used to self-link.

- Text link creation via the Ziplinking method
- $Name and text link anchor updating
- Quick Links

Text link creation via the Ziplinking method

Firstly, some important contextual points re this technique:

- the ziplinking method *creates* normal text links, i.e. originating from anchor text in the note's $Text
- the method is optimised for key-board-only use, i.e. for the touch typist, as a text link can be created without use of the trackpad or mouse
- *there is no specific type of link that is a 'ziplink'* . Once created, the link is indistinguishable from all other text links. Indeed, once the ziplinking method has been used, the link anchor is no different from any other text link anchor, i.e. the ziplinking method does not create a special type link. Thinking of 'ziplinks' is a path to confusion
- this method is an updated replacement for the old Quick Links method of rapid link creation.

This method is particularly useful for adding references to glossary terms, frequently-used sources, or oft-mentioned people and places; in fact anything where the destination note name makes logical anchor text for the link.

Creating a new link (and possibly note) using the ziplinks method

To use this method to make a named note, precede the name of the note with two square brackets and follow its name with two square brackets like this `[[like this]]`. If there is no note with this name in the document, Tinderbox will create one as the *sibling* of the note being editing creating a text link with the new note's $Name as the anchor text. Otherwise, if that note exists, a text link is created to the matched note.

Notes created via this method observe document settings to expand vertically or horizontally if additional space is required for the name. In addition, such notes are positioned to avoid adornments as well as other notes.

The new note's $CreatedFrom attribute contains the path to the note that created the note (i.e. the source of the creating text link).

Option-clicking in the note name popup menu autocompletes the chosen target but leaves the creation process open for further editing. For example, after typing `[[tar` and then choosing "target" from the popup while pressing the option key, the text will read `[[target` and wait for you to complete the link.

The menu Format ▸ Text ▸ Ziplinks toggles whether the selected note(s) allow link creation via the ziplinking method ($Ziplinks).

Keystrokes and syntax for text link creation via the Ziplinking method

Once links creation is invoked the following logic applies:

`[[` shows a list of all notes in the current document.

`[[D` shows a list of all notes with names that contain a "d" (case-insensitive match).

In documents with many notes it makes sense to type a few characters to avoid excessive numbers of possible matches being listed.

When typing link input, Tinderbox will show a pop-up list of related note names to the left of the text pane; the list displays all notes that contain the input string, not just those that begin with the string. Choose any of these listed notes to mak the link (without further typing); or, press the up-arrow or down-arrow keys to select the preferred target note then press the Return key (↩) to complete the link. In more detail:

- Pressing down-arrow (↓) selects the first target, and pressing down-arrow (↓) again selects the second target.
- Return (↩) completes the link to the selected target.
- Option-Return (⌥+↩) is now accepted as equivalent to option-clicking on the list; the target is copied to the current link but the link remains open, allowing you to add other options.
- When typing link input, you can choose a listed target option by pressing Tab key (⇥) if either:
  - there is only one choice in the match menu
  - an item in the pop-up menu has been selected with the up-arrow and down-arrow keys.
- When typing the start of link input, e.g. `[[A`, Tinderbox shows all notes in the document starting with A. Typing `[[/container/A`, Tinderbox shows only notes in the specified container. To see a list of just those notes that are siblings of this note, type `[[./A` (note the full stop before the slash character). For more see section 'Linking to a note in a specified container', below.
- When typing link input, the popup menu displays names of containers in bold. Selecting a container from the match menu does not close the link input immediately. Instead, a new menu of items inside the container appears. Type closing brackets "]]" to link to the container, or select any note or container inside that container from the menu.
- When typing link input, Tinderbox prefers to link to the original note rather than an alias. Pressing the Tab key (⇥) selects the option currently selected in the popup menu. If no item is selected but there is only one item in the menu, pressing the Tab key (⇥) selects it. If there are several items in the menu but none is selected, the Tab key (⇥) selects the first item.
- Each item in the match popup menu shows a tool tip with the note's full path.

Linking to a note in a specified container

If wishing to use the method make a link to a note in a different container, type the container path in the link input:

```
[[/container/note]]
```

Additional logical syntax applies when typing paths:

`[[/` shows a list of all top-level containers.

`[[/D` shows a list of all top-level notes that begin with a "d" (case-insensitive match).

`[[/Dog/` shows a list of all notes in /Dog.

`[[/Dog/R` shows a list of all notes in /Dog that begin with "r".

`[[./` shows a list of all siblings of this note.

`[[../` shows the parent of this note and all the parent's siblings.

Locale-based sorting of target option listings . When typing link input on an English locale system, diacritics are ignored when looking for matches. However, the match popup menu takes into account the user's locale, so 'étude' will sort appropriately for French users and 'Ångstrom' will sort correctly in Norway. Bear in mind the process works off the OS locale so if working under one locale and using accented characters from a different language/locale sorting is still locale based.

Link type. A link created via this method always has no link type assigned, or rather the type "*untitled" (i.e. no link type). If needed, use the Browse Links or Roadmap dialogs to set a different link type.

Adding anchor text for the new link

The method syntax allows for specifying link anchor text, using a pipe ('|') symbol delimiter. The following will replace the brackets with the anchor text and link it to the designated note.:

```
[[note name|anchor text]]
```

If also needing to specify additional $Text for the note (see below), it goes after the optional anchor text:

```
[[note name|anchor::text]]
```

Adding text to the target note

To aid rapid note-taking, it is possible to also set or add to the $Text of a the link's target note:

```
[[note::some new text]]
```

If the note already exists, the new text is appended to the note's existing $Text. Otherwise, the supplied text forms the entire $Text of the new note; where no text is supplied, new notes generated by this method have no $Text.

There is no limit on the size of the link input string. Previously it was limited to 50 chars. If needing to add more target text than is currently supported, there is a simple alternative. Use the link method to create a link without adding any text. Then click the link. This navigates to the link target. Now add the desired $Text. Then use shortcut ⌘+' to navigate back to the original note. These navigation controls are also accessible from the Note menu.

Target note name and anchor text

WARNING: *even if not using the ziplinking link-creation feature* , changing a note name will affect link anchors pointing to that note, as described here.

Making a backlink from the link target

The method can create backlinks from the destination note. [[<that note>]] will create a link to "that note", and then will append the name of this note to the text of "that note" and link the name back to this note. The mark-up needed is used a a wrapper around the entire input string. Thus:

```
[[<New Idea>]]
```

…will create a new note called 'New Idea' and link to it. If this method is mixed with other syntax variations, like so:

```
[[<New Idea|great idea::Recent innovation>]]
```

…will create a new note called 'New Idea' and link to it via anchor text 'great idea' and add a backlink with the anchor text 'Recent innovation'.

IMPORTANT: note how the closing '>' comes *at the end*, immediately before the closing ']]' and not before the '|' as might be assumed if all the different parts of syntax were just chained.

Stamps and use of the Ziplinking method

If a stamp changes the text, Tinderbox will immediately expand any ziplinking-style mark-up found in the revised text; i.e. it will parse the syntax and use the ziplinks method to create new text link(s).

Suppressing the Ziplinking feature

It can be useful to disable this feature in notes using code examples or square brackets, and the system attribute $Ziplinks supports this behaviour. Built-in prototypes, such as ' HTML' and ' Code' have $Ziplinks set to `false` for just such a reason.

Quick links behaviour, in older versions

The previous ' Quick links' behaviour using a "[[" trigger is described separately.

Presetting prototypes and locations

These 9.5.0 features can also be used when naming new notes.

Prototypes. The zip syntax '[[bridge#pStructure]]' makes a link to note 'bridge' and sets its prototype to 'pStructure'. If no such prototype exists, a new prototype is made in the default, i.e. `/Prototypes`, Prototypes container. Specify a note's prototype by adding the prototype name after the # sign. For example, naming a note "Henderson#Person" will name the note "Henderson" and assign it a prototype of "Person". If no such prototype exists, one will be created in /Prototypes. If the prototype container /Prototypes does not exist, it will be created.

Addresses. The zip syntax '[[bridge@Brooklyn]]' makes a link to note 'bridge'. If a note named 'Brooklyn' exists, 'bridge' takes its $Address from 'Brooklyn'. If no such note exists, the $Address of 'bridge' address is set to that of 'Brooklyn'. Specify a note's location by adding its address, or the name of a note that represents the location, after the @ sign. For example, "Eastgate@134 Main St., Watertown MA USA" would create a note named Eastgate and set its address, and a note named "John@Boston Office" would make a note named "John" and assign the Address of that note from the address of the note named "Boston Office".

$Name and text link anchor updating

When a note is renamed, Tinderbox scans other notes in the document for text links that refer to that note by name. If (a) the text link's destination is the renamed note, and (b) the text link's anchor text is the old name of the renamed note. This is often the desired behaviour, but may in some instances not be desirable. Indeed, *be aware as even if you do not use the ziplinks method, there is currently no means to turn off this feature and renaming a note may thus destroy any user set anchors and replace it with the target note's $Name which may not be appropriate.*

There is an option to disable automatic renaming is available in Document Settings: Text: Update after renaming. Automatic renaming may also be disabled on individual notes by setting the value of the new attribute $UpdateTextLinksAfterRename.

This is useful in the context of using the basic form of the ziplinks link method where a text link is based on the target note's $Name.

Quick Links

Warning: this note describes legacy functionality!

Older functionality is replaced by the Ziplinks method. The older behaviour is described below for legacy support.

Old Quick links

When typing in the text pane, a new text link can be added by typing two left brackets "[[" and the first character of the destination note's name (and only the first one); multiple letters can be typed to help cut the number of possible matches. Tinderbox will then display a menu of notes whose title starts with that character. Selecting a note from this menu and clicking Return will generate a text link to that note, using the destination note's name as the link's anchor text.

This method is particularly useful for adding references to glossary terms, frequently-used sources, or oft-mentioned people and places; in fact anything where the destination note name makes logical anchor text for the link.

The link generated is always to the original note, even if aliases exist. Previously this was not always the case and an alias might be selected.

Web Links

Web links are text links that open in a web browser if followed from within Tinderbox. As well as web pages, a URL link can act as a 'mailto:' call or link to local files (using the 'file:///' protocol instead of 'http://'). Basically, a valid URI.

In all cases Finder will dictate whether a browser or some other application, such as a mail client is used to open the URL. This can also be used to use app pseudo-protocols such as are provided by apps like DEVONThink to allow linking to items stored within the app.

To create a web link, select the desired anchor text and then use Note menu ▸ Make Web Link…, or use the [Cmd]+[Opt]+[Ctrl]+L shortcut (note these options are only available if there is a current $Text selection). The result is to display the Create Link dialog with the URL input box displayed. The Disclosure triangle on the dialog exposes a few extra inputs only pertinent to web links: Target, Title and class. If any of the latter three inputs are populated, during HTML export they are exported as extra non-default HTML link attributes.

The same extra data can also be viewed/edited in the Browse Links dialog.

A web link cannot be created at note level (i.e. as can a basic link) but setting the $URL attribute is akin to doing this. If more than one such 'link' is needed, the user should add extra URL-type user attributes.

The Substitutions menu, smart links can be used to make RTF web links in $Text but whilst these can be used to open web links (or other protocols as above) these links are not recognised by Tinderbox as the they are only embedded in the RTF text.

Note that export code ^outboundWebLinks^ only lists actual Tinderbox web links and not URL-type attributes or smart links.

Smart Links and URLs

Tinderbox uses underlying Apple frameworks to automatically create 'Smart' Links (also call Smart URLs) where URLs are detected in $Text. Automatic detection of such URLs is enabled by default. It can be disabled using Edit ▸ Substitutions ▸ Smart Links. The setting applies to the current note rather than per-note and only applies while the note is still selected: on losing focus the setting returns to the default 'on' state.

To control Smart Link creation more persistently, use the $SmartLinks attribute. This preserves state and can be inherited via prototypes, etc. Templates notes are configured in this manner via their prototype as such link creation is unwantec in a template context.

Unless disabled in the Edit ▸ Substitutions menu or $SmartLinks, if Tinderbox encounters a valid URL being typed or pasted into $Text is will create a usable Web links. Previously, these links only worked from within (the RTF of) $Text, but now any detected URLs are added as full Tinderbox links and may be seen/edited in the note's Browse Links pop-over.

When Tinderbox adopts a smart link found in $Text as a Tinderbox web link, it gives sets the path of the new web link to *untitled (i.e. no (visible) link type).

From v9.5.0, Smart Links are now disabled in the built-in prototypes for Code and Action. This prevents expressions like `$MyString.at(0)` from being treated as URLs in Austria.

From v9.6.0, when Tinderbox converts a smart link in the text to a Tinderbox web link, it now assumes that the appropriate scheme is https if no scheme was specified.

Link Types

Every link has 'link type' property. The link type fulfils two tasks:

- The link's purpose (semantics). It is possible to link the same notes with multiple links, each with differing link types. The latter allows for encoding semantics into the hypertext and which internal actions or export code can then interpret.
- Visual labelling. For major views supporting link lines (Map, Timeline), the link's link type is used as the visible label for a given link.

Link types are managed by the Document Inspector's Links tab. Tinderbox supplies a default list of built-in link types but the user can delete these and/or add their own link types as desired. The revised link type list is then stored in the curre document (only) as part of its list on 'known' link types.

The colour and visibility of a link can be set through the Document Inspector Links tab. In addition to the link type default labels created Tinderbox, the user may add their own via the Links tab, or on the fly when creating links or reviewing them.

The link type is applied when first creating a link or may be altered later in the Browse Links pop-over.

The default is for new links to have no link type. Whilst there is no visibly label, such a link is actually—for computing purposes—of link type "*untitled" ("untitled" in some lists/dialogs).

The default link types are:

- *untitled (default value). The default value of '*untitled' results in no visible label and no type being set. N.B. the asterisk prefix to the name.
- agree
- clarify
- disagree
- example
- exception
- note
- note+. (used for reverse links to 'note' links in a pre-v6 Tinderbox feature).
- prototype (prototype links are generally hidden and uncounted)
- response
- update

Although the above are the defaults, the values used in the pop-up lists in Create Link and Browse Links can be edited via the Document Inspector Links tab. All built-in values except 'prototype' and '*untitled' may be deleted.

User supplied values are allowed and can be entered at the Link Types pane or simply added by typing in the 'type' box when creating/editing links.

Link Types are applied manually when creating Links or may be inherited if set in prototypes. Using a prototype note for a new note does not automatically create a 'prototype' link though this might intuitively seem the case.

During a session for any given TBX, when links are manually created, Tinderbox will initially leave the link type box blank in the Create Link dialog. Once a listed link type is chosen, or a new type added manually, the dialog will remember the last used value to make it easy to build link chains of a given type. The last-used type resets to nothing for each new session (i.e. it is not stored when the TBX is closed).

Action operators supporting link type filters.

Export codes supporting link type filters.

Link Comments

It is possible to add plain-text textual comments to individual links. For instance, this may be a comment about the purpose or intent of the link. The comments can only be seen in the UI in the following pop-overs or stand-alone dialogs:

- Browse Links pop-over
- Roadmap stand-alone dialog (Note: but *not* in the Roadmap pop-over)

All the above have an additional box allowing display of the (optional) link comment for the currently selected link. The box also allows the addition/editing/deletion of link comments.

Using action code, link comments can also be retrieved using eachLink(), but note that this is read-only; the link comment's text cannot be edited via this method.

Link comments are plain text; styling of text (bold, italic, etc.) in *not* supported. Line breaks, e.g. to make paragraphs, are allowed. Use the **Return** (↩) key in dialogs but in pop-overs, such as Browse Links, it is necessary to use **Option**+**Return** (⌥+↩) as the Return key alone will close the pop-over.

Link comments can be accessed only via the UI elements listed above and cannot be accessed via automation, such as action or export code.

Link Actions

It is possible to define per-link-type actions, using the Links Inspector. The action is run whenever a link of that type is *created*; note that the action is associated with an entire link type as opposed to single link. It is not possible to define su an action for a single link (unless that link has its own link type and it is used only once—which seems unlikely).

Changing the link type of an existing link does **not** invoke the OnLink action. There is no associated OnLink attribute as the action code is stored as part of the links's data

There is no reverse effect, as in an action that fires when a link is deleted.

When running the OnLink action, the designator source is bound to the link's source note and the designator destination is bound to the link's destination note. The designator this is also bound to the source note, though using 'source' seer to offer greater clarity as to the code purpose.

This action has no associated attribute: the action is a document-level setting and can only be entered via the Links Inspector.

The code is only run once and the OnLink action can be though of as equivalent to applying a stamp to a newly-linked note. Indeed, OnLink code could just consist of code to run stamp(), if so desired.

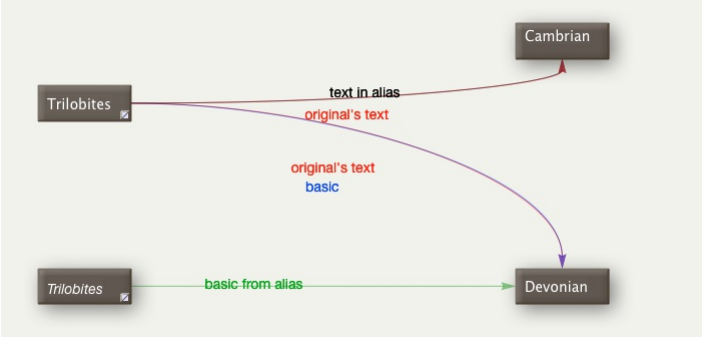
The linkType XML uses a revised format from that previously to allow line breaks in OnLink actions. Note that this change is not backward-compatible before b545, i.e. needs v9.2.0 or later)

Link Groups

This describes various sets of links attached to a note, i.e. the 'link groups' can be addressed when exporting to HTML. These are:

- Basic Links. All note-level outbound links for the current note. Calculated by the export code ^basicLinks^. These may also be thought of as the 'Outbound Links' within the context of intra-TBX links.
- Inbound Links. All note-level inbound links for the current note. Calculated by the export code ^inboundLinks^.
- Ancestor Links. Links created from the current note to each direct predecessor back to the first top level ancestor. Calculated by the export code ^ancestors^.
- Sibling Links. Links created to all notes sharing the same parent as the current note. Calculated by the export code ^siblings^.
- Child Links. Links created to all direct children (i.e. one level down) from the current note. Calculated by the export code ^childLinks^.

Linking & aliases



Whilst the content of an alias (text, most attributes, etc.) is shared with the original note on which the alias is based, its inbound and outbound (basic) links belong to the alias itself:

- basic links: discrete to (each) alias and discrete from the original's.
- text/web links:- shared with original

Thus whilst basic links created to/from an alias are discrete to that alias, text links always belong to the original and are drawn as such on maps. Given these differences, notes that aliases support their own discrete Roadmap view and Brows Links dialog. Be aware that text windows' link counts are always for the original. Aliases have no text window as such, opening their text window simply opens that of its original.

For map use, an upshot of the latter is that maps of aliases, e.g. maps of agent contents can never show text links, only the aliases' own basic links.

Making a copy of alias of another alias does not copy that alias's basic links. The new alias will start with no basic links but does inherit those of its original during export if it has no basic links of its own.

Aliases are (HTML) exported as separate entities linking in a fashion appropriate to their location within the export folders. In an export context, the alias note's text still has the original's outbound text links but its own basic links.

The links() operator only investigates links to/from original notes, thus those from aliases are ignored.

Linking to target $Text

Basic and text links may be made to terminate a specific place in a note (a target $Text 'anchor'), including the note from which the link originates, i.e. self-link. This can done only via the drag-drop method of link creation, done thus:

- Start the link:
  - Basic Link: click-drag from the text pane link widget to the tab bar link park.
  - Text Link: select some $Text (the source anchor text) and click-drag from the text pane link widget to the tab bar link park.
- Select the target note.
- Find the target text anchor and select it, if necessary scrolling the target $Text to find the desired passage.
- Click-drag from the tab bar link park and drop anywhere on the text selection. Do not worry if the link line is not visible all the way to the drop point.
- The link is ready to finish, and the Link Creation pop-up is shown.

NOTE: for both the link well and link park, if you click-hold on the control you will see the destination link pop-up. The latter cannot be used to this type. If it appears, restart the process by clicking outside the pop-up and click-dragging withou a delay to invoke the linking process rather than the pop-up.

To indicate a specific destination of the link, select the source and drag the text link to the link parking space. Then, select the destination and scroll so the destination text is visible. Finally, drag the link out of the parking space and click at the destination location. When the link is followed, Tinderbox will scroll the text view so the destination text is visible and highlights the word at the destination. The highlights are automatically removed after 3 seconds, or when the note's $Text is edited, whichever happens first.

Note these additional behaviours:

- When creating links to a span of text that is selected in the text pane or a text window, if the mouse is clicked inside a text selection then the entire selection becomes the target of the link. If the mouse is clicked outside a text selection the target of the link is the character nearest the click.
- When following a text link with a destination span, then entire span is highlighted, not merely the word closest to the click.

Prototype links

A note's relationship with its prototype, i.e. the note name set in $Prototype, as stored as a basic link, i.e. in the TBX's `<links>` linkbase.

Importantly, this form of link is *not drawn or counted* (e.g. as in $ChildCount) as whilst generically a basic link it is essentially 'behind-the-curtain' use. If treated as a normal link it would throw link counts off by one and and noise to maps usin prototypes.

If necessary, the visibility of the 'prototype' link type can be turned on in two contexts:

- Document Inspector, Links tab, where the type can be selected and then ticked to be made visible. 'prototype' link lines will then be visible in Map and Timeline view where pertinent. The setting is global and reversible, i.e. visibility can be turned off again.
- Roadmap. By default this pop-over/stand-alone dialog includes prototype links but their inclusion can be toggled in/out (globally for Roadmap use) via a tick-box.

Visualising links

Visualising Links

Only two major view types actually visualise links in the view:

- Map view. Draws all links both starting and ending on the map. Stub links are drawn on objects with links whose source or destination is off the current map. Counts of links are given for such stubs although web links are not included Stubs are not drawn for web links alone.
- Timeline view. Draws all links whose starting and ending objects both lie within the timeline, i.e. all events descended from the view container (so unlike maps showing links that join items at different outline depth). Any links whose source or destination is outside the current timeline are not drawn at all. Unlike maps, link labels are not drawn.

Visualisation of links is controlled via Document Settings/Maps and the settings on the Document Inspector Links tab. Customisation of visualisation is possible down to the granularity of individual links types, but not individual links themselves. Bezier (curved) links can be drawn in a more expressive 'broad' style. Prototype links are not displayed (or counted) by default— see more.

Links and Export

Any basic or text links to other notes are not exported as links if the link destination note is not itself exported as a stand-alone page. A note only exported as an ^include^ to another note does not qualify as a stand-alone page.

It might be assumed the 'parent' of the include would inherit the inbound link but it does not. For documents written with export in mind, this can be an important fact to consider when deciding the granularity of notes as every link destination must be a discrete note/page.

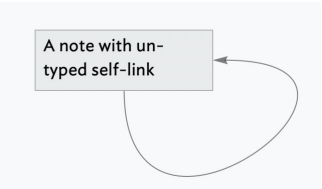
On export only, aliases without basic links of their own will export the basic links of their original, *if* any such exist.

Hierarchical Links

If you choose, Tinderbox can create additional links from text on each exported web page to provide access to the notes that are nearby in the Tinderbox hierarchy. This original feature is discontinued and now, more flexibly, depends on the code added to the HTML template used for export. There is a wide range of such codes.

Self-links

It is possible for basic or text links to link the the current note. Few will have such a need, outside some forms of process mapping, but self-linking is allowed.



Macros

Macros are encapsulated sections of text that may be used to insert larger or more complex sections of text back into the document or into exported pages. They are used in conjunction with the ^do()^ export code or do() action code. The former allows the user to insert the macro's output directly into exported text or HTML (as does do() if enclosed in ^value^. By contrast, internally do() is best used to populate string attributes values via actions and rules.

Macros cannot, generally, evaluate export or action code, so might best be thought of sections of boilerplate text with configurable text inputs. An exception is that ^do()^ may use export code, but only within the context of export from Tinderbox. There is thus a subtle difference in the behaviour of export code vs. action code invoked macros with regard to how inline code is treated.

Macros are managed via the Macro tab of the HTML Inspector.

Macro utility can be used in both queries and actions.

Within macros, the character $ prefixing a number has a special meaning which will be familiar to those who have used regular expressions before. $1 means "the first argument to the macro". This is actually the second argument in do(): the first is always the macro name (same applies to action and export versions). There is no limit to the number of arguments applied, though practicality precludes using excessive numbers; point being that unlike with regular expression back-references you may have more than 9 references, e.g. a $10.

These are applied recursively, so that if we have a macro called "Petshop" whose value is simply: `$1`

Then calling the macro…

```
^do("Petshop","This parrot is $2","dead!")^
```

…will give us…

```
This parrot is dead!
```

…whilst…

```
^do("Petshop","This parrot is $2","just sleeping.")^
```

…will give us…

```
This parrot is just sleeping.
```

In the above example note the need to allow for white space at string joins, so either a space between 'is' and '$2' or at the beginning of the second string.

Another example, take a macro named 'dot' whose value is:

```
<img src="anImage.gif" height="16" width="16" alt="">
```

…is called as…

```
^do("dot")^
```

…and results in…

```
<img src="anImage.gif" height="16" width="16" alt="">
```

As written, this is effectively just boilerplate insertion. To make this a more useful macro and allow it to set the image tag attribute's for name, height, width, and also set the alt to the image name then change the macro's value to:

```
<img src="$1" height="$2" width="$3" alt="$1">
```

…is called as…

```
^do("dot","pic.gif",20,30)^
```

…that results in…

```
<img src="pic.gif" height="20" width="30" alt="pic.gif">
```

Or, if the code is

```
<img src="$1" height="$2" width="$3" alt="$4">
```

…called as…

```
^do("dot","pic.gif",20,30,"caption")^
```

…that results in…

```
<img src="pic.gif" height="20" width="30" alt="caption">
```

If there is no caption, but no width/height, commas indicating a blank input should still be supplied:

```
^do("dot","pic.gif",,,"caption")^
```

…that results in…

```
<img src="pic.gif" height="" width="" alt="caption">
```

Trailing omissions should still be observed, as below where $4 is not supplied:

```
^do("dot","pic.gif",20,30,)^
```

…that results in…

```
<img src="pic.gif" height="20" width="30" alt="">
```

Whereas, a missed alt tag is not a show stopper, providing height/width tags with no value might be mistaken by a browser for a zero value. Better would be this macro code

```
<img src="$1"^if($2)^ height="$2"^endIf^^if($3)^ width="$3"^endIf^ alt="$1">
```

Escaping special macro characters. If you want to remove the special meaning from $ in a macro argument, precede it with a backslash `\$`. Commas in text strings should also be escaped lest they be parsed as argument delimiters, this Similarly, any literal backslash \ must become `\\`. To insert a string like "Prices, $6 \ $16.", change the above macro to:

```
<img src="$1" height="$2" width="$3" alt="$4">
```

…and run as…

```
^do("dot","pic.gif",20,30,"Prices\, \$6 \\ \$16.")^
```

…to get…

```
<img src="pic.gif" height="20" width="30" alt="Prices, $6 \ $16.">
```

Export vs. internal use. In an export context the inline export code in the macro can only be accessed using the `^do()^` export code. Otherwise, all export/action code in the macro or input arguments is treated as literal text (but see the next paragraph below). For in-document actions, use the action code do() operator instead; export code should not be used inline in $Text except for actual export purposes.

Export Codes

This section cover codes specifically used in formatted export. Historically, formatted export was for HTML. Whilst some export codes are closely tied to HTML use, most can be used for generating all sorts of formats, or even things as simple as plain-text CSV or tab-limited files.

Use of export is discussed in the Export section of aTbRef. Also see the Export Inspector and export-related system attributes.

Below is a list of specific 'export codes' using in export template notes:

- Export Codes - Full Listing
- Export Code Types
- Export Code Scope
- Using Export Code
- Altering mark-up for Export Code generated lists
- Debugging user Export code
- Export Code Arguments
- Export codes honouring link types

Export Codes - Full Listing

The full list of Export Codes is split. Much of export code is now done using action code wrapped in a ^value()^ code. Older deprecated codes are not longer listed, *and should not now be used* , although they may still work (in old files for legacy purposes).

Note: meaning of square brackets in code syntax.

Valid Export Codes still in use:

- ^action(action)^
- ^ancestors([start, list-item-prefix, list-item-suffix, end])^
- ^backslashEncode(data)^
- ^basicLinks([start, list-item-prefix, list-item-suffix, end])^
- ^childLinks([start, list-item-prefix, list-item-suffix, end])^
- ^children([template][,N])^
- ^cloud(item, count])^
- ^comment(data)^
- ^descendants([template][,N])^
- ^directory(item)^
- ^do(theMacro [,argument, anotherArgument, etc.])^
- ^docTitle^
- ^documentCloud([count])^
- ^else^
- ^endIf^
- ^file(item)^
- ^host^
- ^if(condition)^
- ^inboundBasicLinks([start, list-item-prefix, list-item-suffix, end, type])^
- ^inboundLinks([start, list-item-prefix, list-item-suffix, end])^
- ^inboundTextLinks([start, list-item-prefix, list-item-suffix, end, type])^
- ^include(item|group[, template])^
- ^indent([data][, N])^
- ^linkTo(item [, data] [,css class])^
- ^nextSibling^
- ^outboundBasicLinks([start, list-item-prefix, list-item-suffix, end, type])^
- ^outboundLinks([start, list-item-prefix, list-item-suffix, end])^
- ^outboundTextLinks([start, list-item-prefix, list-item-suffix, end, type])^
- ^outboundWebLinks([start, list-item-prefix, list-item-suffix, end, type])^
- ^paragraphs([item,] N)^
- ^path[(item)]^
- ^previousSibling^
- ^randomChildOf(item)^
- ^randomLine(item)^
- ^root^
- ^sectionCloud([item, count])^
- ^sectionNumber(item)^
- ^setRoot([newRoot])^
- ^siblings([start, list-item-prefix, list-item-suffix, end])^
- ^similarTo(item, count[, start, list-item-prefix, list-item-suffix, end])^
- ^text([item, N, plain])^
- ^title([item])^
- ^url(item)^
- ^value(expression)^
- ^version^

^action(action)^

Export Code Type: Calculation [other codes of this type]
Export Code Scope of Action: n/a [codes with similar scope]
Export Code First Added: Baseline
Export Code Last Altered: As at baseline

^action(action)^ This code performs any action, or list of actions, that could be performed by an agent or rule. For simplicity, consider it as if it were a rule that is only run when a note is exported using the template holding the ^action()^ code. The **action** code is performed immediately. ^action^ does not export anything itself. For example: `^action($Color="red")^ ^value($Color.format())^` will change the colour of the exported note to "red" and export "red". This allows manipulations for export use without needing to, for instance, hold the value in a user attribute first. Also see, action(), ^value(expression)^. The internal equivalent is a normal action/rule statement, i.e. where is no point in using ^action^ internally. However, if you want some logic during export to change an internal attribute value (e.g. exported notes having a new colour), then use ^action^ in the context of your export codes. IMPORTANT: if using ^action()^ in the *text* of an exporting note (i.e. in $Text), remove all line breaks in the enclosed action code. Otherwise a mis-evaluation will occur as each code line will be treated as an HTML paragraph enclosed in `<p></p>` tags.

^ancestors([start, list-item-prefix, list-item-suffix, end])^

Export Code Type: Creation of Links [other codes of this type]
Export Code Scope of Action: item [codes with similar scope]
Export Code First Added: Baseline
Export Code Last Altered: As at baseline

^ancestors(["start", "list-item-prefix", "list-item-suffix", "end"])^
returns links to all of the ancestors of this note (if any). Unlike the pre-v.4.6 format of links in a single line, with colons between the elements, this defaults to an HTML unordered list like other list-generating codes.
If the optional **start**, **list-item-prefix**, **list-item-suffix**, **end** arguments are omitted then the default values of <ul>, <li>, , are used.
This allows you to format the group of links as an HTML list or table in the exact format you want, where " **start**" is quote-enclosed text that will be inserted before all of the links, " **end**" is text that will be inserted after all of the links, " **list-ite prefix**" is text that will be inserted before each link, and " **list-item-suffix**" is text that will be inserted after each link. See more on altering list type or other HTML output.
If "Hierarchical lists" in the HTML pane of the preferences is unchecked, this will not work.
Mark-up (export code) elements embedded in ^ancestors^ are evaluated.
To get the pre-v.4.6 format use:

```
^ancestors("","," : ,"")^
```

Exported links use the CSS class and target based on their underlying Tinderbox link values, if one is specified; the link type value is not picked up as a class name.
Aliases export as discrete notes (if the alias' container is set to ExportChildren) with content of the original but was any contextual hierarchical inks appropriate to the aliases' location. The ^ancestors^ for an alias also export as the ancestors of the alias, not the ancestors of the source note.
Any ancestor set to `$HTMLDontExport=true` is skipped.
Item objects source and destination may be used in expressions to indicate from where data used within the expression's arguments is drawn.
The export is relative to *current* rather than *this*, facilitating work with included files.

^backslashEncode(data)^

Export Code Type: Export Mark-up [other codes of this type]
Export Code Scope of Action: n/a [codes with similar scope]
Export Code First Added: Baseline
Export Code Last Altered: As at baseline

^backslashEncode(data)^
The ^backslashEncode^ code has been added for the convenience of exporting to XML and JavaScript. ^backslashEncode^ prefaces single and double quotation marks with a single backslash.

^basicLinks([start, list-item-prefix, list-item-suffix, end])^

Export Code Type: Creation of Links [other codes of this type]
Export Code Scope of Action: item [codes with similar scope]
Export Code First Added: Baseline
Export Code Last Altered: As at baseline

Note: this is deprecated in favour of ^outboundLinks^, though the behaviour is the same and this method still works, for legacy purposes.
^basicLinks(["start", "list-item-prefix", "list-item-suffix", "end"])^
returns all of the basic links from this note, formatted as an unordered list. The logical counterpart of this code is ^inboundLinks^. This code does not include outbound text links on the presumption that these are already present in the outpu ^text^ of the note. There is no single code to list all outbound links.
If the optional **start**, **list-item-prefix**, **list-item-suffix**, **end** arguments are omitted then the default values of <ul>, <li>, , are used.
This allows you to format the group of links as an HTML list or table in the exact format you want, where " **start**" is text that will be inserted before all of the links, " **end**" is text that will be inserted after all of the links, " **list-item-prefix**" is text will be inserted before each link, and "**list-item-suffix**" is text that will be inserted after each link. See more on altering list type or other HTML output.
Exported links use the CSS class and target based on their underlying Tinderbox link values, if one is specified; the link type value is *not* picked up as a class name.
Mark-up elements embedded in ^basicLinks^ are evaluated.
Note that unlike other link in/outbound link codes this code and ^inboundLinks^ do not offer an optional link type filter.
Item objects source and destination may be used in expressions to indicate from where data used within the expression's arguments is drawn.
Originals and aliases export their own basic links (i.e. aliases can differ), but if an alias has no in/outbound basic links it will export those of the original.
Also see: ^outboundBasicLinks^, ^outboundTextLinks^ and ^outboundWebLinks^.
The export is relative to *current* rather than *this*, facilitating work with included files.

^childLinks([start, list-item-prefix, list-item-suffix, end])^

Export Code Type: Creation of Links [other codes of this type]
Export Code Scope of Action: item [codes with similar scope]
Export Code First Added: Baseline
Export Code Last Altered: As at baseline

^childLinks(["start", "list-item-prefix", "list-item-suffix", "end"])^
returns links to all of the children of this note (if any), formatted as an unordered list. NB! If "Hierarchical lists" in the HTML pane of the preferences is unchecked, this will not work.
If the optional **start**, **list-item-prefix**, **list-item-suffix**, **end** arguments are omitted then the default values of <ul>, <li>, , are used.
This allows you to format the group of links as an HTML list or table in the exact format you want, where " **start**" is text that will be inserted before all of the links, " **end**" is text that will be inserted after all of the links, " **list-item-prefix**" is text will be inserted before each link, and "**list-item-suffix**" is text that will be inserted after each link. See more on altering list type or other HTML output.
Mark-up elements embedded in ^childLinks^ are evaluated.
If "Hierarchical lists" in the HTML pane of the preferences is unchecked, this will not work.
The HTMLDontExport settings of children are respected.
Usually, rules and queries to test whether a note has children will test ChildCount. But ^if(^childLinks^)… is better if you want the expression to be `false` if the note has children BUT none of the children are exported.
Exported links use the CSS class and target based on their underlying Tinderbox link values, if one is specified; the link type value is not picked up as a class name.
Item objects source and destination may be used in expressions to indicate from where data used within the expression's arguments is drawn.
If appears in a note (or its template) which does not export its own children, it instead creates links to the original of each child.
The export is relative to *current* rather than *this*, facilitating work with included files.

^children([template][,N])^

Export Code Type: Data Include [other codes of this type]
Export Code Scope of Action: item [codes with similar scope]
Export Code First Added: Baseline
Export Code Last Altered: As at baseline

^children^

Tells Tinderbox to include the immediate children of the current note (or agent) in the current web page using their default template file.

^children(template)^

Tells Tinderbox to include the descendants of the current note (or agent) in the current Web page using the specified **template** file.

^children(template,N)^

Tells Tinderbox to include only the first **N** immediate children of the current note (or agent) with the specified **template** file. If **N** is used without a **template**, a comma must be placed before N.

This replaces the deprecated ^justChildren(template)^. The original ^children^ code was replaced by ^descendants^.

^children^ evaluates its template argument, allowing agents and containers to use expressions like

```
^children(^value($MyTemplate)^)^
```

The HTMLDontExport settings of children are respected.

IMPORTANT: If you build templates where the exported (HTML) documents where one document includes content from child/other Tinderbox notes, links to the included content from other documents will not get created. Tinderbox is not able to convert those links to #in-page-addresses within the overall target document. The same limitation applies to ^include()^.

^cloud(item, count])^

Export Code Type: Export Mark-up [other codes of this type]
Export Code Scope of Action: item [codes with similar scope]
Export Code First Added: Baseline
Export Code Last Altered: As at baseline

^cloud([item, count])^

This provides an HTML interface to the information displayed in the Get Info dialog's words tab. It displays the 100 most common words in the note, omitting stop words (there may be fewer words if there are fewer valid words in scope). The words are wrapped in tags to permit the most common words to be displayed in a larger font; the tags used may be customised through the new attributes HTMLCloud1Start/End, HTMLCloud2Start/End, etc.

^cloud^ takes two optional arguments:

- **item**. This refers to a specific note by name, path, or by an item object keyword such as *parent*. If *document* is used or item is omitted, the scope is the whole whole document.
- **count**. This specifies the maximum number of common words to be displayed.

Examples:

```
^cloud^
^cloud(this,25)^
^cloud(/this/that/other/note)^
^cloud(Interesting Note, 75)^
```

See also ^sectionCloud^, ^documentCloud^

^comment(data)^

Export Code Type: Data Include [other codes of this type]
Export Code Scope of Action: item [codes with similar scope]
Export Code First Added: Baseline
Export Code Last Altered: As at baseline

^comment(data)^

All of the **data** within the parentheses of the ^comment^ is exported as an HTML comment. Thus:

```
^comment(This is a comment)^
```

exports as an unseen HTML code comment as in:

```
<-- This is a comment -->
```

^descendants([template][,N])^

Export Code Type: Data Include [other codes of this type]
Export Code Scope of Action: item [codes with similar scope]
Export Code First Added: Baseline
Export Code Last Altered: As at baseline

^descendants^

Tells Tinderbox to include the descendants of the current note (or agent) in the current web page using their default template file.

^descendants(template)^

Tells Tinderbox to include the descendants of the current note (or agent) in the current Web page using the specified **template** file.

^descendants(template,N)^

Tells Tinderbox to include only the first **N** descendants of the current note (or agent) with the specified **template** file. If **N** is used without a **template**, a comma must be placed before N. In finding N matches, the first branch is to its tip, then siblings then recursing up branch #1's sub-branches before starting branch #2.

To export immediate children but not the children's descendants, see ^children(template)^. This replaces the old usage of ^children^.

^descendants^ evaluates its template argument, allowing agents and containers to use expressions like

```
^descendants(^value($MyTemplate)^)^
```

The HTMLDontExport settings of children are respected.

^directory(item)^

Export Code Type: Data Include [other codes of this type]
Export Code Scope of Action: item [codes with similar scope]
Export Code First Added: Baseline
Export Code Last Altered: As at baseline

^directory^

Exports the relative path for this web page's file from the top-level directory you selected for storing HTML files to this file (excluding the filename). This equates to $Container in HTML-exported name form.

Example:

For the note exporting to "`/archives/May.html`", ^directory(this)^ generates "`archives/`".

The directory for this note is: "`index/Automating_Tinderbox/Coding/Export_Codes/Export_Codes_-_Full_Listing/`"

^do(theMacro [,argument, anotherArgument, etc.])^

Export Code Type: Data Include [other codes of this type]
Export Code Scope of Action: item [codes with similar scope]
Export Code First Added: Baseline
Export Code Last Altered: As at baseline

^do(theMacro)^

Allows you to use macros in HTML (and XML) export, where **theMacro** is a macro you have defined in Tinderbox.

^do(theMacro [,argument, anotherArgument])^

Allows you to use macros in HTML (and XML) export, where **theMacro** is a macro you have defined in Tinderbox; and **argument** and **anotherArgument** are additional arguments that can be passed to the macro.

There are no limits, within sensible reason, to the number of arguments that may be supplied.

Read about macros for usage details.

Internally, the do() operator may be used.

Unlike do(), and *specifically in the context of export*, ^do^ will evaluate inline export code within the macro's code.

^do^ accepts the **theMacro** argument as a quoted string (to meet current action code usage for string data).

^docTitle^

Export Code Type: Data Property [other codes of this type]
Export Code Scope of Action: document [codes with similar scope]
Export Code First Added: Baseline
Export Code Last Altered: As at baseline

^docTitle^

The name of the Tinderbox document (i.e. the OS filename). The '.tbx' file extension is omitted if present.

Note that this data cannot be retrieved using action code.

The ^docTitle^ of this TBX file is: " `aTbRef-95`".

^documentCloud([count])^

Export Code Type: Export Mark-up [other codes of this type]
Export Code Scope of Action: document [codes with similar scope]
Export Code First Added: Baseline
Export Code Last Altered: As at baseline

^documentCloud([count])^

This provides an HTML interface to the information displayed in the Get Info dialog's words tab. Here the scope is the whole TBX document. It displays the 100 most common words in the note, omitting stop words (there may be fewer words if there are fewer valid words in scope). The words are wrapped in tags to permit the most common words to be displayed in a larger font; the tags used may be customised through the new attributes HTMLCloud1Start/End, HTMLCloud2Start/End, etc.

^documentCloud^ takes one optional arguments:

- **count**. This specifies the maximum number of common words to be displayed.

Examples:

```
^documentCloud^
^documentCloud(50)^
```

See also ^cloud^, ^sectionCloud^

Below is the output from ^documentCloud(25)^. Note that ideally it should be placed in some form of HTML container rather than be allowed to spread across the page otherwise the visual impact is lost. Here anyway is the ^documentCloud(25)^ 'raw' output:

action *attribute* *attributes* *baseline* *code* *current* *data* *default* *item* *list* *may* *name* *note* *notes* *set* *string* *text* *tinderbox* *type* *used* *using* *value* *via* *view* ***weekly***

^else^

Export Code Type: Conditional Mark-up [other codes of this type]
Export Code Scope of Action: n/a [codes with similar scope]
Export Code First Added: Baseline
Export Code Last Altered: As at baseline

^else^

If the condition specified in ^if(condition)^ is not `true`, then export everything from the ^else^ statement to the ^endif^.

Thus:

```
^if($Width>5)^The $Width is greater than 5.^else^The $Width is 5.0 or less.^endIf^
```

...which, depending on the outcome exports out either the string "The $Width is greater than 5." or the string "The $Width is 5.0 or less."

Note there is no extended branching form, such as an 'else if' method. For that, nest further ^if()^ tests in the negative result branch (i.e. after the ^else^).

^endIf^

Export Code Type: Conditional Mark-up [other codes of this type]
Export Code Scope of Action: n/a [codes with similar scope]
Export Code First Added: Baseline
Export Code Last Altered: As at baseline

^endIf^

Indicates the end of the alternatives indicated by the ^if(condition)^ statement. This closing statement is not mandatory if there is no ^else^ branch to the resulting actions. Thus the following are acceptable alternatives:

```
^if(condition)^do stuff[end of template line auto-closes code]
^if(condition)^do stuff^endIf^
```

For the next test a closing ^endIf^ is mandatory:

```
^if(condition)^do stuff^else^do other stuff^endIf^
```

^file(item)^

Export Code Type: Data Property [other codes of this type]
Export Code Scope of Action: item [codes with similar scope]
Export Code First Added: Baseline
Export Code Last Altered: As at baseline

^file(item)^

The filename of the file for the web page for an **item** (i.e. note). This is the combination of $HTMLExportFileName plus $HTMLExportExtension. If no item is supplied, it defaults to value of 'this'. If HTMLExportFileName is not manually set it is created dynamically at export from $Name, with the filename varying according to the length set in $HTMLFileNameMaxLength (for aTbRef it is currently set to: 100).

The export filename of this note (`^file^`) is: "`/index/Automating_Tinderbox/Coding/Export_Codes/Export_Codes_-_Full_Listing/file_item.html`".

The export filename of note "HTMLExportBefore" (`^file(HTMLExportBefore)^`) is:
"`/index/Automating_Tinderbox/Coding/Use_of_Attributes/Attribute_Listings/System_Attribute_List/HTMLExportBefore.html`".

But see that '`^value($HTMLExportFileName+" + "+$HTMLExportExtension)^`' gives: '` + .html`' ← note the missing filename! The latter problem occurs as $HTMLExportFileName is not evaluated until export.

^host^

Export Code Type: Data Property [other codes of this type]
Export Code Scope of Action: document [codes with similar scope]
Export Code First Added: Baseline
Export Code Last Altered: As at baseline

^host^

The host of the site of this web page. This is derived from the URL (Web server) setting in the HTML pane of document Preferences.

The ^host^ value for this TBX file is: "" (i.e. nothing). In this document it is not defined but were it defined for the site's normal domain it would return "https://acrobatfaq.com"

^if(condition)^

Export Code Type: Conditional Mark-up [other codes of this type]
Export Code Scope of Action: n/a [codes with similar scope]
Export Code First Added: Baseline
Export Code Last Altered: As at baseline

---

^if(condition)^
^if(condition)^ ^endIf^
^if(condition)^ ^else^ ^endIf^

Test the condition, resolving to a `true` or `false` result:

- if the **condition** is `true`, then includes everything from the ^if()^ statement to the ^else^ or the ^endif^.
- if the **condition** is `false`, then include nothing, or if an ^else^ branch is offered, include everything after the ^else^ statement to the ^end^.

Though an ^endIf^ closing marker is not mandatory, users—especially if new to export code—are advised to add one as it gives more clarity as to the end of the output of a `true` match to **condition**. If the else/endIf mark-up is omitted the text following the closing)^ of the if tag of the condition is returned up to the end of the current line/paragraph (i.e. up to the next line break in the template source code).

Unlike in very early versions of the app (and thus old examples, the condition statement is a query written in action code (essentially the same syntax as if writing an if() action test. For example,

```
^if($Name.contains("tbx"))^Matched 'tbx' string.^endIf^
```

…is `true` if the note's title contains the string "tbx", and exports the string "Matched 'tbx' string.". For a reversed test:

```
^if(!$Name.contains("tbx"))^No matched 'tbx' string.^endIf^
```

…is `true` if the note's title does *not* contain the string "tbx", and exports the string "No matched 'tbx' string.". For a numerical test:

```
^if($Width>5)^The $Width is greater than 5.^endIf^
```

…is `true` if the note's width exceeds 5.0 and exports out the string "The $Width is greater than 5."

For the first two examples above the value matched can be an actual string or a regular expression.

Using an ^else^ logic branch, different information can be exported for `true` and for `false` **condition** matches. Thus:

```
^if($Width>5)^The $Width is greater than 5.^else^The $Width is 5.0 or less.^endIf^
```

…which, depending on the outcome exports out either the string "The $Width is greater than 5." or the string "The $Width is 5.0 or less."

In the ^if^ condition, the **condition** query is always evaluated in the context of this note.

Note there is no extended branching form, such as an 'else if' method. For that, nest further ^if()^ tests in the negative result branch (i.e. after the ^else^).

^inboundBasicLinks([start, list-item-prefix, list-item-suffix, end, type])^

Export Code Type: Creation of Links [other codes of this type]
Export Code Scope of Action: item [codes with similar scope]
Export Code First Added: Baseline
Export Code Last Altered: As at baseline

---

^inboundBasicLinks(["start", "list-item-prefix", "list-item-suffix", "end", "type"])^

returns all of the notes that have a basic (note-level) link to the current note, formatted as an unordered list, excluding prototype links.

The optional arguments allow you to format the group of links as an HTML list or table in the exact format you want, where **start** is text that will be inserted before all of the links, **end** is text that will be inserted after all of the links, **list-item-prefix** is text that will be inserted before each link, and **list-item-suffix** is text that will be inserted after each link. The type argument restricts the scope of link inclusion and is discussed separately below. See more on altering list type or oth HTML output.

If the optional **start**, **list-item-prefix**, **list-item-suffix**, **end** arguments are omitted then the default values of <ul>, <li>, , are used.

The optional fifth argument, **type**, lets you restrict exported lists to specific link types, as in the named types seen in the TBX's Link Types palette. **type** my be omitted but if you do wish to use it, the preceding 4 arguments must all be specified.

For example:

```
^inboundBasicLinks("<ul>","<li>","</li>","</ul>","example")^
```

will list only inbound links of type "*example*". As the **type** argument is a regular expression, besides specifying a particular link type name, you can specify wildcard characters or lists of eligible link types, etc.

```
^inboundBasicLinks("<ul>","<li>","</li>","</ul>","(example|untitled)")^
```

will list links of the types "*untitled*" and "*example*".

Also see ^inboundLinks^, ^inboundTextLinks^. The logical opposite of this code is ^outboundBasicLinks^.

This code exports CSS class and target based on their underlying Tinderbox link values, assuming these are actually completed; the link type value is *not* picked up as a class name.

If there is more than one inbound link from the same source, said link is only listed once.

New item objects source and destination may be used to indicate from where data used within the arguments is drawn.

The export is relative to *current* rather than *this*, facilitating work with included files.

^inboundLinks([start, list-item-prefix, list-item-suffix, end])^

Export Code Type: Creation of Links [other codes of this type]
Export Code Scope of Action: item [codes with similar scope]
Export Code First Added: Baseline
Export Code Last Altered: As at baseline

---

^inboundLinks(["start", "list-item-prefix", "list-item-suffix", "end"])^

returns all of the notes that have basic links to the current note, formatted as an unordered list, except prototype links. The logical counterpart of this code is ^outboundLinks^. This code does not include inbound text links; unlike the case with ^basicLinks^ (where such links are in the body text and arguable already 'listed'), there is no method to include inbound text links in the overall inbound link list. There is no single code to list all inbound links.

If the optional **start**, **list-item-prefix**, **list-item-suffix**, **end** arguments are omitted then the default values of <ul>, <li>, , are used.

This allows you to format the group of links as an HTML list or table in the exact format you want, where " **start**" is quoted text that will be inserted before all of the links, " **end**" is text that will be inserted after all of the links, " **list-item-prefi** text that will be inserted before each link, and "**list-item-suffix**" is text that will be inserted after each link. See more on altering list type or other HTML output.

Mark-up elements embedded in ^inboundLinks^ are evaluated.

Exported links use the CSS class and target based on their underlying Tinderbox link values, if one is specified; the link type value is *not* picked up as a class name.

Note that unlike other link in/outbound link codes this code and ^outboundLinks^ do not offer an option link type filter.

Item objects source and destination may be used in expressions to indicate from where data used within the expression's arguments is drawn.

Originals and aliases export their own basic links (i.e. aliases can differ), but if an alias has no in/outbound basic links it will export those of the original.

Also see ^inboundBasicLinks^, ^inboundTextLinks^.

The export is relative to *current* rather than *this*, facilitating work with included files.

^inboundTextLinks([start, list-item-prefix, list-item-suffix, end, type])^

Export Code Type: Creation of Links [other codes of this type]
Export Code Scope of Action: item [codes with similar scope]
Export Code First Added: Baseline
Export Code Last Altered: As at baseline

---

^inboundTextLinks(["start", "list-item-prefix", "list-item-suffix", "end", "type"])^

returns all of the notes that have a text link (a link anchored in body text) to the current note, formatted as an unordered list, excluding prototype links.

The optional arguments allow you to format the group of links as an HTML list or table in the exact format you want, where **start** is text that will be inserted before all of the links, **end** is text that will be inserted after all of the links, **list-item-prefix** is text that will be inserted before each link, and **list-item-suffix** is text that will be inserted after each link. The type argument restricts the scope of link inclusion and is discussed separately below. See more on altering list type or oth HTML output.

If the optional **start**, **list-item-prefix**, **list-item-suffix**, **end** arguments are omitted then the default values of <ul>, <li>, , are used.

The optional fifth argument, **type**, lets you restrict exported lists to specific link types, as in the named types seen in the TBX's Link Types palette. **type** my be omitted but if you do wish to use it, the preceding 4 arguments must all be specified. For example:

```
^inboundTextLinks("<ul>","<li>","</li>","</ul>","example")^
```

will list only inbound links of type "*example*". As the **type** argument is a regular expression, besides specifying a particular link type name, you can specify wildcard characters or lists of eligible link types, etc.

```
^inboundTextLinks("<ul>","<li>","</li>","</ul>","(example|untitled)")^
```

will list links of the types "*untitled*" and "*example*".

Also see ^inboundLinks^, ^inboundBasicLinks^. The logical opposite of this code is ^outboundTextLinks^.

This code exports CSS class and target based on their underlying Tinderbox link values, assuming these are actually completed; the link type value is not picked up as a class name.

If there is more than one inbound link from the same source, said link is only listed once.

New item objects source and destination may be used to indicate from where data used within the arguments is drawn.

The export is relative to *current* rather than *this*, facilitating work with included files.

^include(item|group[, template])^

Export Code Type: Data Include [other codes of this type]
Export Code Scope of Action: group [codes with similar scope]
Export Code First Added: Baseline
Export Code Last Altered: 9.5.0

---

^include(item|group)^

Includes a single **item** or **group** notes, using their default template(s), in the web page for the current note. The scope may also be set from an attribute holding a list of paths or a function returning such a list.

^include(item|group, template)^

Includes the item or group using the specified export **template** rather than each note(s)' default export template. For instance, is a note is set to export a complete page, the default would be inappropriate where the note's output is to be used embedded in another page. In the later case, it is necessary to specify a **template** that simply returns inline content markup.

From v9.5.0, the **template** argument is evaluated. If **template** is not a literal path to or unique name of a template, it will be evaluate to try and resolve a value resulting in a template name. For example:

```
^include(/colophon,^value($MyTemplate)^)^
```

would export the top-level note named "colophon" using the template named in $MyTemplate for the note currently being processed. In other words, $MyString is referenced for the context of the note calling the and not the note referenced by **note**.

Another generalised way to understand this operator is:

^include(which object(s), what content/what form)^

The first argument resolves to telling where Tinderbox should look to pull data in export template form. The second argument overrides the template set for/inherited by the object(s) set via the first argument and allows a different export template to be used to render the data returned via ^include^.

IMPORTANT: Tinderbox will not generate HTML links for internal links that point to notes that are only exported via the ^include()^ method; consider linking to the note's (export) parent note instead. The same implies to other forms of transclusive operator: e.g. ^children()^. Put another way if you build templates where the exported (HTML) documents where one document includes content from child/other Tinderbox notes, links to the included content from other documents will not get created. Tinderbox is not able to convert those links to #in-page-addresses within the overall target document.

Syntax

A quoted string literal can be used with either a single **item** or a **group** (list). A literal **item** reference:

```
^include("a note")^
```

A literal **group** reference via a literal list:

```
^include([a note;another note])^
```

If the **item** or **group** is being assembled via a simple expressions such as string concatenation, an additional ^value()^ wrapper is needed to ensure the value passed to ^include()^ is an actual string or list:

```
^include(^value("Some"+" note")^)^
```
→ include calls the note "Some note"

If a stored attribute value is used, **group** assignments must be enclosed in a ^value()^ code whereas they are not needed for **item** assignments. So, for calls which return a single **item**:

```
^include($MyString)^
^include($MyString("Some note"))^
```

In both cases above, $MyString must hold a note's $Name or its full $Path value. This method is not used for retrieving the value of any attribute to insert into the export. For that you use ^value()^.

Examples for as **group** scope, where the target attribute holds a list of notes and a ^value()^ code encloses the attribute called:

```
^include(^value($MyList)^)^
^include(^value($MyList("Another note"))^)^
```

Expressions that resolve to either **item** or **group** scope must also use a ^value()^ wrapper. The most common use with an expression is a find():

```
^include(^value(find(inside(Some note)))^)^
^include(^value(find($Text.contains("Nelson")))^)^
```

Note, ^include()^ does not mind if the initial input is a list of one (**item**) or many (**group**), but the above variations ensure that the actual data passed into the operator are in the correct format.

Using the Template optional argument

The optional second argument specifies the export **template** to use when including is evaluated. Why might this be needed? In the basic form above, the operator tell the HTML export process to include data from *other* note(s) into the data being created for the currently-processed note.

By default the nature of the included data is set via the target's export template (or the Tinderbox document's default export template). However, it is most often the case with include that only some data, e.g. just a few attributes, are actually required. The **template** argument allows the user to specify a different, specific template for the inclusion task. **template** must be a valid template note in the current TBX document. In this way to get and include values of 3 attributes for 'a note', the process could call:

```
^include("a note", "3_attributes")^
```

Where the template "3_attributes" has the template code:

```
^value($FirstName)^ ^value($LastName)^. Tel: ^value($Phone)^
```

Of course, the same values could also be called as `^value($FirstName("a note"))^`, etc., instead of using ^include()^. Choosing between the methods is a matter of style and the nature of the task.

As with the **item/group** input, template can be specified as a literal string, a string value stored in an attribute or an expression that resolves to a literal string holding a template $Name or $Path. Thus it is possible to have both inputs permits parameterised, constructions as in the following:

```
^include(^value($StringAttribute(ItemNameOrPath))^,^value($TemplateName)^)^
```

^indent([data][, N])^

Export Code Type: Export Mark-up [other codes of this type]
Export Code Scope of Action: item [codes with similar scope]
Export Code First Added: Baseline
Export Code Last Altered: As at baseline

---

^indent([data][, N])^

Exports the **data** as a string, repeated once for each ancestor of the currently exported note (i.e. the equivalent of $OutlineDepth-1 times), or N times if the latter is specified.

^indent^

Exports a tab character for each ancestor of the currently exported note except root level, i.e. tab is the default indent if no **data** argument is supplied.

data. The number of times **data** or a tab is emitted is ($OutlineDepth-1); for aliases this is the alias' value and not that of the original.

By original intent, this code applied to the *(HTML) source* of the exported note, but depending on string used for **data**, this might affect the visual render. Whereas the default `\t` only affects the HTML, a **data** value of `  ` would insert two non-breaking spaced per tab that would render in the output.

N. The default can be superseded by supplying an optional second argument **N**. **N** is evaluated so may either be a number or an expression, e.g. `$OutlineDepth+3`. This kind of thing is useful if trying to correctly indent source code and when the export 'base' is at a level other than the root of the TBX.

If the **N** argument is used, the **data** argument *must* also be supplied. To explicitly specify a tab, as in the normal default, use "\t", e.g. `^indent("\t",6)^`.

For web use, tabs are ignored for white space so consider using one or more " " space HTML entities as the **data** value.

Examples

In the source TBX document this note is at Outline Depth 6, so `^indent(data)^` will emit the **data** value 5 times. Thus, if we use a hyphen as the indent's data value, you see:

-----*Text indented with* `^indent(-)^`.

In the next example the on-screen text should be visibly indented three (non-proportional) spaces:

Text indented with `^indent( )^`.

Note how the visual HTML output of the next two examples is not visibly indented as tabs are not interpreted as white space mark-up in HTML, but are present in the HTML source:

Text indented with `^indent^` (i.e. tab).

Text indented with tabs to outline level plus 4 (i.e. `^value(eval($OutlineDepth+4))^` tabs, thus 10 tabs)

Thus, if you view the HTML source for the page, you will see first of the two examples immediately above has output `6` tabs. For the example following it, you will see `10` tabs have been output; depending on the program viewing the source you may be able to 'see' these as tabs by moving the cursor through the source code with the arrow keys: you will see it jump from tab to tab.

Root level notes are not indented.

^linkTo(item [, data] [,css class])^

Export Code Type: Creation of Links [other codes of this type]
Export Code Scope of Action: item [codes with similar scope]
Export Code First Added: Baseline
Export Code Last Altered: As at baseline

---

^linkTo(item [, data] [,css class])^

Exports the **data** as the anchor text of a link to the web page corresponding to the designated **item** (or note). The **data** can be simply some text or another export function's result and may include HTML mark-up code. The **data** argument i optional, and if not used the name of the destination note is used. The **item** can be a named note, an item object or a path (e.g. /headlines/latest/).

If included, the **css class** element causes the HTML link output to include a *class*="" attribute.

^linkTo()^ exports links from included files *relative to the current page*.

Example link to "Colophon": `^linkTo(Colophon)^` outputs 'Colophon'

Same but with a 'data' argument: `^linkTo(Colophon,Read the Colophon)^` outputs 'Read the Colophon'

Same but with a 'data' and 'css class' argument: `^linkTo(Colophon,Read the Colophon,xref)^` outputs 'Read the Colophon'

To get the relative URL of a note, use ^url()^.

^nextSibling^

Export Code Type: Boolean Comparison [other codes of this type]
Export Code Scope of Action: item [codes with similar scope]
Export Code First Added: Baseline
Export Code Last Altered: As at baseline

---

^nextSibling^

A simple Boolean test for whether the currently processed note has a next sibling (N.B. not always the original's next sibling). This is useful for export context where differing output is required when the current note is the last sibling.

This test also solves the problem of testing for siblings when when exporting agent aliases or mixed original/alias siblings.

Example:

```
^if(^nextSibling^)^…
```

The test is analogous to testing nextSibling or lastSibling in an action code context.

^outboundBasicLinks([start, list-item-prefix, list-item-suffix, end, type])^

Export Code Type: Creation of Links [other codes of this type]
Export Code Scope of Action: item [codes with similar scope]
Export Code First Added: Baseline
Export Code Last Altered: As at baseline

---

^outboundBasicLinks(["start", "list-item-prefix", "list-item-suffix", "end", "type"])^

returns an unordered list, excluding prototype links, to all of the notes to which the current note has basic (note-level) links.

The optional arguments allow you to format the group of links as an HTML list or table in the exact format you want, where **start** is text that will be inserted before all of the links, **end** is text that will be inserted after all of the links, **list-item-prefix** is text that will be inserted before each link, and **list-item-suffix** is text that will be inserted after each link. The type argument restricts the scope of link inclusion and is discussed separately below. See more on altering list type or oth HTML output.

If the optional **start**, **list-item-prefix**, **list-item-suffix**, **end** arguments are omitted then the default values of <ul>, <li>, , are used.

The optional fifth argument, **type**, lets you restrict exported lists to specific link types, as in the named types seen in the TBX's Link Types palette. **type** my be omitted but if you do wish to use it, the preceding 4 arguments must all be specified. For example:

```
^outboundBasicLinks("<ul>","<li>","</li>","</ul>","example")^
```

will list only inbound links of type "*example*". As the **type** argument is a regular expression, besides specifying a particular link type name, you can specify wildcard characters or lists of eligible link types, etc.

```
^outboundBasicLinks("<ul>","<li>","</li>","</ul>","(example|untitled)")^
```

will list links of the types "*untitled*" and "*example*".

Also see ^outboundTextLinks^, ^outboundWebLinks^. The logical opposite of this code is ^inboundBasicLinks^.

This code exports CSS class and target based on their underlying Tinderbox link values, assuming these are actually completed; the link type value is *not* picked up as a class name.

New item objects source and destination may be used to indicate from where data used within the arguments is drawn.

The export is relative to *current* rather than *this*, facilitating work with included files.

^outboundLinks([start, list-item-prefix, list-item-suffix, end])^

Export Code Type: Creation of Links [other codes of this type]
Export Code Scope of Action: item [codes with similar scope]
Export Code First Added: Baseline
Export Code Last Altered: As at baseline

---

^outboundLinks(["start", "list-item-prefix", "list-item-suffix", "end"])^

returns all of the basic links from this note, formatted as an unordered list. The logical counterpart of this code is ^inboundLinks^. This replaces the now-deprecated ^basicLinks^, though the functionality is the same. This code does not include outbound text links on the presumption that these are already present in the output ^text^ of the note. There is no single code to list all outbound links.

If the optional **start**, **list-item-prefix**, **list-item-suffix**, **end** arguments are omitted then the default values of <ul>, <li>, , are used.

This allows you to format the group of links as an HTML list or table in the exact format you want, where "**start**" is text that will be inserted before all of the links, "**end**" is text that will be inserted after all of the links, "**list-item-prefix**" is text will be inserted before each link, and "**list-item-suffix**" is text that will be inserted after each link. See more on altering list type or other HTML output.

Exported links use the CSS class and target based on their underlying Tinderbox link values, if one is specified; the link type value is *not* picked up as a class name.

Mark-up elements embedded in ^basicLinks^ are evaluated.

Note that unlike other link in/outbound link codes this code and ^inboundLinks^ do not offer an optional link type filter.

Item objects source and destination may be used in expressions to indicate from where data used within the expression's arguments is drawn.

Originals and aliases export their own basic links (i.e. aliases can differ), but if an alias has no in/outbound basic links it will export those of the original.

Also see: ^outboundBasicLinks^, ^outboundTextLinks^ and ^outboundWebLinks^.

The export is relative to *current* rather than *this*, facilitating work with included files.

^outboundTextLinks([start, list-item-prefix, list-item-suffix, end, type])^

Export Code Type: Creation of Links [other codes of this type]
Export Code Scope of Action: item [codes with similar scope]
Export Code First Added: Baseline
Export Code Last Altered: As at baseline

---

^outboundTextLinks(["start", "list-item-prefix", "list-item-suffix", "end, "type"])^

returns an unordered list, excluding prototype links, to all of the notes to which the current note has text links (links anchored in body text) and web links.

The optional arguments allow you to format the group of links as an HTML list or table in the exact format you want, where **start** is text that will be inserted before all of the links, **end** is text that will be inserted after all of the links, **list-item-prefix** is text that will be inserted before each link, and **list-item-suffix** is text that will be inserted after each link. The type argument restricts the scope of link inclusion and is discussed separately below. See more on altering list type or oth HTML output.

If the optional **start**, **list-item-prefix**, **list-item-suffix**, **end** arguments are omitted then the default values of <ul>, <li>, , are used.

The optional fifth argument, **type**, lets you restrict exported lists to specific link types, as in the named types seen in the TBX's Link Types palette. **type** my be omitted but if you do wish to use it, the preceding 4 arguments must all be specified. For example:

```
^outboundTextLinks("<ul>","<li>","</li>","</ul>","example")^
```

will list only inbound links of type "*example*". As the **type** argument is a regular expression, besides specifying a particular link type name, you can specify wildcard characters or lists of eligible link types, etc.

```
^outboundTextLinks("<ul>","<li>","</li>","</ul>","(example|untitled)")^
```

will list links of the types "*untitled*" and "*example*".

The is no option to show only the text links without the web links. However, careful use of different link types for text vs. web links would allow filtering via **type**.

Also see ^outboundBasicLinks^, ^outboundWebLinks^. The logical opposite of this code is ^inboundTextLinks^.

This code exports CSS class and target based on their underlying Tinderbox link values, assuming these are actually completed; the link type value is *not* picked up as a class name.

New item objects source and destination may be used to indicate from where data used within the arguments is drawn.

- ^outboundBasicLinks([start, list-item-prefix, list-item-suffix, end, type])^
- Links tab
- ^outboundWebLinks([start, list-item-prefix, list-item-suffix, end, type])^
- destination
- ^inboundTextLinks([start, list-item-prefix, list-item-suffix, end, type])^
- source
- Regular Expression usage
- Altering mark-up for Export Code generated lists

always uses the original note, even when this is an alias.

The export is relative to *current* rather than *this*, facilitating work with included files.

^outboundWebLinks([start, list-item-prefix, list-item-suffix, end, type])^

Export Code Type: Creation of Links [other codes of this type]
Export Code Scope of Action: item [codes with similar scope]
Export Code First Added: Baseline
Export Code Last Altered: As at baseline

---

^outboundWebLinks(["start", "list-item-prefix", "list-item-suffix", "end", "type"])^

returns an unordered list, of all the web (i.e. external) links from the current note.

The optional arguments allow you to format the group of links as an HTML list or table in the exact format you want, where **start** is text that will be inserted before all of the links, **end** is text that will be inserted after all of the links, **list-item-prefix** is text that will be inserted before each link, and **list-item-suffix** is text that will be inserted after each link. The type argument restricts the scope of link inclusion and is discussed separately below. See more on altering list type or oth HTML output.

If the optional **start**, **list-item-prefix**, **list-item-suffix**, **end** arguments are omitted then the default values of <ul>, <li>, , are used.

The optional fifth argument, **type**, lets you restrict exported lists to specific link types, as in the named types seen in the TBX's Link Types palette. **type** my be omitted but if you do wish to use it, the preceding 4 arguments must all be specified. For example:

```
^outboundWebLinks("<ul>","<li>","</li>","</ul>","example")^
```

will list only inbound links of type "*example*". As the **type** argument is a regular expression, besides specifying a particular link type name, you can specify wildcard characters or lists of eligible link types, etc.

```
^outboundWebLinks("<ul>","<li>","</li>","</ul>","(example|untitled)")^
```

will list links of the types "*untitled*" and "*example*".

Also see ^outboundBasicLinks^, ^outboundTextLinks^. For self evident reasons, there is no inbound equivalent of this code.

This code exports CSS class and target based on their underlying Tinderbox link values, assuming these are actually completed; the link type value is *not* picked up as a class name.

New item objects source and destination may be used to indicate from where data used within the arguments is drawn.

The text anchor is the note anchor text; previously it was the document name).

^outboundWebLinks^ always uses on the original note, even when this is an alias.

The export is relative to *current* rather than *this*, facilitating work with included files.

^paragraphs([item,] N)^

Export Code Type: Data Include [other codes of this type]
Export Code Scope of Action: item [codes with similar scope]
Export Code First Added: Baseline
Export Code Last Altered: As at baseline

---

^paragraphs([item,] N)^

Exports the first **N** paragraphs of the text of specified **item** (note). If **item** is not specified, ^paragraphs(N)^ equates to ^paragraphs(this, N)^.

This method effectively allows insertion of only part of ^text^ whilst using the same inline evaluations for mark-up.

Alternatively, for unevaluated $Text excerpts use ^value($Text.paragraphs(N))^. See ^value^, String.paragraphs.

^path[(item)]^

Export Code Type: Export Mark-up [other codes of this type]
Export Code Scope of Action: document [codes with similar scope]
Export Code First Added: Baseline
Export Code Last Altered: As at baseline

---

^path[(item)]^

Exports the relative path for this web page's file from the top-level directory you selected for storing HTML files to this file (including the filename). This equates to $Path in HTML-exported name form. Optionally, **item** may be supplied, such a note name^path(some note)^ or a designator ^path(original)^.

This note's `^path^` is: "`index/Automating_Tinderbox/Coding/Export_Codes/Export_Codes_-_Full_Listing/path_item.html`".

This note's `^path(nextSibling)^` is: "`index/Automating_Tinderbox/Coding/Export_Codes/Export_Codes_-_Full_Listing/previousSibling.html`".

^previousSibling^

Export Code Type: Boolean Comparison [other codes of this type]
Export Code Scope of Action: item [codes with similar scope]
Export Code First Added: Baseline
Export Code Last Altered: As at baseline

---

^previousSibling^

A simple Boolean test for whether the currently processed note has a previous sibling (N.B. not always the original's previous sibling). This is useful for export context where differing output is required when the current note is the first sibling.

This test also solves the problem of testing for siblings when when exporting agent aliases or mixed original/alias siblings.

Example:

```
^if(^previousSibling^)^…
```

The test is analogous to testing prevSibling or firstSibling in an action code context.

^randomChildOf(item)^

Export Code Type: Data Include [other codes of this type]
Export Code Scope of Action: item [codes with similar scope]
Export Code First Added: Baseline
Export Code Last Altered: As at baseline

---

^randomChildOf(item, template)^

Includes an export of a random child of the specified **item** or note through its default/specified **template**.

Internally, the randomChild item object may be used.

^randomLine(item)^

Export Code Type: Data Include [other codes of this type]
Export Code Scope of Action: item [codes with similar scope]
Export Code First Added: Baseline
Export Code Last Altered: As at baseline

^randomLine(item)^

Scans the text of a designated **item** or note (or, if no argument is supplied, the current note). The text is split at paragraph breaks (e.g. carriage returns), and one line (i.e. paragraph) is chosen at random.

Here is a random line from note "Specimen text for test":-

This is a sentence. And another. Yet more. The first paragraph ends now. Now paragraph two follows. It has sentences too. Note very many, and it stops here. The third paragraph is not different. It too has sentences. It then finishes. ^do(Check)^ The fourth is the last paragraph. It completes here.

^root^

Export Code Type: Export Mark-up [other codes of this type]
Export Code Scope of Action: document [codes with similar scope]
Export Code First Added: Baseline
Export Code Last Altered: As at baseline

^root^

Returns the relative path from the current notes's exported file to the directory that contains the cover page. The relative paths generated will still work during internal preview (i.e. without any export being generated).

When concatenating ^root^ with a path, *do not* add a trailing (folder) slash as this is created as part of the ^root^ output. Thus use:

```
^root^images/icon.gif
```

…and not…

```
^root^/images/icon.gif
```
WRONG!

Note that this is not necessarily the same as ^root(this)^, used in the case where a note (this) is embedded inside another note that is included in a page.

The ^root^ of this file's source note is: (see your web browser's URL bar to see how this resolves in overall URL terms.).

For additional finer control of relative path calculation, consider use of ^setRoot()^.

^sectionCloud([item, count])^

Export Code Type: Export Mark-up [other codes of this type]
Export Code Scope of Action: group [codes with similar scope]
Export Code First Added: Baseline
Export Code Last Altered: As at baseline

^sectionCloud([item, count])^

This provides an HTML interface to the information displayed in the Get Info dialog's words tab. In this case the scope is 'section' (parent as indicated by item). It displays the 100 most common words in the note, omitting stop words (there may be fewer words if there are fewer valid words in scope). The words are wrapped in tags to permit the most common words to be displayed in a larger font; the tags used may be customised through the new attributes HTMLCloud1Start/End, HTMLCloud2Start/End, etc.

^sectionCloud^ takes two optional arguments:

- **item**. This refers to a specific note by name, path, or by an object keyword such as **parent**.
- **count**. This specifies the maximum number of common words to be displayed.

Examples:

```
^sectionCloud^
^ sectionCloud(this,60)^
^ sectionCloud(/this/that/other/note)^
^ sectionCloud(Interesting Note, 40)^
```

See also ^cloud^, ^documentCloud^

^sectionNumber(item)^

Export Code Type: Data Property [other codes of this type]
Export Code Scope of Action: item [codes with similar scope]
Export Code First Added: Baseline
Export Code Last Altered: As at baseline

^sectionNumber(item)^

The hierarchical number of the **item** (note) based on the present hierarchy and outline sort order (i.e. it changes if the source Tinderbox file layout is altered); a number such as 1.3.2 is the second grandchild of the third child of the first top leve note. It is the $SiblingOrder number of each ancestor joined by periods.

If **item** is omitted, e.g. used as ^sectionNumber^, the number exported is that of the current note and is equivalent to ^sectionNumber(this)^

If trying to do calculations based on this data, bear in mind that adornments, although not shown outside Map view, number as the last sibling in the level on whose Map they appear.

The `^sectionNumber^` of this page's source note is: "`2.6.2.6.1.37`".

Or use `^value(if($OutlineDepth >1){collect(ancestor,$SiblingOrder).reverse.format(".")+"."+$SiblingOrder;}else{$SiblingOrder};)^` giving: 2.6.2.6.1.37

^setRoot([newRoot])^

Export Code Type: Export Mark-up [other codes of this type]
Export Code Scope of Action: document [codes with similar scope]
Export Code First Added: Baseline
Export Code Last Altered: As at baseline

^setRoot([newRoot])^

The ^root^ code (see) normally exports the relative path from a note to the top level of the Tinderbox document. This is valuable for defining relative paths, especially in macros. But in some cases you may want a relative path in one page (an HTML page, say) and an absolute link in another (an RSS feed, where relative links may be forbidden).

Thus, the export element

^setRoot(newRoot)^

tells Tinderbox to use its first argument (**newRoot**) as the value of ^root^ until either

- another ^setRoot^ is found.
- the current file being exported is complete.

To restore relative URLs, use ^setRoot()^ with no arguments.

For example:

```
^setRoot(https://example.com/)^
```

Using ^setRoot()^ to set the logical host or root path for a page affects not only ^root^ but also ^url^, ^childLinks^, and ^basicLinks^.

^siblings([start, list-item-prefix, list-item-suffix, end])^

Export Code Type: Creation of Links [other codes of this type]
Export Code Scope of Action: item [codes with similar scope]
Export Code First Added: Baseline
Export Code Last Altered: As at baseline

---

^siblings(["start", "list-item-prefix", "list-item-suffix", "end"])^

returns links to all the siblings of this note, i.e. the other children of the note's parent. Output is formatted in a single line, with colons between the elements.

If the optional **start**, **list-item-prefix**, **list-item-suffix**, **end** arguments are omitted then the default values of <ul>, <li>, , are used. See more on altering list type or other HTML output.

This allows you to format the group of links as an HTML list or table in the exact format you want, where " **start**" is text that will be inserted before all of the links, " **end**" is text that will be inserted after all of the links, " **list-item-prefix**" is text will be inserted before each link, and "**list-item-suffix**" is text that will be inserted after each link.

The $HTMLDontExport settings of siblings are respected.

Mark-up elements embedded in ^siblings^ are evaluated.

Exported links use the CSS class and target based on their underlying Tinderbox link values, if one is specified; the link type value is **not** picked up as a class name.

Item objects source and destination may be used in expressions to indicate from where data used within the expression's arguments is drawn.

The export is relative to *current* rather than *this*, facilitating work with included files.

^similarTo(item, count[, start, list-item-prefix, list-item-suffix, end])^

Export Code Type: Creation of Links [other codes of this type]
Export Code Scope of Action: item [codes with similar scope]
Export Code First Added: Baseline
Export Code Last Altered: As at baseline

---

^similarTo("item", count[, "start", "list-item-prefix", "list-item-suffix", "end"])^

The HTML Export element ^similarTo^ locates notes that appear similar to a given note.

The query will find up to **count** notes, but may find fewer, and exports a list of notes it finds. The **count** argument is not evaluated so must be stated as a value and not an expression.

Basic form, returning links to the 10 most similar notes:

```
^similarTo("this",10)^
```

The longer form is only needed if it is desired to use non-default list HTML mark-up or include extra per-item captioning. If one of the option input is specified, all the others must also be, even if only as the default value(s).

If the optional **start**, **list-item-prefix**, **list-item-suffix**, **end** arguments are omitted then the default values of <ul>, <li>, , are used. See more on altering list type or other HTML output.

The code must be used in $Text or directly in a template. If used in a code include to a template, the links generated will have incorrect relative paths.

Similarity is based on several factors, including:

- the text of the note
- the note title
- any text contained in user attributes

In addition, weighting is applied for:

- notes having the same prototype
- notes having roughly similar amounts of text

The results are based on the same logic as the similar tab of the Get Info dialog. The same matching is also available as an action similarTo().

New item objects source and destination may be used with list/item inputs to indicate from where data used within such inputs are drawn.

The export is relative to *current* rather than *this*, facilitating work with included files.

^text([item, N, plain])^

Export Code Type: Data Include [other codes of this type]
Export Code Scope of Action: item [codes with similar scope]
Export Code First Added: Baseline
Export Code Last Altered: As at baseline

---

^text([item][, N] [, plain])^

The body (text and graphics) of the **item** or note, i.e. includes any existing mark-up. If a note name is not specified, 'this' is assumed as the focus. The parentheses my be omitted if no arguments are supplied.

`^text^` exports the current note

`^text(this)^` exports the current note

`^text(/Tasks/Task1)^` exports the $Text of note at path '/Tasks/Task1'

`^text("Task2")^` exports the $Text of note 'Task2'

Item can also be a string attribute holding a path or title of a note. Thus if $MyString has the value "Task2"

`^text($MyString")^` exports the $Text of note 'Task2'

If the note only contains an image, the image's tag () is exported.

Alternative usages…

N words of $Text

The first **N** words of the text of the item or note, where **N** is a number (N.B. **item** must be specified too). Here the first 5 words of the current note's HTML-rendered $Text are inserted:

```
^text(this, 5)^
```

Alternatively, the first **N** words of the text of the item or note, without added HTML mark-up and where **N** is a number (again note that **item** must be supplied). Here the first 10 words of the current note's *un-rendered* $Text are inserted:

```
^text(this, 10, plain )^
```

Plain text export

If no HTML text mark-up is required. This inserts the $Text of the current note without any added HTML mark-up codes

```
^text(this, plain )^
```

N.B. The 'plain' option is still subject to the effects of $HTMLQuoteHTML and $HTMLMarkupText. For instance, if the latter is `true`, then the phrase "This & that." will get exported as "This & that." which might not be as intended, e.g. if exporting via runCommand().

Alternatively, to get the text of the note, totally untouched by other processes (such as $HTMLQuoteHTML), use ^value($Text)^ which is effectively the same as ^text(this, plain)^.

^title([item])^

Export Code Type: Data Include [other codes of this type]
Export Code Scope of Action: item [codes with similar scope]
Export Code First Added: Baseline
Export Code Last Altered: As at baseline

---

^title([item])^

The title of the **item** or the current note if not specified note. The value is the same as the Name attribute, but using HTML entities where appropriate (see below for plain text export). The parentheses my be omitted if no arguments are supplied.

The `^title^` of this page is: "^title([item])^".

The ^title^ of note "Conditional Mark-up" (using code `^title("Conditional Mark-up")^`) is: "Conditional Mark-up".

Export and HTML entities

To avoid the use of HTML entities when exporting a note's title use the ^value()^ code:

```
^value($Name)^
```

which for this note renders as: "^title([item])^".

^url(item)^

Export Code Type: Creation of Links [other codes of this type]
Export Code Scope of Action: item [codes with similar scope]
Export Code First Added: Baseline
Export Code Last Altered: As at baseline

^url(item)^

The relative URL of the exported HTML web page for the **item**. The URL of an alias to a note is the URL attribute of that note's original. It is effectively ^host^ + ^path^ + ^file(note)^.

Do not confuse this with ^value(URL)^ which will return the content of the note's URL field.

^url(this)^ will return the relative path from the *current* page to the page the included note creates when it is exported. If the note is not exported, but only appears on included pages, ^url(this)^ is undefined.

As ^url()^ only supplies the 'href' argument value for an HTML link, if you want to customise your link, e.g. by setting a CSS class argument, you can do that in the template rather than needing to do so in the TB code itself.

^url(this)^ gives a relative path from the current page to the note; by comparison, ^path(this) gives the path from the top level of the site to the note. Thus:

`^url(BorderBevel)^` gives: "`index/Automating_Tinderbox/Coding/Use_of_Attributes/Attribute_Listings/System_Attribute_List/BorderBevel.html`".
`^path(BorderBevel)^` gives: "`index/Automating_Tinderbox/Coding/Use_of_Attributes/Attribute_Listings/System_Attribute_List/BorderBevel.html`".
`^url(this)^` gives: "`index/Automating_Tinderbox/Coding/Export_Codes/Export_Codes_-_Full_Listing/url_item.html`".
`^path(this)^` gives: "`index/Automating_Tinderbox/Coding/Export_Codes/Export_Codes_-_Full_Listing/url_item.html`".

To make a deliberate 'manual' HTML link to another note see ^linkTo()^.

^value(expression)^

Export Code Type: Data Include [other codes of this type]
Export Code Scope of Action: n/a [codes with similar scope]
Export Code First Added: Baseline
Export Code Last Altered: As at baseline

^value(expression)^

The argument to ^value^ is an **expression**, a value that could be assigned to an attribute. ^value^ evaluates the expression and exports the result, as a string (some data types may also require the format() operator too). This allows manipulations for *export* use without needing to, for instance, hold the value in a user attribute first.

^value^ is a more flexible replacement version of the deprecated codes ^get^ and ^getFor^ but which still allows the called attribute's value to be manipulated. For example:

`^value($Width)^`

exports the width of the current note. To export that attribute value for a different note (than in current scope) use action code style offset addressing. So, to call the value of $Width for "Some Other Note" rather than from the currently-processed note, use:

`^value($Width("Some Other Note"))^`

Examples

Number-type data:

`^value(sqrt($Width))^` gives: "`1.732050807568877`".
`^value(sqrt($Width).format(2))^` gives: "`1.73`".

String-type data:

`^value("$Name: "+$Name(parent))^` gives: "`$Name: Export Codes - Full Listing`".
`^value($Name(parent)+":"+$Name)^` gives: "`Export Codes - Full Listing:^value(expression)^`".

Note that ^value($Text)^ equates to ^text(plain)^.

Color-type data:

`^value($AccentColor)^` gives: "`dark warm gray dark`".
`^value(format($AccentColor))^` gives: "`#403a35`".
`^value($AccentColor.format())^` gives: "`#403a35`".

Date-type data:

`^value($Created)^` gives: "2007-07-03T20:54:36+01:00".
`^value(format($Created,"l h:mm"))^` gives: "`03/07/2007 20:54`".
`^value($Created.format("l h:mm"))^` gives: "`03/07/2007 20:54`".
`^value(format($Created,"l"))^` gives: "`03/07/2007`".
`^value($Created.format("l"))^` gives: "`03/07/2007`".
`^value(format($Created,"*"))^` gives: "`Tue, 3 Jul 2007 20:54:36 +0100`".
`^value($Created.format("*"))^` gives: "`Tue, 3 Jul 2007 20:54:36 +0100`".

N.B. Unformatted Date-type data returns the host OS' local "short date" format plus time (in hh:mm form). For the aTbRef author's UK-locale OS that equates to a format string of "`l h:mm`" but it may vary for other OS locales. However, a date format string *must* be given if format() or .format() are used:

`^value(format($Created))^` gives: "" (no date format string!).
`^value(format($Created))^` gives: "`03/07/2007, 20:54`" (no date format string!).

See also ^action (action)^ usage which allows action code to be run during export.

Another useful aspect of ^value^ is that it give access during export to 'raw' attribute data without any process. That is why there is the reference above to some data types needing use of format(). The upside of this raw state is attributes like HTMLMarkupText and HTMLQuoteHTML have no effect; unlike older versions non-ASCII are not encoded. If using the HTML Export process to generate/export formats other than HTML or if accessing the command line it is often necessary to use different forms of escaping. These different outputs can be achieved by using the action code formatting operators within the ^value()^ call. For exporting $Text, *in HTML format*, ^text^ will give a better result but for other export formats ^value($Text)^ may suffice.

Passing variables into export code

^value()^ can be used, *within the context a single template* , to insert a var() variable declared within an ^action()^ code. Note that the variable must be declared before use, i.e. before as in reading template code top to bottom. **Replacing deprecated export codes** ^value()^ replaces both the deprecated ^get()^ and ^getFor()^ codes. ^get()^ maps to ^value($SomeAttribute)^ and ^getFor()^ maps to offset use ^value($SomeAttribute("Some Other Note"))^. If existing documents use the older codes they will work for the present but it is highly recommended updating them to use ^value()^ instead.

^version^

Export Code Type: Data Property [other codes of this type]
Export Code Scope of Action: document [codes with similar scope]
Export Code First Added: Baseline
Export Code Last Altered: As at baseline

^version^

The version number of Tinderbox used to create this web page. The number is exported as a string without a prefix, i.e. "4.6.0" *not* "v4.6.0".

Note that this data cannot be retrieved using action code.

The `^version^` for this file is: "`9.6.2`".

Export Code Types

Export codes can perform various sorts of roles:

- Boolean Comparison
- Calculation
- Conditional Mark-up
- Creation of Links
- Data Include
- Data Property
- Export Mark-up

Boolean Comparison

These codes all result in a Boolean `true/false`.

- ^nextSibling^
- ^previousSibling^

Calculation

These codes calculate values which are then added to the output data.

- ^action(action)^

Conditional Mark-up

These codes create the framework for assessing Boolean conditions. The basic structure is:

if ([not] (condition)) result1 else if ([not] (condition)) result2 else result3 endif.

The optional 'not' argument reverses the conditional check to matching `false` instead of `true`.

- ^else^
- ^endIf^
- ^if(condition)^

Creation of Links

These codes all result, with the exception of ^linkTo^ and ^url^, in the creation of a set of links in HTML output; ^linkTo^ creates a link to a specific item object or note. Note the arguments are optional. ^ancestors^ works without any other arguments, outputting just the links without any further optional formatting.

Ampersands in link anchor text are exported in HTML entity for to help ensure validating HTML code output.

- ^ancestors([start, list-item-prefix, list-item-suffix, end])^
- ^basicLinks([start, list-item-prefix, list-item-suffix, end])^
- ^childLinks([start, list-item-prefix, list-item-suffix, end])^
- ^inboundBasicLinks([start, list-item-prefix, list-item-suffix, end, type])^
- ^inboundLinks([start, list-item-prefix, list-item-suffix, end])^
- ^inboundTextLinks([start, list-item-prefix, list-item-suffix, end, type])^
- ^linkTo(item [, data] [,css class])^
- ^outboundBasicLinks([start, list-item-prefix, list-item-suffix, end, type])^
- ^outboundLinks([start, list-item-prefix, list-item-suffix, end])^
- ^outboundTextLinks([start, list-item-prefix, list-item-suffix, end, type])^
- ^outboundWebLinks([start, list-item-prefix, list-item-suffix, end, type])^
- ^siblings([start, list-item-prefix, list-item-suffix, end])^
- ^similarTo(item, count[, start, list-item-prefix, list-item-suffix, end])^
- ^url(item)^

Data Include

These codes add data to the exported content.

When including children that are aliases using the codes listed here, children that happen to be aliases are included as aliases. Previously, the original was included.

Since for most purposes the alias and its original are interchangeable, this change will seldom affect export. The nuance here is that intrinsic properties of the alias ($Xpos, $Ypos, $Height, $Width, $SiblingOrder, and so forth) are exportable.

- ^children([template][,N])^
- ^comment(data)^
- ^descendants([template][,N])^
- ^directory(item)^
- ^do(theMacro [,argument, anotherArgument, etc.])^
- ^include(item|group[, template])^
- ^paragraphs([item,] N)^
- ^randomChildOf(item)^
- ^randomLine(item)^
- ^text([item, N, plain])^
- ^title([item])^
- ^value(expression)^

Data Property

These codes return various properties, some from attributes, preferences or just with information about the document or the program itself (e.g. the version number). These are similar to the Data Includes in that they add data to the export but are more focused in their scope.

- ^docTitle^
- ^file(item)^
- ^host^
- ^sectionNumber(item)^
- ^version^

Export Mark-up

These codes are used to help structure the exported data.

- ^backslashEncode(data)^
- ^cloud(item, count])^
- ^documentCloud([count])^
- ^indent([data][, N])^
- ^path[(item)]^
- ^root^
- ^sectionCloud([item, count])^
- ^setRoot([newRoot])^

Export Code Scope

A fair percentage of Export Codes reference either a group (e.g. the children group designator) or an item (either a note by name or via an item designator). In some cases the object or group may be derived from an expression but do not assume arguments are evaluated unless this is documented so for the given code. The various scopes are listed below:

- Document-based
- Group-based
- Honour link type filter
- Item-based
- Scope not applicable

Document-based

Export codes that operate at document scope:

- ^docTitle^
- ^documentCloud([count])^
- ^host^
- ^path[(item)]^
- ^root^
- ^setRoot([newRoot])^
- ^version^

Group-based

A family of HTML Export mark-up elements operates on groups of notes. For example:

`^every(child, attribute)^`

returns *true* if every child of the note has an attribute value of `true`, and `false` otherwise.

If the group is empty, the group operator returns the value of the attribute for the current note.

For example, ^any(child,$Urgent)^ returns the note's current value of $Urgent for notes that have no children. This makes it easier to construct rules, reducing the need for |= and complex ^if^ constructions.

The possible referenced groups include:

- all
- children
- descendants
- ancestor
- sibling

The codes are:

- ^include(item|group[, template])^
- ^sectionCloud([item, count])^

Honour link type filter

These codes use a scope that may be modified based on a link type based filter:

- ^outboundBasicLinks([start, list-item-prefix, list-item-suffix, end, type])^
- ^outboundTextLinks([start, list-item-prefix, list-item-suffix, end, type])^
- ^outboundWebLinks([start, list-item-prefix, list-item-suffix, end, type])^
- ^inboundBasicLinks([start, list-item-prefix, list-item-suffix, end, type])^
- ^inboundTextLinks([start, list-item-prefix, list-item-suffix, end, type])^

Item-based

These code work in the context of a single item:

- ^ancestors([start, list-item-prefix, list-item-suffix, end])^
- ^basicLinks([start, list-item-prefix, list-item-suffix, end])^
- ^childLinks([start, list-item-prefix, list-item-suffix, end])^
- ^children([template][,N])^
- ^cloud(item, count])^
- ^comment(data)^
- ^descendants([template][,N])^
- ^directory(item)^
- ^do(theMacro [,argument, anotherArgument, etc.])^
- ^file(item)^
- ^inboundBasicLinks([start, list-item-prefix, list-item-suffix, end, type])^
- ^inboundLinks([start, list-item-prefix, list-item-suffix, end])^
- ^inboundTextLinks([start, list-item-prefix, list-item-suffix, end, type])^
- ^indent([data][, N])^
- ^linkTo(item [, data] [,css class])^
- ^nextSibling^
- ^outboundBasicLinks([start, list-item-prefix, list-item-suffix, end, type])^
- ^outboundLinks([start, list-item-prefix, list-item-suffix, end])^
- ^outboundTextLinks([start, list-item-prefix, list-item-suffix, end, type])^
- ^outboundWebLinks([start, list-item-prefix, list-item-suffix, end, type])^
- ^paragraphs([item,] N)^
- ^previousSibling^
- ^randomChildOf(item)^
- ^randomLine(item)^
- ^sectionNumber(item)^
- ^siblings([start, list-item-prefix, list-item-suffix, end])^
- ^similarTo(item, count[, start, list-item-prefix, list-item-suffix, end])^
- ^text([item, N, plain])^
- ^title([item])^
- ^url(item)^

Scope not applicable

These codes have no particular scope:

- ^action(action)^
- ^backslashEncode(data)^
- ^else^
- ^endIf^
- ^if(condition)^
- ^value(expression)^

Using Export Code

Tinderbox has 45 built-in 'export codes' associated with HTML and text export, including export macros. Although they may be used in actions, queries and rules such use outside an export context is strongly deprecated and likely to less we supported, or not supported at all, in forthcoming versions. Action code has been developed for 'internal' use, i.e. in queries/action/rules as opposed for actual export from Tinderbox and thus internal use of export codes is *strongly deprecate* (though often still legacy-supported) in favour of the new syntax.

Export codes can be used anywhere in the template HTML (or other format) file, though bear in mind the above developments and guidance. Export codes are marked with a caret (^) symbol, entered by typing Shift+6. All export codes start with a caret and—optionally—also end with one. In many cases the closing caret can be omitted, e.g. for brevity in complex ^if()^ statements. However, to ensure Tinderbox emits 'clean' HTML (with proper tag nesting) always include the the closing caret if a code comes at the end of a paragraph or note and the is no whitespace after it. The advice here to new users is to always include the closing caret, regardless.

A typical scenario is putting ^childLinks at the end of a note; the HTML output will render but it is better to put ^childLinks^ to ensure correct nesting of HTML tags. Without the closing ^ mark-up Tinderbox has to make a guess and will wrap the link list in

tags creating code that will not validate (if meeting such a criterion is required). Although Tinderbox generally guesses export code closures correctly, it is better practice for the user expressly to close their codes to avoid any ambiguity during processing.

Arguments. Export codes usually have one or more arguments; there is a separate listing explaining the meaning of code arguments such as 'item', 'data', etc.

Evaluation of arguments. Individual code arguments may be evaluated, i.e. allowing substitution of export code or regular expressions, date placeholders, etc., for explicit argument values. However, at present there is no way to tell if a given code's arguments are evaluated (unless it says so here in aTbRef) other than by trial and error (or asking Eastgate directly). One, all or no arguments may be evaluated for any given code. It is safest to assume an argument is not evaluated unless there is clear evidence in the notes to the contrary. Unlike in action code where attributes references are marked by using a $ prefix with the attribute name, this is not done in export code, though such syntax may work if used.

Tinderbox uses ^ as a delimiter for export elements such as

`^title^`

`^value($Name(parent))^`

It is not a requirement to provide a closing caret at the end of the code, though not doing so leaves Tinderbox to have to guess the users intent. The guess is normally right, but for the sake of an extra character you can avoid the guesswork. New users, especially, are advised to always close their export codes with a caret, thus:

`^title` ← works but deprecated for new users

`^title^` ← safer usage

Sometimes, however, a caret is simply a caret. Two common cases appear:

- errors, where we specify an export element that does not exist because we have mistyped the name: `^tite^`, `^value($Name(parant))^`
- uses of carets that are not intended for Tinderbox, such a regular expressions in JavaScript scripts.

Tinderbox exports a space after an caret in an unrecognised export element, because expressions like

`$MyName=^Name^` ← WRONG

could loop infinitely. (The user doubtless meant to write either $Name or ^title^; there is no export element ^Name^). Thus on saving the note or dialog, Tinderbox would insert a space, thus:

```
$MyName=^ Name^
```

Exporting the extra space works well in some contexts, but not when the caret is part of a regular expression in, for example, JavaScript. The additional space is exported only if the name could be a mark-up element:

```
^Name^ → ^[space]Name^
^Title^ → ^[space]Title^
```

but

```
^[a-z] ^ → ^[a-z]^
```

In post-v4 use there is also a de facto move to expect explicit quoting of all string literals (with double-quotes, ""), a significant exception being path arguments (e.g. path, partial path or note name) and designators which by convention are quoted. Doing this will tend to pre-empt unexpected failures of user code.

There are syntax considerations for colour and date type attributes when read:

- ^value($ColorAttribute/format())^. In HTML Export, named colours are exported as hex code strings which include the # prefix character, e.g. "#99ff00".
- ^value($DateAttribute.format("formatString"))^. The format string is optional. Tinderbox offers numerous date formats.

Altering mark-up for Export Code generated lists

There are a number of export codes that generate lists of links. By default, they generate an unordered (bulleted) HTML list, i.e.

```
<ul>
   <li> … </li>
</ul>
```

Those 4 HTML tags can be modified by supplying 4 optional arguments:

```
^basicLinks( "start", "list-item-prefix", "list-item-suffix", "end" )^
```

Thus using:

```
^basicLinks( )^
```

is functionally the equivalent of a more explicit:

```
^basicLinks( "<ul>", "<li>", "</li>", "</ul>" )^
```

Using any one of these optional arguments, i.e. changing the HTML tag for one, requires all four inputs to be explicitly stated. Thus:

Incorrect: `^basicLinks( , "<il class="xlist">", , )^`

Correct: `^basicLinks( "<ul>", "<il>", "</li>", "</ul>" )^`

Setting a numbered list style

To change a bulleted list to a numbered list, give all four inputs, but change the first and last inputs (re-specifying the defaults for the other tow), thus:

```
^basicLinks( "<ol>", "<il>", "</li>", "</ol>" )^
```

Export Codes with link type filtering

A subset of list-generating codes that allow filtering by link type use a fifth option input giving the link type(s) to be included in the list:

```
^inboundBasicLinks("start", "list-item-prefix", "list-item-suffix", "end", "type" )^
```

For these codes, if the 'type' argument is supplied, all other arguments must be supplied even if not customised, e.g.

```
^inboundBasicLinks()^
```

or

```
^inboundBasicLinks( "<ul>", "<il>", "</li>", "</ul>", "(example|untitled)" )^
```

Debugging user Export code

In additions to the techniques suggested for Action code debugging, consider:

* Using ^action()^ to record template-only calculations back into user attributes for later inspection. This is useful with complex exports. Once the export process is validated, instead of using attribute values, action code variables can be us instead. If using variables to store sections of pre-evaluated code, it is a good idea to purge the attributes after use. By comparison variables are non-persistent and so purge themselves. * Using the text pane's HTML and Preview tabs to check export code before actual export. * Full HTML export looks at whether source attributes have changed, thus changes in templates code are not 'seen' and the process can become obfuscated in complex include exports. One way around this is simply to delete (or move) previously exported files as full export will export a note if it is not found at the export location. * The File menu ▸ Export Selected Note option can 'force' export a single note, though note the offered filename may not be that used in full export if the note's automatically derived filename is not unique within the scope of its present container. If exporting over a previous full export, select the old filename in the export location dialog so the name is re-used and the new file overwrites the old one. * The effects of $HTMLDontExport and $HTMLExportChildren on the use of ^include^. The latter cannot return data from a note that is configured to not export. * For $Text export usir ^text^ remember the effects of $HTMLMarkupText and $HTMLQuoteHTML. * The slightly different effects of ^title^ vs. ^value($Name)^ and ^text^ vs. ^value($Text)^.

Export Code Arguments

The argument **action** refers to a statement that can be evaluated as if an Action or Rule.

The arguments **argument** and **anotherArgument** refer to text or code supplied as macro arguments.

The argument **attribute** refers to an explicit attribute name.

The argument **count** refers to a number, in figures, e.g. 5 not five.

The argument **css class** is the (text) name of a CSS mark up class.

The argument **data** (**data1**, **data2**, etc.) refer to either:

- an explicitly entered value.
- an attribute's value.
- a regex (i.e. regular expression)

The argument **date** refers to either:

- an explicit date "2 Feb 2007".
- a date object, e.g. today, yesterday - 5.
- a date attribute, ^value($MyDate("some other note").format("formatString"))^.

The argument **expression** refers to a statement that can be evaluated and assigned to an attribute.

The argument **format** refers to either:

- a date/time formatting string.
- a colour string in #000000 format.

The argument **group** is either:

- a relative reference to a group of notes or agents, like in ^any([group], AttributeName)^.
- another export code that gets evaluated in to a group, like in ^value(expression)^.
- a predefined group designator (e.g. descendants, etc.,).

The argument **item** is either:

- a note or agent's name, like in ^value($Name)^.
- a relative reference to a note or agent, like in ^value(AttributeName([item])^.
- another export code that gets evaluated into a note or agent's name, like in ^value(expression)^.
- a predefined item designator (e.g. child, etc.,).

The argument **N** refers to a number, in figures, e.g. 5 not five.

The argument **precision** refers to the number of decimal points that are exported.

The argument **path** refers to the path to a note or agent (including the latter's Name).

The arguments **start**, **list-item-prefix**, **list-item-suffix**, **end** refer to optional alternate HTML mark up and text for a list of links.

The argument **target** refers to either:

- a (text) string value to be matched against. Number and other value types are effectively treated as strings for comparative purposes.
- an attribute value.

The argument **template** refers to a template file name either:

- stated as a string.
- stored in a named attribute.
- inherited as a default from Preferences.

Export argument values *may* be enclosed in double quotes, especially if doing so can help avoid parsing difficulties; the enclosing quotes will be ignored by the parser. Use of quotes for string values is recommended. This is because many new operators have been added and the scope for misinterpretation of string literals as code has increased. For example, if a note is named…

```
she/he
```

…the comma would confused the parser in expressions like:

```
^linkTo(she/he)^ <- will not work
```

…so this may (optionally) be written with double quotes:

```
^linkTo("she/he")^
```

Characters warranting explicit quoting are those that may be mistaken for simple maths or equality operators: so, + - * / = ! | & > < |= &= != >= <= etc!

Link lists can use two new references in their optional mark-up arguments:

- source: refers to the note that is the source of the link.
- destination: refers to the note that is the destination of the link.

Export codes honouring link types

The listing below shows those export codes which are aware of link types.

- ^inboundBasicLinks([start, list-item-prefix, list-item-suffix, end, type])^
- ^inboundTextLinks([start, list-item-prefix, list-item-suffix, end, type])^
- ^outboundBasicLinks([start, list-item-prefix, list-item-suffix, end, type])^
- ^outboundTextLinks([start, list-item-prefix, list-item-suffix, end, type])^
- ^outboundWebLinks([start, list-item-prefix, list-item-suffix, end, type])^

Tinderbox URL schema

Tinderbox has its own private pseudo-protocol URL scheme. Entering a Tinderbox URL in a browser or similar program will launch Tinderbox (if installed!) and request specific actions.

The schema is as follows:

```
tinderbox://document/path?action
```

The protocol handler for `tinderbox://` URLs will automatically open and bring to focus documents in the Recent Files list as well as documents that are already open.

A Tinderbox URL pointing at (selecting) the current note in the current view type can be copied to the clipboard using the Note menu or the view pane selection's context menu. This URL is also exposed for any note via the read-only $NoteURL attribute. The desired selection is cited by that note's $ID. If there is a multiple selection the discrete $IDs are a semi-colon delimited list.

document is the name of an open document, without the '.tbx' file extension.

path is a conventional Tinderbox path within the **document** to a specific note (or container).

actions, which may be multiple, joined with an ampersand.

Schema Actions

pane, which opens an inspector to the designated pane (there is no documentation of valid pane values):

```
?inspect=pane
```

view, selects a tab with the designated parent and view type, or creates a new tab in no existing tab can be found. Allowed values for *viewType* include {*map*, *outline*, *chart*, *attributeBrowser*}.

```
?view=viewType
```

select, which selects the desired note(s) when the target document takes focus:

```
?select=ID
```

Thus:

```
tinderbox://Notes/Issues?select=1429560859
tinderbox://Notes/Issues?select=1429560859;1429458942
tinderbox://Notes/Issues?select=1429560859&view=map
```

Coding conventions

How many discrete code syntaxes does Tinderbox use? Two: action and export. The former is used internally and is Tinderbox's scripting 'language'. Export code is used only for 'HTML' export purposes, enabling the insertion of mark-up and links int the exported text. In both types of code arguments are supplied as a comma-delimited list within parentheses (i.e. normal brackets), being comma-delimited if there is more than one.

Whitespace in codes. For codes using parentheses to hold inputs, there must be no whitespace between the last character of the code and the opening parenthesis. Within the parentheses, additional whitespace—e.g. between inputs or in expressions—is ignored.

Action code syntax. Used within Tinderbox documents for use with rules, agents, $OnAdd and the like. Action code operators are case sensitive. More on Action code syntax.

Export code syntax. The original Tinderbox automation syntax was 'export code', used for exporting code to web(log) pages; such code is always started and ended with a caret symbol (^, typed via shift+6). Expert users may optionally omit the closing carets though this leaves Tinderbox to infer where a code ends. Originally, Export code could also be used internally in actions and queries but this is now *deprecated*. As most export task-specific codes can be replicated using an action code enclosed in the ^value()^ export tag and so many of the export codes are now *deprecated* in current use.

Query codes. *All now deprecated as there are direct action code replacements. You might still see query codes in agent queries in old files. Such codes start with a hash (#) and all have direct action code replacements.*

Case-sensitivity

Attributes and action codes are always case-sensitive, both for system and user attributes. The convention is to use 'CamelCase', i.e. initial capital and capitals for concatenated words; thus the convention would suggest `MyString` as a us attribute name but alternates such as `mystring`, `Mystring` would also work but—importantly—as different attributes from `MyString`.

Unlike attribute names, export codes are case insensitive but convention when using them is to capitalise internal word joins but not the initial letter; thus 'correct' usage takes forms like ^endIf^, urlEncode but you will find URLEncode, UrlEncode, UrLeNcOdE, etc., will all work just as well.

Documentation will tend to follow the conventions, but feel free to use different casing if it works for you. More on case sensitivity.

Attribute name limitations

This is discussed under Attribute Naming.

Use of parentheses in export code

In export code where no arguments are required, parentheses can usually safely be omitted. With codes whose only argument is 'item', the parentheses can usually be omitted is the desired scope is 'this'. Thus ^title^ is equivalent to ^title(this)^.

Delimiting arguments

A comma is used as the delimiter between arguments. Some arguments may be optional. Any syntax examples in the manual or Release Notes where argument(s) are listed in square brackets (e.g. [data]) are optional.

A notable exception to the comma as a delimiter is the links() operator where a dot (period character) notation is used.

Where arguments allow non-quoted text strings, if the string includes a comma the whole string should be quoted.

Legacy constraints:

- **General code conventions from v5.0.0:**
  - all action references to attributes, both left and right side of operations now take a '$' prefix. A significant exception is that the AttributeName(regex) operator *does not take a $ prefix*.
  - all action code literal strings are now double-quote enclosed.
  - export codes use a mix of quoted/non-quoted on a code-by-code basis (i.e.see notes on the individual codes).
  - where possible export code is now deprecated in favour of action code, i.e. everywhere *except* in an export context. Support for internal use of export code is likely to be dropped in the future so existing code should be updated to reflect this.
- **Changes as from v5.7.0**

A new subs-set of dot-operators were introduced, in the form *(data-typed-object or literal value).operator*, e.g. `$MyString.contains("ant")`, `"Advice for Defendants".contains("ant")`. In this type of operator, the first part is a code expression, an attribute reference or a literal (number, quoted string, etc.). After the dot, the operator name follows post v4.6 conventions; some operators require no inputs and trailing empty parentheses are not required. More advance users can thus chain operators as long as they know the data type of the object to which they are dot-attaching additional operators.

More on coding conventions:

- Case-sensitivity
- Lexical vs. numeric sorting
- Quoting and escaping strings in Tinderbox coding
- Code: Straight vs. Typographic 'curly' quote characters
- Attribute references: quoting of literal strings vs. paths
- Quoting Regular Expressions
- Using $ and quotes in Export code
- Single and double quotes
- aTbRef's naming conventions
- Code examples using attributes with a 'My' prefix
- Square Brackets in code operator explanations

Case-sensitivity

Tinderbox's host OS is case-sensitive as is the app in the majority of cases. Notable exceptions are export codes and agent query special arguments. In both cases there is a general TB usage convention to intercapitalise such words with a lower case start and capitals at concatenated word boundaries. Inter-capitalisation helps distinguish different Tinderbox code types.

Conversely, attributes are case-sensitive: IsInside, Isinside, isInside and isinside would all be treated as different attributes by Tinderbox if forced to guess. Of the four styling the first is recommended, the second less so and the last two are deprecated to avoid user or app confusion as to purpose when using in code.

To help illustrate this, consider a fictional 'insideness' state—'is inside'—as expressed through the major code types, noting that:

- ^isInside(arg)^. An export code: inter-capitalised plus the carets indicate this is export code. (This is also a real export code). *Export codes are case-insensitive*.
- IsInside. An attribute: opening capital plus inter-capitalisation. The styling of the words would also tend to imply in Tinderbox usage that this would be a Boolean type, `true/false` attribute. (This is not a real system attribute). *Attributes are case-sensitive*.
- isinside(arg). An action operator, all lowercase. This would likely be the internal analogue of ^isInside^, though the case insensitivity of the later codes means export and action codes tend not to share a name. (This is not a real action operator). *Action codes are case-sensitive*.
- Deprecated/defunct:
  - #isInside(arg). A special query argument. The opening hash characters is a give-away plus inter-capitalised styling. Note: this is not a real argument, though #inside(arg) does this function. (*#codes are deprecated since v4.6 for action codes*). Old #codes were c*ase-insensitive*; new query codes, being action code, are *case-sensitive*.

As Attribute and Action codes are case-sensitive it is a good working model to pretend Export codes are also case-sensitive, and thus stick to a consistent case for the latterTbRef follows this model using lowercase with uppercase letter for word boundaries, as in '^ descendedfrom^'.

Other areas where case sensitivity has an impact are in searching (queries & find dialog) and sorting (containers). By default:

- new (agent) queries are case-insensitive but that state can be toggled by a tick-box on the agent's Action Inspector's Query sub-tab and via the AgentCaseSensitive attribute.
- the Find dialog opens with case-sensitivity set 'on' but this can be toggled via a tick-box on the dialog.
- containers sort case-sensitively by default, i.e. letters sort by character number with A-Z sorting before a-z. Container sort has controls both via note/agent Action Inspector's Sort sub-tab and via the Sorting group of system attributes.

Sorting is discussed in more detail separately.

Lexical vs. numeric sorting

Tinderbox containers can sort their contents, as can agents. In addition action code offers methods for sorting lists. Sorting generally occurs in one of two forms, lexical or numerical.

Sorting can be set via action code or more normally via the Sort tab of the Action Inspector.

Lexical sort. A lexical sorts characters in broadly alphanumeric order, for unaccented Roman alphabet languages like English. In fact such sorts look at the underlying ASCII/Unicode character number and sort from lowest to highest for each character, in turn of a word or string of characters. Thus numbers always sort before uppercase letters and upper case before lower case letters. Accented characters come after that. This unusual order reflects the numerical sequence of codes used indicate different letters symbols and numbers. This order has several odd effects:

- numbers sort out of arithmetical sequence: `1, 10, 11, 120, 13, 2` instead of `1,2,10,11,13,120`.
- instances of the same word in different letter case do not sort together: `Ant, Bee, ant` not `Ant, ant, Bee`.
- words with accents may sot out of sequence.
- sequential numbers in word strings do not sort sequentially: `Chapter 1, Chapter 11, Chapter 2` not `Chapter 1, Chapter 2, Chapter 11`.

Numerical sort. Only used for number sequences. Here the numerical values of the whole number string is computed and these values sorted in ascending numerical sequence order. Thus the order `1,2,10,11,13,120` not the lexical ord of `1, 10, 11, 120, 13, 2`.

Dates. Date sort, in date order naturally. The exact form is neither strictly lexical or numerical but Tinderbox takes care of date sort correctly.

Transforms

To work around some of the limitations of basic lexical sorts, as seen from a human perspective, Tinderbox also offers some 'transforms' which tweak the way sorting occurs:

- *case-sensitive*. Sorts upper and lower case sort in separate sequences, not alphabetically (a 'computer' sort). This is the default.
- *case-insensitive*. Converts elements to lower case before comparing them, resulting in a more normal 'human sort' where similar letter sort together alphabetically regardless of letter case.
- *last word*. Sorts on the last word of the value, using a case-insensitive sort (see above). This is useful when the specified attribute values are personal names, as often occurs in bibliographies and blog rolls. The last 'word', is the substring of characters between the last space in the string and the
- *original note*. Instructs the sort routine to sort any (alias) note based on the properties of the original note, not on the intrinsic property of the alias. When applied to attributes that are not intrinsic, the transformer has no effect. This setting is useful in agents when the source of the alias is itself an alias as opposed to an original notes where sorting on $OutlineOrder would not necessarily give the expected outcome (see the Release Notes for fine detail as to why).

Sorting in accented/non-roman text languages

This may likely not be as expected due to the limitations of lexical sorts which are not, without further manipulation ('collation'), aware of per-language sorting nuances. This area of the application is noted as having scope for improvement and likely more locale-specific collation will become available in due course.

So for other characters, accents, etc., sorts may not meet linguistic expectation as the values will be based on Unicode sort order. Thus:

"dog" > "cat"
"dog" > "Dog"
"dogs" > "dog"
"dogs" > "dogma"
"dogs" < "døg" <-- NOTE!

The prevailing locale's sorting rules for handling diacritics and accents.

Tinderbox will use the OS' localisation settings to determine what rules apply for the sorting of accented and other characters such as a ß. If it is desirable to sort using a different localisation, consider use of locale() to alter the local Tinderb environment.

Sorting and Lists/Sets

The discrete values are sorted such they are listed in lexical sort order.

Quoting and escaping strings in Tinderbox coding

From version 4 onwards Tinderbox has moved towards a convention of using double quotes to enclose string literals (i.e. text) when used in Tinderbox codes. An example for ambiguity is where we wish to clarify that we mean 'yesterday' as the word yesterday as opposed to the Tinderbox date placeholder 'yesterday' that is used to subtract one day in doing date calculations. In such a scenario with want to ensure the first instance is quoted and the later is not, thus:

```
$DisplayExpression = "Yesterday is: " + date(yesterday)
```

In many contexts the app will still guess—usually correctly—if quotes are not used. The reason for the change in convention is the increase in the scope of coding that Tinderbox offers and with it the scope for unintended consequences if the app must figure out what is text and what is not. As a plus sign may be either a potential string concatenator (joiner) or an addition sign and a dash be a minus sign or hyphen, then quoting strings also helps when the app must coerce numbers to strings and vice versa.

Although the advised norm for enclosing string literals is to use double quotes, Tinderbox can handle balanced single or double quotes as text string delimiters. This can be helpful when trying to handle strings that must use double quotes, e.g. the quotes surrounding HTML attributes:

```
$MyString = '<div width="' + $ItemWidth + '">';
```

Code: Straight vs. Typographic 'curly' quote characters

IMPORTANT: Both Action Code and Export Code *do not treat straight and typographic quote characters as equivalents* .

The issue of unseen character auto-substitution for OS or app

Only straight single/double quotes have keys on the Mac keyword, whilst the typographic or 'curly' style of quote—more conventionally using in printed text—require use of keyboard shortcuts unfamiliar to many users (see table below). As a result, the macOS frameworks used by Tinderbox often default to auto-substitution of straight quotes into their typographic equivalents. this is true for the $Text area of the Tinderbox text pane.

This auto-substitution of straight quotes with typographic ones can be controlled via the $SmartQuotes boolean system attribute. See more on Smart Quotes, and further below in the section on use of built-in prototypes.

Quotes and Tinderbox Code

In both Action Code and Export Code, only straight quotes, single or double, are treated as valid string delimiters in code. Thus:

WRONG `$Color = “red”;`
WRONG `$Color = ‘red’;`
CORRECT `$Color = "red";`
CORRECT `$Color = 'red';`

Either form can be used inside a correctly quoted-enclosed string:

CORRECT `$MyString = "He said “It’s ‘just’ an example”, then left";`

But, take care if using straight quotes inside a string. The next example will not work as expected:

WRONG `$MyString = 'He said "It's 'just' an example", then left';`

The outer single straight quotes happily enclose the double straight quotes in the string. But, the string also contains straight single quotes, the first of which closes the string, i.e. 'He said "It'. If using typographic quotes in the string is not possible, defining a string that contains both types of straight quotes requires defining the whole text as several discrete stings such that no single string contains a quote of the type used to define the string. For instance:

CORRECT `$MyString = 'He said "'+"It's 'just' an "+'example", then left';`

Notice how the second string switches to double quotes around the string which contains straight single quotes.

Setting up a TBX file for code work

Because coding in Tinderbox uses the $Text area, auto-substitution of quotes can be problematic. Reflecting this, a number of Tinderbox's built-in prototypes are pre-configured with their $SmartQuotes set to `false` thus suppressing quote substitution. These prototypes include:

- Action.
- Code.
- HTML.
- Poster.

The different quote types defined

Character	Description	Unicode	macOS	HTML
'	straight single quote	U+0027 (APOSTROPHE)	'	' (or ')
"	straight double quote	U+0022 (QUOTATION MARK)	"	" (or ")

Character	Description	Unicode	macOS	HTML
‘	opening single quote	U+2018 (LEFT SINGLE QUOTATION MARK)	option +]	&lsquo;
’	closing single quote	U+2019 (RIGHT SINGLE QUOTATION MARK)	option + shift +]	&rsquo;
“	opening double quote	U+201C (LEFT DOUBLE QUOTATION MARK)	option + [	&ldquo;
”	closing double quote	U+201D (RIGHT DOUBLE QUOTATION MARK)	option + shift + [	&rdquo;

Attribute references: quoting of literal strings vs. paths

Faced with an unquoted string as an attribute argument referring to another note, if the string starts with a forward slash Tinderbox will assume the string is a Tinderbox paths.

Unquoted strings as arguments:

```
$MyString(Some note)
```

will be tested for matches to paths or designator syntax before looking for a more localised match such as a $Name value.

In contrast, a quoted string argument will be assumed to be a $Name value.

```
$MyString("Some note")
```

Quoting Regular Expressions

As action code syntax eclipses older syntaxes there is some scope for confusion where regular expressions (regex) are used. With 'dot' operators, e.g. String.contains(regex), one of the options for the 'regex' input argument is a regular expression which needs to be quoted, i.e. enclosed in double quotes. This conforms to a general rule of thumb that anything that is a string (literal, template name, regex, etc.) should be quoted. This helps Tinderbox to detect where the argument is actually an expression.

The exception to this is the long standing, **but now deprecated**, AttributeName(regex) operator originally introduced as part of the query language; with this do not quote your regular expression. This operator should new be considered deprecated in favour of the $Attribute.contains(regex) operator. Thus:

Old usage: `$MyString(\W+)` (now deprecated)

Current usage: `$MyString.contains("\W+")`

- Regular Expression usage

Regular Expression usage

Note that an input that takes regular expression can take either:

- a literal string, e.g. "Hello"
- an entirely regular expression encoded string, e.g. "\d{2,3}"
- a mix of the two, e.g. "Hello.+".

The operators .contains() and .containsAnyOf() use Apple's regular expression engine.

In action codes (or other operators) that are described as using regular expression patterns for their input arguments (i.e. 'patterns'), it is possible to use the `\uNNNN` method to use the 4-digit Unicode code point number for a character to te for a particular character, especially those that cannot easily be typed. Thus a non-breaking space, which *looks like* a normal space when querying text, is encoded as `\u00A0`, whilst a normal space is `\u0020`:

```
$Text = $Text.replace("\u00A0","\u0020");
```

…replaces every instance of a non-breaking space with a normal space character.

An alternative, if the character is in the ASCII range, is to use the ASCII decimal number in the form `\xNN`. Thus:

```
$Text = $Text.replace("XXX","\x22");
```

…replaces every instance of 'XXX' with a straight double quote character.

Some reference links:

- Syntax for regular expressions using the Apple Regular Expression engine.
- Wikipedia ASCII codes. IMPORTANT: use the values in the 'Hex' column. Thus a straight double quote is `22`, a straight single quote is `27`.

Historical note re pre-v6 Tinderbox

Originally, regular expressions in Tinderbox used Perl language conventions as further defined in documentation for the Boost regex code library: https://www.boost.org/doc/libs/1_34_1/libs/regex/doc/syntax_perl.html.

Using $ and quotes in Export code

Much of older export code is likely to be eclipsed by newer codes like ^value(…action code…)^. Although historically export code did not need, or sometimes did not like, $-prefixes or quoted strings the ^codes have been reviewed in v5.7+ and should behave correctly if you use action code type conventions.

All more reason to review old code and adopt current syntax for all new work.

Single and double quotes

When referring to quoting values such as strings, double quotes should be assumed. In edge cases where a string needs to enclose a double quote, for instance a complex command line string, paired single quotes may also be used.

Within quoted text a backslash (`\`) escapes a character and a double backslash sequence (`\\`) allows a literal backslash (`\`) to be used. Besides escaping non-pairing quotes, standard escape codes like `\n` indicating a new line character or for a tab may be used.

However, a backslash cannot be used to escape literal single or double quote characters. This effectively limits the nesting of different types of quotes to one set inside another.

Whilst the mechanism should work for other characters, it is generally only single & double quotes which cause difficulties when used unmatched (or nested) within string literals (including regular expressions) in Tinderbox. The latter arises a quotes are used as delimiters for inputs and current parsing does not support a quote escape mechanism at initial-parse level. The above method 'hides' the quotes from that initial parse.

aTbRef's naming conventions

Tinderbox users can create action code, export code, user-defined attributes, rules, edicts, stamps, functions, prototypes, templates, etc. This is a bit like programming, and for good documentation and readability, a common technique is to use "standard" conventions. The following discussion suggests some conventions that are most commonly used by Tinderbox users which will be found in aTbRef.

Thus the document will appear to assert as rules, constraints which are not actually so. Expert users can apply their own, less consistent naming and it is assumed they are expert enough to know where it is safe to do so. Otherwise, qualifying all the above 'rules' will be undervalued. However, it is understood expert users may (within Tinderbox's actual limits) with to use naming styles with which they are more familiar in other areas of their works (other apps, coding languages, etc.)

A few terms are used in the discussion:

- 'CamelCase'. A typographical convention in which an either an initial capital, or an initial lower case letter, is used for the first letter of a word forming the elements of a closed compound, e.g. PayPal, iPhone, MasterCard. Camel case is seen in two forms in Tinderbox: with capital initial letter, e.g. AgentQuery, and with lower-case initial, e.g. wordsRelatedTo().

What things in code can the user name? Individual items lists are discussed in more detail further detail below:

- User Attributes
- Stamp names
- Action code functions
- Action code variables

In addition, some things like the action and export code operators have strict formatting so must be typed correctly. In the case of action code, some code input location support auto-completion, so helping with typing operator names correctly.

Attributes

Attribute names. User attributes (only) can be named by the user and should follow the CamelCase convention seen with system attributes. Attributes are always a single word (no spaces). At simplest: start with a capital, Roman letter (i.e. A-Z), accents and non-Roman scripts are allowed, then capitalise each compounded word, i.e. 'Price' or 'LastName'. Acronym styling is a choice for the user: 'SourceURL' or 'SourceUrl' are both valid attribute names. However, note that Tinderbox will consider 'SourceURL' and 'SourceUrl' as two different user attributes.

Although there is some support for other scripts, beginners are advised to use unaccented Roman characters whilst getting to know how Tinderbox works.

Abiding by these guideline will avoid surprises. More nuanced aspects are discussed under Attribute Naming below.

System attributes always use the CamelCase beginning with an initial and *cannot* be renamed by the user.

Reference to attribute values . In action code (see below), referring to the value of an attribute to get or set its value is done using a '$' prefix. Thus, $ChildCount refers to *the value of* the attribute named 'ChildCount'. However, within aTbR and much general Tinderbox documentation, inline references in general text which refer to Tinderbox attributes names generally use a $-prefix as a marker that it is an attribute name being cited. The latter is a slight variance with in-app usage but does aid clarity in the articles. In actual action code, the $-prefix *always* indicates a reference to a value.

For example, consider a user creating an attribute named BookCategory, and assigns it a value "Non-Fiction". Then, then the action code in a stamp that accesses BookCategory's value in the selected note would refer to it via the code `$BookCategory`.

Code example attribute naming. A convention, adopted before the Sandbox group of system attributes for testing were added, is to use a left-side attribute name that indicates the type of data to be expected from the right side. This 'MyString' indicates receipt of String (textual) data, 'MyList' a list of values, 'MyDate' a date-object, etc. The reader is not required to use the exact left-side attribute stated but rather to understand the data type they should use for their own attributes

Export codes

Tinderbox export codes are case-sensitive and use the CamelCase convention with a lowercase initial letter. Export codes cannot be renamed. They must be indicated with a caret '^' before and (optionally) after the code, i.e. ^children(“t_item”)^. If the optional closing caret is omitted, Tinderbox guesses where the code ends. For this reason, and to help signal intent, aTbRef always uses a closing caret and suggests the same a standard best practice.

Action code operator codes

Action codes are case-sensitive and use use the CamelCase convention with a lowercase initial letter. Action codes cannot be renamed. Queries, either for agents or for query-based operators use action code, albeit with special query operators. Basic actions take a syntax of 'left-side is set to the value of right-side where either may be as simple as an attribute value or a complex expression resulting in such a value. At simplest, an example is: `$MyColor = "red";`. A left-side Color-type attribute is being set to a literal string value of 'red'. Note that in this context the attribute name in the action code is prefaced with "$" as it is setting a *value* for the attribute.

Action code 'dot' operators

In aTbRef listings, dot operators will be referred to with the data type before the dot, e.g. Color.blue, so that inline dots in text do not get misread as punctuation. In practice, Color.blue would be used with any Color-type attribute:

```
$MyColor.red = "#ff"
```

A Date.minute would attach to a date attribute:

```
$MyNumber = $MyDate.minute
```

Export and Action code arguments (inputs)

In code examples in Tinderbox Help and in aTbRef, or in descriptions, there is a convention where arguments (i.e. inputs or parameters) are named to show data type (string vs. number) or purpose to indicate that input is used. Optional inputs are enclosed with square brackets (i.e. […]) and may be omitted when using that operator. If no inputs are used and there is no clear statement stating trailing empty parentheses may be omitted, include them, i.e. 'code()' not 'c ode()'. This way, the intent of the code is clear. In Tinderbox documentation, inputs use the CamelCase convention with a lower-case initial letter, as with the codes themselves.

Designators

Designators are case-sensitive and follow the Camelcase convention using a lowercase initial letter. Designators cannot be renamed. Care should be taken not to use input values that are the same are designator names. As a result, the convention is that designators used as input are *not quote-enclosed* even though they are strings. By comparison, a string value that is the same as a designator must be quote-enclosed to make clear to Tinderbox that it is not a designator.

For example, grandparent is a note designator. The code

```
$MyString = grandparent;
```

has a completely different meaning to Tinderbox than

```
$MyString = "grandparent";
```

Action code Functions & Variables

Functions and Variables have no stipulated limits on naming but should not use a name already used by any Tinderbox system attributes, export, action code, or designator. A function name is *always* followed by opening and closing parentheses after the name, even if the function has no arguments defined.

aTbRef naming prefixes

Again, for consistency and to aid learning, aTbRef is (moving to) a system of using prefix letters to refer to other objects in action and export code. These are for guidance: users are not required to use these conventions in their own work. If nothing else prefixes both indicate intent and avoid naming-collisions with other codes or note names. If writing code to share with other users (who may have less expertise also consider adopting all or part of these suggestions.

These prefixes are suggested to make reading one's code easier and for distinguishing among different elements of a document. It is suggested that single, initial lower-case prefixes can help indicate what kind of element is being defined. Th recommended prefixes are all single-letter and lower case (assuming use in a case-sensitive context):

- 'p' for prototype names. Examples: pCharacter, pStudent.
- 't' for export template names. Examples: tMain_Report, tSummary.
- 'f' for function names. Example: fSetColorRed().
- 'i' for function arguments (i.e. inputs). Note, these effectively define a variable of that name **within** the scope of the function. Example: fSetColorRed(iNoteList), fProcessFeatures(iProductName).
- v for variable names. Examples: vDessert, vLinkList.
- 'a' (or 'an' for readability) for loop variables, such as with .each(). Examples: aProduct, anItem.

For the last three (all types of variable) it can—depending on complexity/context—be useful for the name to indicate the purpose and/or data type of the variable. Examples: 'iProductName', 'vLinkList', 'aProduct'.

Code examples using attributes with a 'My' prefix

In documenting action code, examples are given where pertinent. In order to help learners, where a left side attribute is appropriate, an attribute is used whose name indicates the type of data returned by the right side of the example.

In the majority of cases the returned value is is an explicit or implied string:

```
$MyNumber = $MyDate.hour;
$MyString = $MyString.words(-1);
```

But other types do occur:

```
$MyInterval = interval($Created,$Modified);
```

Processes returning a list are shown as returning a List:

```
$MyList = $MyList.unique;
```

But you can chose a different approach, if you have a different result in mind

```
$MyList = collect(children,$FavFruit); $MyList is [Apples;Oranges;Pears;Apples]
$MySet = collect(children,$FavFruit); $MySet is [Apples;Oranges;Pears]
$MyString = collect(children,$FavFruit); $MyString is [Apples;Oranges;Pears;Apples]
```

Above the first case is the best generalised example. But if the source has duplicates, you can de-dupe it if desired simply by passing a list (i.e. multi-value) to a Set-type attribute as shown in the second example. Possible, but of less use is passing the kist ti String where you receive a *single* string of all the list value, separated by semi-colons (essential a list in its raw form).

This same choice of left-side attribute can word in other context:

```
$MyNumber = $MyColor.blue; (gives 255)
$MyString = $MyColor.blue; (gives "255")
```

In a few cases an action operator requires no left-site attribute as the process returns nothing, but instead carries our the specified action:

```
action($Text("Test-stamp"));
stamp("Do Stuff");
$Text.speak("Tessa");
```

Two more conventions worthy of note. Designators are by convention not quoted, in this case the 'children' forming the first input argument.

```
$MyList = collect(children,$Name);
```

Also when using note titles as offset addresses for an attribute reference, a $Name value is quoted

```
$MyInterval = drivingTimeTo("Swarthmore");
```

but a $Path values is not:

```
$MyInterval = drivingTimeTo(/places/faves/Swarthmore);
```

Those designator and title path quoting 'rules' are not hard-and-fast, but unless experienced enough with code to understand possible edge cases, most users would do well to treat them as actual rules, in order to avoid unexpected outcomes.

Examples are also deliberately pedantic over end-of semi-colon expression markers. These all work:

```
$MyString = "Hello world.";
$MyString = "Hello world."
$MyString = "Hello world"; $MyNumber = 2;
$MyString = "Hello world"; $MyNumber = 2
```

as do expressions if split onto different lines:

```
$MyString = "Hello world";
$MyNumber = 2;
```

But, if the attribute or stamp holding the code has more than one expression, you must place a semi-colon between expressions. So, this is wrong:

```
$MyString = "Hello world" $MyNumber = 2
```

Tinderbox might correctly guess you are using two discrete actions, or not. Better to close your expressions with a semi-colon, even if only to remind yourself

In, turn this leads to another confusion. Semi-colon expression delimiters are **not** used in queries. This is not so confusing as at first sight as queries are used in very few contents:

- agent queries.
- the *condition* in a if(*condition*)
- find(*query*)

Semi-colons turn up in one more place, in literal lists:

```
$MyList = [ant;bee;cow];
```

Results in adding *three* discrete value to $MyList.

Tinderbox ignores white space within/between expressions. These all work the same:

```
$MyString = "Hello world"; $MyNumber = 2;
$MyString = "Hello world";$MyNumber = 2;
$MyString="Hello world";$MyNumber=2;
```

The same holds for white space in input arguments:

```
$MyList = collect(children,$FavFruit);
$MyList = collect(children, $FavFruit);
```

However, within a string, the same does not hold true, these set different values:

```
$MyString = "Hello" + " " + "world."; → "Hello world."
$MyString = "Hello " + " " + " world." → "Hello  world."
```

Of course, the latter is in Tinderbox. To simulate that here in HTML , non-breaking spaces have to be used in the TBX as web browsers collapse strings of space to a single one. Thus string "Hello world." in Tinderbox exports as "Hello world." but renders online as "Hello world."

Square Brackets in code operator explanations

In the names and text of notes describing action code and export operators, square brackets may be seen. For instance: **linkTo(scope[, linkTypeStr])**. Generally, these apply to (some) of an operator's arguments.

This notion follows general conventions for documentation of computer code, so should already be familiar to those with some exposure to code documentation.

Optional Use

What the **[ ]** imply is that the enclosed item(s) are used optionally. Thus in the above example the **linkTypeStr** does not have to be supplied for the code to work.

The syntax can be though of as a composite of two forms:

linkTo(scope)

linkTo(scope, linkTypeStr)

Lest it seem odd not just to write out both forms, some operators have multiple optional operators so writing out all possible forms would lead to long lists and make articles less easy to read.

Literal [ ] in code

There are some operators where square brackets are intentional and necessary, e.g. **attribute(attributeNameStr)[keyStr]** where the **[ ]** enclose and indicate the name of the dictionary key **keyStr** being referenced.

A further and very recent use is for nested lists.

Which is which usage: optional or literal?

The key is to read the article if unsure; with experience it will be easier to judge by context. For new users, read the whole article and not just browse the title. Optional content/usage is always described as such.

Deprecated Usage

This section covers notes relating to deprecated usage, i.e. such use is no recommended as it it has been replaced by some newer method. In other words, these notes describe things you *may* be still able to do successfully but which you a advised not to. This could be for one of more of these reasons:

- The feature has been removed, i.e. no longer supported at all.
- The feature has been replaced. Legacy support may continue but should not be assumed, so update code to the new syntax.
- The feature does not work properly or in contexts deprecated may not give expected outcome.

Many deprecated action and export codes are no longer listed, although they remain in older versions of aTbRef.

- References to deprecated aspects of Tinderbox

References to deprecated aspects of Tinderbox

This is a listing of notes referring to deprecated usage. Specific deprecated codes have separate discrete listings for export and action code. Deprecated usage:

- (!$AttributeName) (i.e. a short form test for no value)
- ^basicLinks([start, list-item-prefix, list-item-suffix, end])^
- ^children([template][,N])^
- ^outboundLinks([start, list-item-prefix, list-item-suffix, end])^
- ^value(expression)^
- $BooleanAttribute != false
- $DateAttribute == date("date")
- AccentColor
- Action and query code
- AgentCaseSensitive
- aTbRef's HTML export templates
- Attribute name styles in listings
- Attribute values
- attributeEncode(dataStr)
- Attributes
- Attributes - $ prefix notation
- AutomaticIndent
- backups folder
- Base
- Basic Comparison Operators
- Bend
- Caret delimiters: export code operators
- Case-sensitivity
- child[N]
- Coding conventions
- Color-Type Attributes
- Color2
- Colour Swatches
- Command Line
- Compound Actions
- Defining 'group' list objects—one or more items—in action code
- Defining a function
- Deprecated attributes
- Deprecated Usage
- Dictionary-Type Attributes
- dictionary(dictionaryStr)
- Dictionary[keyStr]
- Direction
- Displayed Attributes replace Key Attributes
- Displayed Attributes sub-menu
- Displayed Attributes table
- DisplayedAttributes
- DisplayedAttributesDateFormat
- DisplayedAttributesFont
- DisplayedAttributesFontSize
- Dollar-sign prefix: attribute references
- EvernoteNotebook
- Explicit declaration of dictionaries using curly braces
- Explicit declaration of lists using square brackets
- Export Codes - Full Listing
- Exporting code samples
- favorites folder
- format(dataStr, formatStr[, additionalArguments])
- FormattedAddress
- Full Operator List
- HideDisplayedAttributes
- HideKeyAttributes
- HTML Export: ^^value^^ vs. ^^get^^
- HTMLMarkdown
- HTMLOverwriteImages
- Interval-Type Attributes
- IsSeparator
- KeyAttributeDateFormat
- KeyAttributeFont
- KeyAttributeFontSize
- KeyAttributes
- Keywords for Date-Type Attributes

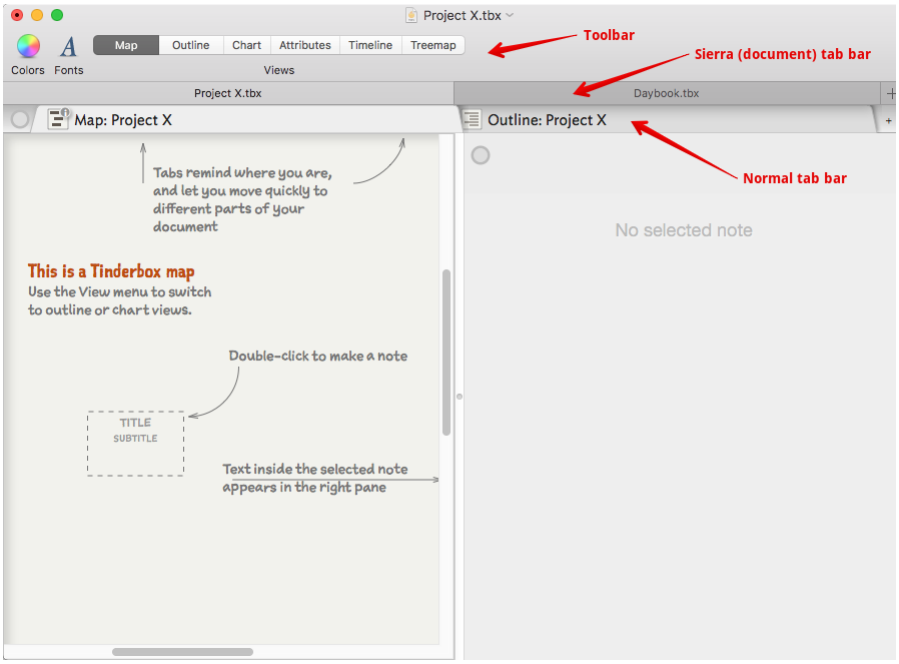
- LeafBase
- LeafBend
- LeafDirection
- LeafTip
- linkedFrom(scope[, linkTypeStr])
- linkedTo(scope[, linkTypeStr])
- List-Type Attributes
- List/Set.at(itemNum)
- MapBackgroundAccentColor
- MapBackgroundColor2
- MapNameSize
- MapPrototypeColor
- MapTextSize
- mt_allow_comments
- mt_allow_pings
- mt_convert_breaks
- mt_keywords
- Note Key Attributes attributes - replaced by Displayed Attributes
- OPML Export
- Optimising code for performance
- OutlineColorSwatch
- OutlineNameSize
- OutlineTextSize
- Paths
- prevSibling
- Quoting Regular Expressions
- ReadCount
- RSSChannelTemplate
- RSSItemLimit
- RSSItemTemplate
- Separator
- Set-Type Attributes
- ShowTitle
- Square brackets: lists and nested lists
- String.contains(regexStr)
- String.icontains(regexStr)
- String.json()
- System tab
- TextAlign
- TextExportTemplate
- TextPaneRatio
- TextPaneWidth
- TextSidebar
- Tip
- TitleBackgroundColor
- TitleFont
- TitleForegroundColor
- twitter(usernameStr, statusStr)
- Using Export Code
- v9.5.2b606 (28 Feb 2023)
- v9.6.0b632 (1 Aug 2023)
- var
- WeblogPostID
- WordCount

Windows

All TBX documents will show at minimum of one document window. Other document windows, the Inspector or and secondary windows my be open.

- Document Window
- Secondary windows
- Inspector

Document Window



At the top of the window, the caption bar contains the document's filename and standard OS controls (via the pop-up) to rename/move the file. The file icon if Cmd-clicked shows the path to the file's stored location from root.

Below the caption bar, and hidden by default is the toolbar.

Below the toolbar, also hidden by default, is the OS-level document tab bar.

Beneath these, and visible by default, is the main toolbar and link park. A Tinderbox document window will always contain a minimum of one (view) tab. By default new blank windows open with two tabs; the second tab may then be closed or additional tabs opened as required.

Beneath the tab bar the window is split into 2 halves by a (moveable) splitter bar. The position of the bar is stored by the tab. Each tab can have a different splitter position.

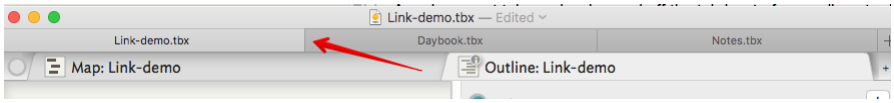
The left side forms the 'View pane', holding a view such as a Map or Outline. This pane equates to a major view type window in pre-v6 Tinderbox.

The right side of the window is used for the 'Text pane' and which equates to a text window in pre-v6 Tinderbox.

The View pane, Text pane and window Tab bar, can all take 'first' focus, i.e. when switching focus into into the app form a different app or switching back from the Inspector to the document window. Note: clicking in the Displayed Attributes area of the Text pane should be avoided as it does not set focus in a meaningful location.

- OS-level Document tab bar
- Toolbar
- Tabs and the tab bar
- Multiple document windows
- Document Windows, Tabs and selections
- Breadcrumb bar (for hoisted main views)
- Window saved tabs Gallery pane
- Command Bar
- Using the pane splitter
- View pane
- Text pane

OS-level Document tab bar



From OS 10 Sierra, a document-level tab-bar can be shown, the default is to be hidden. If displayed it shows one tab per open TBX document. It is placed below the toolbar and above the normal (internal) tab bar. The tab bar's visibility is toggled via the Window menu.

This bar allows document windows to be 'docked' to a single tab bar. Discrete document windows within a single document can be docked here.

Any document tab can be dragged off the tab bar to form a discrete document window.

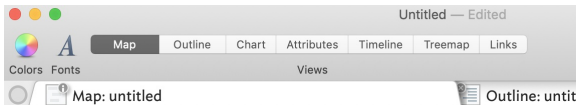
Although windows from different documents can be docked together, this does not mean in-document tabs can be dragged to different document.

The Window menu has a command to merge all windows which will create one document window with all existing windows/documents as tabs within it.

The tab bar has a '+' symbol at the right side. If clicked, a new un-saved document is created as a new tab on the current windows tab bar.

If a document with docked windows (document tabs) is opened in a pre-Sierra OS, each tab will open as discrete window.

Toolbar

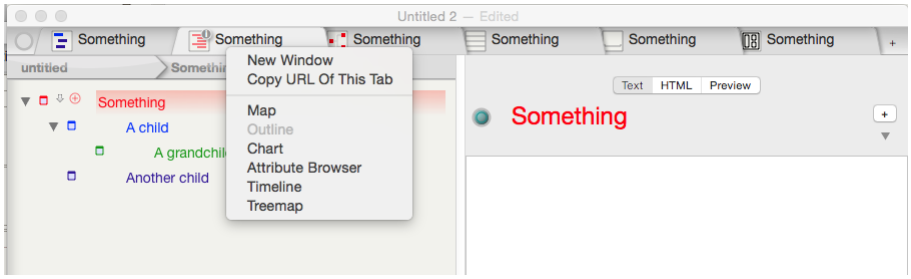


Document windows can optionally show a toolbar in the caption bar at the top of the document window. There are three controls:

- **Colors**. Clicking this shows/hides the OS colour picker dialog.
- **Fonts**. Clicking this shows/hides the OS fonts dialog.
- **Views**. This button bar allows the view type to be changed for the current window tab. The selected view type's button is showed as pressed-in. The 'Links' button is for Hyperbolic view.

Tinderbox remembers whether the toolbar was visible when a document was saved, and if so makes it visible when the document is again loaded.

Tabs and the tab bar



The window tabs are placed at the top of the document window. A Tinderbox document window will always contain a minimum of one (view) tab. By default, new blank windows open with two tabs; the second tab may then be closed or additional tabs opened as required. The tab bar is visible by default but may be hidden (see below). The tab bar also contains the main link park. The view type can be changed view the tab's pop-up context menu.

The selected tab is tinted with the accent colour from the user's macOS account.

Link park. The circle at the left end of the tab is the link park. If dragging a link from a note in either of the view panes and the destination is not visible, drag the link to the parking circle. The circle will then show a green fill. More on link parking.

Tab(s). The man part of the bar is taken up with one or more tabs. The selected tab is differentiated by making it sit in front of the tabs either side of it. The tabs will will expand or contract in width so they fill the tab bar width. If the window is narrow or there are many tabs and the full title cannot fit on the tab the title is clipped and closed with an ellipsis. The tab can be closed by the 'x' icon at its top left. When the active tab, all types except Attribute Browser, show an 'i' icon top right which opens the views properties pop-over.

- **Tab icon** the left of each tab is an icon. As illustrated, each main view type (as in the left window pane) uses a slightly different icon. The symbols in the icon use a tint of $Color of the current container (Map uses the $Color of the first item on the map). The icons seen in the picture are in order: Map, Outline, Chart, Attribute Browser, Timeline and Treemap.
- **Tab title**. The title uses a consistent format of the main view type, a colon, a space and the name of the current container, e.g. "Outline: Something". If the tab is narrow the view type prefix is omitted. Note the view type can distinguished by the icon.
- **New Tab**. At the right end of the bar a '+' symbol opens a new tab if clicked.

Tab Info Widgets

Tab, with the exception of attribute Browser, have a 'i' icon on the top right of the tab. For Map/Chart/Outline views this opens the Edit Background properties pop-over. For Treemap view, it opens the Treemap Properties pop-over. For Timeline view, it opens the Timeline Settings properties pop-over.

Auto-hiding the Tab bar

The tab bar may be hidden (View ▸ Tab ▸ Show Tabs). When hidden, tab bar is "spring loaded" such that when the mouse cursor moves near the top of the window the tab bar will be revealed and remain until the mouse moves away.

Dragging Tabs

Tabs can be dragged to re-order them in the tab bar. Tabs cannot be dragged between document windows (even of the same parent document).

Dragging a tab down from the tab bar opens a new window with (only) that tab.

Opening and Closing Tabs

New tabs are added using the button at the right end of the tab bar. A new tab created this way always re-uses the current tabs's context (i.e. the container on which the tab's main view is focused. The new tab is always added at the right end of the existing tabs.

To close a tab place the cursor over the top left of a tab's icon and a small 'x' button will appear in its top corner. Click this to close the tab. On closing, the rightmost remaining tab will take focus.

The tab bar may be hidden (View ▸ Tab ▸ Show Tabs). The hidden tab bar is "spring loaded": when the mouse moves near the top of the window the tab bar will be revealed and remain until the mouse moves away. Moving away hides the tab bar again, but only after a brief delay; this delay lets you use the breadcrumb bar if you wish.

Saving tabs

See the Gallery pane for saving tabs. This can be accessed from the tab's right-click context menu or via View menu ▸ Tab sub-menu.

Background Tabs

These tabs abs consume little or no CPU time unless or until they are selected. Thus their memory demands are trivial and so the number of tabs in a document should not affect performance.

I doubt there is any reason to avoid adding tabs.

Multiple document windows

Although all documents show a single window by default, it is possible to open additional windows using the File menu.

When a new window is created, it replicates the current window. This includes all tabs with their view type, current scope and note selection.

Document Windows, Tabs and selections

Document windows and tabs within windows treat current note selections in different ways:

- Use windows to view the same note in different views - selection is common to all windows
- Use tabs to use the same view with different selected notes - selection is discrete to the tab

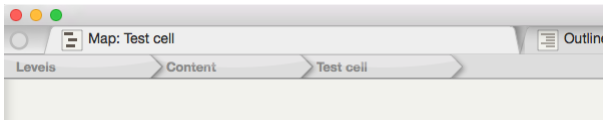
Windows. All windows from the same document share a selection. If one window changes selection the current tab of all other windows changes to the selection, although the scope of the view does *not* change, i.e. the tab in other window may show note text data for a note not currently visible in that tab's view.

Tabs. Each window tab supports its own selection. However, if there are multiple windows , another window may change its selection unexpectedly.

Although new windows replicate all source window tabs it is a good idea to delete all tabs except the desired view type for more predictable selection change caused by window/tab focus changes.

If a tab's view type is changed, the view scrolls to show the selected item.

Breadcrumb bar (for hoisted main views)



The default context for the main pane's view is the root of the document. However, Tinderbox offers a number of methods for setting the context at a deeper level:

- View menu ▸ Focus view. This 'hoists' the context so the selected item becomes the root of the main view (Outline, Chart) or the scope of a Map or Timeline view).
- Double-clicking a note icon in chart or outline view does the same.
- Map view, with a container note currently selected, the down-arrow (↓) will 'drill down' and focus the map on the container child map.

By any of these methods, the context of a main view is now set at a level below root level. To help make this clear, a 'breadcrumb' bar is shown at the top of the main view. A view without a breadcrumb bar is thus at root level. The bar is not shown if not needed so as to preserve screen space for actual main pane content. The breadcrumb bar lists the name of each ancestor container from the document root (at left) down to the current container.

When the cursor is hovered over the breadcrumb bar a tooltip explains its purpose and the possibility of clicking breadcrumb bar items to shift view focus upwards to the clicked item.

In the illustrated example, a map view in a document 'levels' has been focused down 2 levels in the overall document outline, i.e. at $OutlineDepth of 3. The map shows the children of container 'Test cell'. 'Test cell' is a child of root-level container 'Content'. The first crumb is the document itself as it is the container of at root-level items.

It is possible to move focus back up one level by using:

- View menu ▸ Expand view.
- Map view: press the up arrow key (↑).
- Breadcrumb bar, click the item one-from rightmost (the rightmost is the current container so does not move anything).

The breadcrumb bar offers even more flexibility as clicking any item (bar the right-most) will move focus to that level. Clicking the left-most return the document to root and the breadcrumb bar will disappear.

Window saved tabs Gallery pane

Menu View ▸ Tab ▸ Gallery displays a list of current tabs and a list of saved tabs for the current document. An existing tab can be added to the gallery list, making it available later—even if the original tab has been closed. Any saved tab may be re-added to the tab bar of the current document.

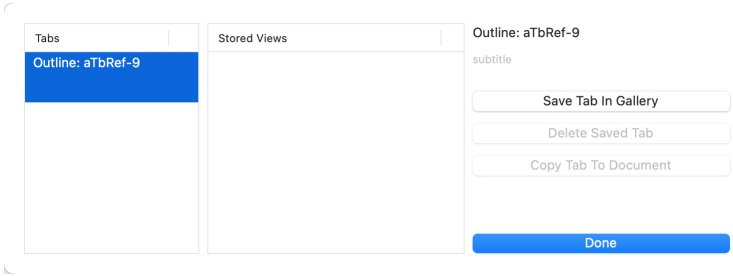
The gallery saved tab data is essentially a document level resource even though it is accessed via a window tab.

Saving a tab retains the tab's view type and view customisations.

The pane contains two lists:

- **Tabs**. This lists the current document's existing tabs.
- **Stored Views**. This lists stored tabs which may be added to the current document.

The right-hand side of the pane shows:



- Name of currently selected tab - in either column. This name, if set is the label shown in that tab instead of the default label of '*view type: root container name*'.
- [subtitle]. This place holder text indicates a space where a short descriptive text can be added/edited for the selected tab. The subtitle is seen *only* in the Gallery pane.
- **Add To Gallery**. The selected (left list) document tab is saved to the Gallery (right list). This means its configuration data (or those parts saved in the tab) is stored in the current TBX document even if the tab is then closed
- **Delete Saved Tab**. (Only for existing tabs)
- **Copy Tab To Document**. This adds a gallery tab (right list) to the list of tabs in the current document (left list)en added, the tab uses the configuration data saved in the tab's XML data.
- **Done**.

At present there is no means to share saved tabs between documents, but those comfortable editing XML could copy the relevant XML code for saved tabs into the source of of another TBX document. If doing so, ensure any *user* attribute referred to by the tabs pre-exist in the new document. Viewing the Tinderbox XML syntax notes' article/sub-article on the <tab> tag will help show where to look for such possible conflicts.

From v9.5.0, saved tabs in the Gallery may now be reordered by drag and drop.

Command Bar

The command bar gives you keyboard access to many Tinderbox commands and lots of Tinderbox information. Its initial placement is tied to the (frontmost) main document window.

To open the command bar, select menu **Help ▸ Commands & Info…** (⌘⇧-U)

The command bar displays only fairly good matches, instead of displaying every match sorted by quality of the match.

Type any word or phrase to see available topics. Arrow ↑ and ↓ keys move through the list. Press **Return** to perform the selected action. **Home** and **End** keys move to the first and last items, respectively. **Esc** dismisses the command bar, as does clicking outside the command bar. Various commands can be used:

- **Open…** Lets you open any Tinderbox document you have recently used, or any Inspector pane. Type "Open " followed by part of the document name. Or, type "Open Border Inspector" to open a particular Inspector sub-pane.
- **Select…** Lets you select any note in the current document; if possible, Tinderbox will scroll to make the note visible. Type "Select" and the note's path or name (if the note's title is unique in this document).
- **Explain…** Provides instant access to some key places in aTbRef, e.g. to explain Tinderbox actions and their usage. For example "Explain Menus". To access more specific topic like an operator, simply type its name, e.g. "collect_if".
- **View…** Access some notable Tinderbox tutorial videos.
- **Attribute values**. Type the attribute name. Clicking on a command bar item that describes an attribute opens the Quickstamp Inspector for that attribute, allowing quick setting or reading of that attribute value for the current note.
- **Check update status**.g. "Can I update"

Using the pane splitter

The two panes in a window's tab are divided by a splitter bar that can be dragged to give more space to either the view or the text pane. By default a tab opens with a 50:50 split between the two halves. The splitter can be moved manually to give a desired ratio of the two panes. The current splitter position is saved and remembered between sessions.

Resizing the document window by dragging its left/right edges or any corner retains the split ratio. If the view pane to text pane ratio is 2:1, dragging the window wider will retain a 2:1 balance between the panes.

A Cmd+drag of the window will re-size it so as to retain the current text pane width, i.e. all extra pane width is added to/removed from the (left) view pane only.

Three shortcuts are offered to toggle one of 3 states:

- Cmd+4 / Text Only. The splitter is moved all the way to the left of the window so the text pane fills the whole window.
- Cmd+5 / View and Text. The splitter is placed so there is an equal division between the left (view) and right (text) panes.
- Cmd+6 / View Only. The splitter is moved all the way to the right of the window so the view pane fills the whole window.

The Cmd+5 shortcut always gives a 50:50 equal pane width allocation and so overrides any pre-existing split ratio. There is no current mechanism to remember and return to a custom pane split ratio after using Cmd+4or Cmd+6.

View pane

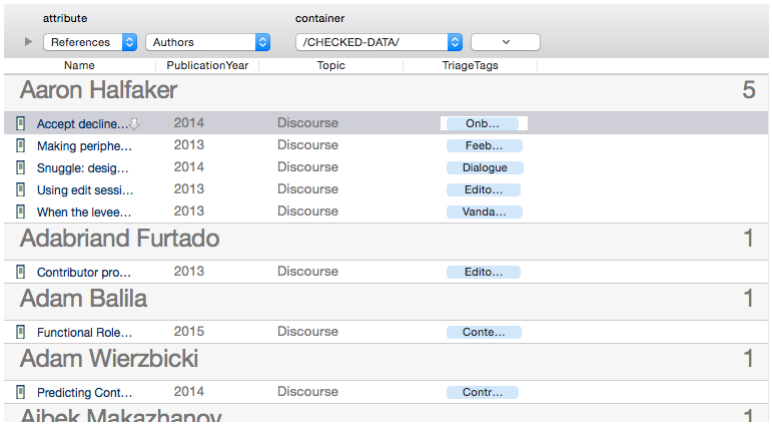
The View pane, the left side of any document window tab, shows one of 8 possible views. Some views have settings that are view-specific and can only be set via a Setting dialog (e.g. Map Settings) accesses via the 'i' icon on the active ta label

Historically Map/Chart/Outline/Treemap were the original views. Then followed: Timeline (v5.6.0), Attribute Browser (v6.0.0), Hyperbolic view (v7.0.0), Crosstabs (v8.0.0). Generally the Attribute Browser and later view store the view configuration in the TBX file as part of their current tab. To save-re-use tabs after they are closed, save the tab in the tab-gallery.

These are the current views available:

- Attribute Browser view
- Chart view
- Crosstabs view
- Hyperbolic view
- Map view
- Outline view
- Timeline view
- Treemap view
- Find toolbar (view pane)
- View filter toolbar

Attribute Browser view



The Attribute Browser shows you the contents of any container in the current document, categorised by the values of a chosen attribute. By default the scope is whole-document. The scope it shown in the right-hand pop-up in the top row of controls in the view's toolbar. By default, all except the first row of controls are hidden. The remainder of the controls can be revealed/re-hidden via a disclosure triangle on the control panel.

The scope and appearance of the view are set via the Attribute Browser controls at the top of the view pane, which includes a search pane for looking up attributes. If a tab using another view type is switched to Attribute Browser view, the scope remains the same. See more detail about all the Attribute Browser controls.

The Attribute Browser can also use column view and as in other views the column data is editable.

Multiple records can be selected at a time, aiding use of stamps and quickstamp in this context.

For attribute data types that support multiple values, e.g. Set and List, notes are grouped for every unique list value present. Thus a note with more than one value may list several times, once under every discrete value in that note's list. Formerly, this behaviour applied to Sets but not to Lists.

Optionally, value groupings can show a count of the number of the items (configured via the view's controls). By default, this is the per-group count for the view's chosen attribute. It can also be for any of the attributes additionally displayed via column view. If the count is enabled, this summary figure is shown at the right end of the category bar. For Number-type attributes, a range of mathematical options are also offered (see browser controls for more).

Dragging a note to a new category changes the note's value for that listed attribute to that of the category. For multi-value attributes, regardless of the category the dropped note receives all of the values for the attribute for the note listing after the dropped note. In such circumstances it can be useful to list the attribute as a column view item to see the full range of values allotted.

Items listed are drawn in $Color and show $Name. Badges are shown, and a link-drag widget is shown after the item name. The icon used for each item is as per Outline view, indicating the degree of text, recentness of edit and in/outbounc links.

A link widget for dragging links is drawn at the right side of the selection highlight in the view pane.

Items can be assigned new category (attribute) values my dragging them into different categories. The attribute changed is the one being categorised in the view pane. For single value attributes, a drag replaces the existing value. For list-based attributes the new vale is added to existing values. Notes can also be dragged into empty categories (i.e. with no current items listed, and into no value categories. In the latter case the existing value(s) are deleted.

The name column uses the same font and type size as used in the outline view at standard magnification, allowing the title font type and size to be altered.

The Attribute Browser will not display notes that are (currently) separators.

Dates are formatted as specified by $DisplayedAttributesDateFormat. Dates displayed in this view are formatted using the medium system date format (i.e. format "n").

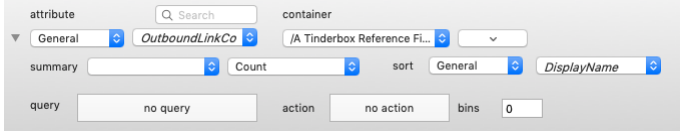
The view's header displays the count of notes in the scope of the current view. If the same note appears in multiple groups, the note is counted separately in each group.

The view accepts View menu Magnify and Shrink commands (and shortcuts) and scales rows appropriately.

From v9.6.0, selected notes may now be copied, regardless of whether the selection is an individual note or more than one note.

- Attribute Browser controls
- Attribute Browser column pop-up

Attribute Browser controls



This control panel is shown at the of any view pane Attribute Browser view. By default, all bar the first row of controls are hidden, accessed via a standard disclosure triangle to the left of the top set of controls. The labels in the top row refer t the controls in the second row of the panel.

attribute. This comprises a pair of pop-ups and a search box (the latter to the right of the label).

- **Attribute group pop-up**. The left hand pop-up selects a system or user attribute group.
- **Attribute pop-up**. The right-hand pop-up is populated based on the selection in the group pop-up. The attribute selected here is the attribute whose values are then summarised in the main part of the Attribute Browser view. Selecting an attribute here is the primary configuration task with this view.
- **Search box**. Placed in row #1 of the controls, to the right of the label. The search box autocompletes attribute names and clicking Return will auto-select both the above pop-ups for the attribute name matched in the search box.

container. More accurately this is the scope of the view.

- **container selection pop-up**. By default a view is whole document (or current container if the view is created by altering the tab view type of an existing tab. The items listed are both immediate children and any descendants of the chosen container.
- **(container sort order)**. The button with a downwards-pointing chevron label controls the view sort order, in terms of sorting of the view's categories. Clicking the button toggles the sort order. When the sort order is reversed, the butto label changes to an upwards-pointing chevron. The 'empty' category—no value assigned for the inspected attribute—is listed last in the default order (its position toggles to first in reversed sort). The sort order of value categories depends on data type:
- **String-based** (all types—including list-based—*except*: Number, Interval, Boolean, Date types). Alphabetical lexical order, case-insensitively. Numbers sort first, i.e. : 1, 10, 2, aardvark, ant, Ant, bee, etc. Note that numerical text values do *not* sort numerically.
- **Number-type**. Ascending numerical for Number-type (negative numbers first). However, see the section below on numerical value 'bins' as numbers are categorised slightly differently to textual values.
- **Boolean-type**. Default is `false` then `true`. Note that there is no unassigned group as a Boolean attribute can only be `true` or `false`.
- **Interval-type**. Ascending based on duration.
- **Date-type**. Earliest to latest. The 'no date' value, i.e. `"never"`, list lasts as for other types' defaults.

summary. The controls are for per-category summaries. By default there is no summary. The first pop-up lists the $Name (default) plus an attributes shown in column view. The second control offers a choice of operation to apply to the choice. The default is no choice. The choices offered are:

- non-Number type attribute selected:
  - **none**. (default)
  - **Count**. This sums the number of items in that category with a value for the attribute. Thus in the basic Name/Count configuration the summary is a count of items in that category, as each note has a (single!) title.
  - **Count Fraction**. For each category, this shows the category count as well as the total count (i.e. the total number of items in scope).
- Number-type attribute selected offers a larger range of summary types:
  - **none**. (default)
  - **Count**. (only option for non-Number type attributes)
  - **Total**. The total value of all the category's items for the selected attribute.
  - **Mean**. The mean value of all the category's items for the selected attribute.
  - **Minimum**. The smallest value of amongst all the category's items for the selected attribute.
  - **Maximum**. The largest value of amongst all the category's items for the selected attribute.

sort. Within a category, the items can be sorted on a chosen attribute. The default is $DisplayName. Attribute Browser categories are sorted by locale; thus, diacritical marks should not create unexpected sort orders. When categories are sorted by the summary, ties are sorted by the category name. A pair of group/attribute pop-ups allow the sort attribute to be selected.

- **Attribute group pop-up**. The left hand pop-up selects a system or user attribute group.
- **Attribute pop-up**. The right-hand pop-up is populated based on the selection in the group pop-up.

query. A query can be added to further refine the overall choice of included items. The query applies *within the current container scope* as set via the **container** controls (above). Clicking in the input box opens a query pop-up. The descripti entered in the pop-up is used as a label in this box.

action. Akin to an agent action this optional action code is applied to all items listed in the view. Clicking in the input box opens an action pop-up. The description entered in the pop-up is used as a label in this box.

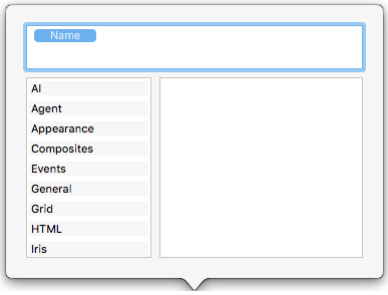
bins. (Default: 0) Only displayed/used where the **attribute** analysed is of Number-data type. If a figure is entered the view divides the difference of the lowest and highest attribute value to the desired number of *equally-sized* bins. It is not possible to Set a floor or ceiling for the lowest/highest bins. There is no stated limit for the number of bins that may be defined.

Attribute Browser column pop-up

This pop-up is shown by right-clicking the title bar of the view pane of an Attribute Browser view.

To add an attribute, as a column either:

- Type the desired name in the box in the upper section of the pop-over. Autocomplete is offered for all currently defined system and user attributes. The box is blank if no Displayed Attributes have yet been defined. If large numbers of attributes are added, the box will auto-expand to keep existing attribute choices on view.
- use the two bottom lists. First, select the attribute group in the left pane then tick the appropriate attribute in the right pane (un-ticking will also remove an existing item).



To delete a current Displayed Attribute, click to select the item in the top list and delete it. Or, use the lower two panes to find the item and un-tick it. The first column remains $Name. If only one column is set and it is another attribute, it will be automatically changed to $Name.

To re-order columns, click on an item in the upper list box and drag it to the correct location in the list. The first column must be $Name, if altered to something else, on close the dialog will reset that column to $Name.

To close the pop-over click outside it or press Escape. On closing, the changes made (above) are effected.

The meaning of the styling of different rows (bold, strikethrough, etc.) is explained here.

Chart view

A chart view shows the hierarchical structure of the document very clearly, as a tree chart. A chart is also handy for seeing many notes in one window. It is easy to re-structure the hierarchy of the document in a chart window, by dragging parts of the hierarchy to other parts of the hierarchy. The scope of the chart depends on the selection when the chart view is called. If in root view and nothing is selected, the chart represents the entire outline, otherwise it is of the selected item and all its descendants.

Subtitles appear in chart view. Chart View has a Chart options popover, accessible by clicking the Info button on the chart view's tab. The popover allows change the chart style and to adjust the width and spacing of chart items. A choice is offered of either a left-to-right (as previously) or top-to-bottom arrangement of the chart.

A note icon with a pointed right end is a container. By default a new chart is drawn with all branches expanded. However, any container note can be collapsed (like with an Outline) by clicking the triangular area at the right edge of container icons. Collapsed containers show a number, e.g. "+17", indicating the total number of descendants (i.e. children, grandchildren, etc.) that are in the hidden branch. The count includes agents and aliases but excludes separators and adornments as neither of these types of object is drawn in Chart view.

A chart view does not show any of the links in the document. A chart view does not differentiate between notes and agents as in outline or map view (through if a container holds *only* aliases, it is most likely an agent). Chart view dog-ears do not show the ageing colour (as in outline view, etc.).

Charts can wrap their titles to multiple lines and editing-in-place is supported.

The note's badge is shown to the left of the title. Right-clicking in this area opens the badge menu.

The note or agent icon is the same as that used in Outline view and shows the presence/amount of text, and links, and can be used to select prototypes.

The link icon for dragging links is shown, for the selected note, between the note icon and the badge.

Item icons and titles are drawn in $Color.

Chart view is *not suitable for very large outlines* , e.g. the whole of a document such as aTbRef, so avoid using this view for very large numbers of notes. This is because Tinderbox runs out of virtual space in which to draw the overall outline (i.e. it is an internal limit within the app). Reserve chart use for small files or just sections of larger files. Large charts can take a long time to render and may not scroll smoothly due to their sheer size (N.B. outlines are more compact). The conventional left-to-right Chart View is somewhat more efficient in its layout.

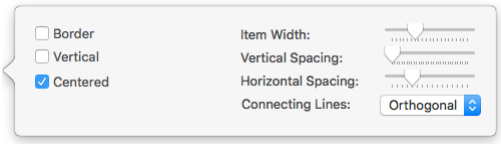
Chart tabs remember and restore the expansion state of their view.

- Chart Settings pop-over

Chart Settings pop-over

This is the configuration pop-up for the current Chart view:

- **Border**. Default: un-ticked. If ticked a thin border is drawn around each item on the chart.
- **Vertical**. Default: un-ticked. If tick-ed, the chart is drawn from top-to-bottom of the screen as opposed to the default left-to-right layout
- **Centered**. Default: ticked. If un-ticked, the item top aligns (default layout) or left aligns (vertical layout) with the line from the parent item.
- **Item Width**. Scrubber control sets the width of chart items
- **Vertical Spacing**. Scrubber control sets the amount of vertical spacing between chart items.
- **Horizontal Spacing**. Scrubber control sets the amount of horizontal spacing between chart items.
- **Connecting Lines** pop-up:
  - **Orthogonal**. (Default) Lines interconnecting items are drawn using horizontal and vertical sections.
  - **Straight**. Lines interconnecting items are drawn using straight lines.
  - **Curved**. Lines interconnecting items are drawn using bezier curves.



Crosstabs view

The Crosstabs view allows exploration of the values of two selected attributes in the current document, or a limited scope within the document. The view displays a table in which one attribute's values are listed horizontally and the second attribute's values are displayed vertically. The following attribute types are supported: Number, String, Date, Boolean, List, and Set.

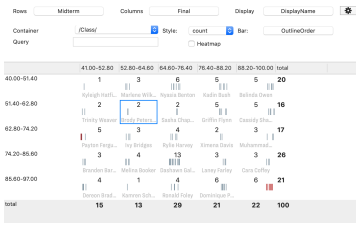
By default, the scope of the view is of the whole document. This, and the attributes to be tabulated, can be set using the view's control panel.

At the bottom of each cell that is not empty, Tinderbox shows a display attribute of one note in the cell. The Display attribute is typically $DisplayName, but may be any attribute.

In the centre of each crosstabs cell is a bar that shows the relative location of each note in the cell in the document's outline. The left edge of the bar corresponds to the first descendant of the chosen container, and the right edge corresponds to the last descendant of the chosen container. A vertical line representing each note appears in the note's outline colour.

Hover the mouse over a crosstabs cell to browse through other notes in that cell. The first note in outline order is viewed by placing the mouse at the left edge of the cell and the last note in outline order is viewed by placing the mouse at the right edge of the cell. Click to select the current note and to see its text and Displayed Attributes. The ↑ and ↓ keys can also be used to select the next and previous notes in the cross tabs cell, and is a more practical way to navigate the cell contents if there are many items.

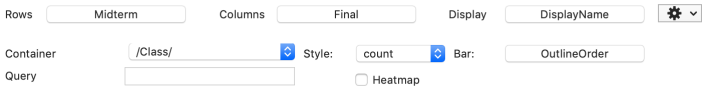
It is possible to make an agent that collects the items corresponding to interesting cells of the table, see the 'Copy Agent For This Cell' in the content menu for a selected cell. Via that menu it is also possible to apply an action or a stamp to all the notes in a cell.



- Crosstabs controls

Crosstabs controls

The attributes to be viewed are displayed in the two buttons at the top of the view. Pressing the **Rows** or **Columns** button to change the target attributes. The resulting popover window also lets you set the maximum number of rows or columns to use; setting the limit to the upper bound (25 rows or columns) asks Tinderbox to display a separate row or column for each unique value. For Boolean attributes, there are only ever two bins. Set and List attributes always have unlimited bins, allowing each tag or element its own bin. For Date attributes, the date "never", if present, is always placed in its own bin.



At the bottom of each cell that is not empty, Tinderbox shows a display attribute of one note in the cell. The **Display** attribute is typically $DisplayName, but may be any attribute, use the button to set the desired attribute.

The gear-wheel icon opens a pop-up menu offering a number of ways to export data from the view:

- Export As CSV. This exports the table including column/row labels and totals to a comma-separated variable (CSV) file; an OS file same dialog is shown for saving the file.
- Export As TSV. As above but in Tab-delimited text (TSV) format.
- Copy TSV To Clipboard. The table in TSV format (see above) is copied to the clipboard.

In the second row of controls, the **Container** pop-up control allows the view's default scope of 'entire document' to be changed to any container in the current document.

The **Style** popup lets you choose, using a pop-up menu, to display either:

- the **count** of notes in each cell
- the **percentage** of in-scope notes in the cell.

The **Bar** shows, by default the relative location of each note in the cell in the document's outline (using $OutlineOrder). The left edge of the bar corresponds to the first descendant of the chosen container, and the right edge corresponds to the last descendant of the chosen container. A vertical line representing each note appears in the note's outline colour.

Query allows the container choice to be (further) constrain using a query added in the box. As the query is applied to the container

A tick-box offers the optional of a **Heatmap** that will colour cells in proportion to the number of notes they contain, using the system accent colour ($AccentColor) for sparsely-populated cells and the complement of that colour for heavily-populated cells.

The notes viewed in the crosstabs view may be all the notes in the document, or may be restricted to those inside a specific container. Select the desired container from the **Container** popup menu.

Hyperbolic view

The hyperbolic view shows notes that are linked to or from a specific note—the 'focus' note—which is initially the selected note. The focus note is identified by a thin red border.

Starting at the focus note, Tinderbox identifies all the notes that link to that note, and that to which the selected note links. This is repeated until all notes reachable from the selected note, and all notes from which the selected note can be reached, are included.

Those notes are then arranged in a view with the selected note at the centre. Notes linked to that note are arranged radially around that note, and notes linked to those notes are arranged radially around them.

Finally, this tree is projected onto a hyperbolic plane; notes near the centre are large, and notes further from the centre are progressively smaller.

Hovering the mouse over a note will reveal its full name as a tooltip. This is often useful out near the rim of the view where note's appears smaller.

Web Links and Prototype links are ignored in hyperbolic view.

Navigating the Hyperbolic view.

Some controls for altering the view's setting are shown in the view toolbar.

Note shape and link colour are retained. Note that the normal map shapes map be distorted slightly reflecting the way they are drawn on the hyperbolic surface. Lines that represent links in the spanning tree are drawn as arcs, conforming to the underlying hyperbolic geometry.

The view supports mouse-wheel and two-finger swipe scrolling. Holding down the option key (⌥) when using the mouse-wheel or two-finger swipe scrolling changes the magnification.

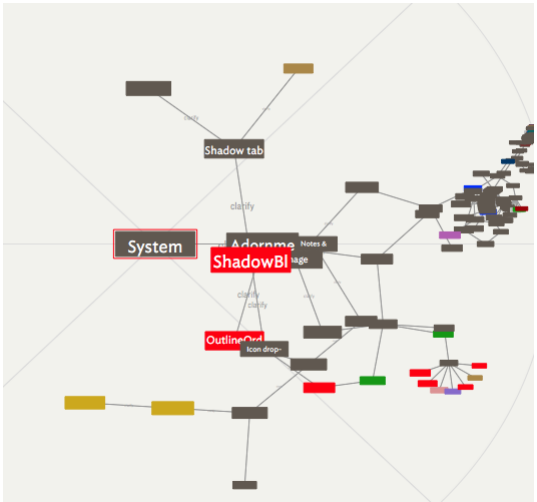
A context menu for the selected allows traversal from the selected note using any outbound basic or text link. The destination note is selected but does not take focus. The context note also allows the selected note to be deleted.

It is possible click-drag from any note to another note and create a new link. Or, by dragging a link to a blank part of the view and releasing the cursor, a new linked note is created. The Link creation dialog opens allowing the new note to be named, before the other dialog controls are configured. If not named the note is called 'untitled'. If you change your mind and do not name the new note, the newly-created note and its link are automatically deleted.

Existing links, for the selected note, can be reviewed/edited using the Browse Links dialog.

The background of the view draws only the limit circle is drawn, and not the axis guides as in the past.

Hyperbolic view has been revised to provide better layout, especially in large a complex documents. To build the hyperbolic view, Tinderbox constructs a spanning tree that starts at the focus note. It examines outbound links from that note; if they lead to a note not already in the tree (i.e. the view), it adds that link and the destination note to the tree. After checking all the links in the focus note, it now repeats the procedure with each unvisited note added to the tree, continuing until it has either (a) exhausted the list of unevaluated notes, or (b) exhausted the list of unevaluated outbound links. In the resulting spanning tree, all links that connect notes in the tree to other notes in the tree, but which are not part of the spanning tree, are considered 'cross links'. There is a view settings option to hide these cross links.



For new views, the '*untitled' link type is used for initial tree display construction.

The contextual menu of the focus note (i.e. with red outline) omits a 'Delete option, since deleting the focus would leave nothing in the view.

See also:

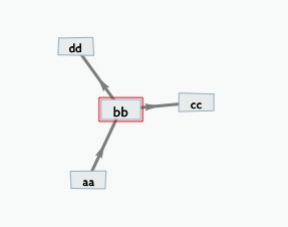
- Hyperbolic view controls
- Navigating hyperbolic view

Hyperbolic view controls

In hyperbolic view a number of controls are shown at the top of the view:

- **Refresh** (from v9.6.0): tell the view to restart the force directed layout.
- ~~**Spacing** (pre v9.6.0): controls the distance (*in hyperbolic space*) between notes, and simultaneously adjusts (again, in hyperbolic space) the size of each note. So, increasing the spacing makes notes farther apart and bigger. Replaced by the **Refresh** button (see above).~~
- **link type: drag to reorder** (from v9.6.0): a list of all link types in use in the document. Each individual can be selected both/either for whether it is used to build the main view tree, and/or whether it is presented as a crosslink. Dragging a link type higher in the list increases its priority; high-priority links will be included in the spanning tree in preference to lower-priority links.
- ~~**Highlight** (pre v9.6.0): emboldens a selected path. Replaced by the link type panel (see above).~~
- **Aspect Ratio**: changes the ratio between the height and width of the notes. Again, this is in hyperbolic geometry; notes not near the centre are not rectangles.
- **> disclosure control** (v9.6.0+): this allows vertical expansion of the control panel, making more of the link type list visible.
- ~~**Cross Links** (pre v9.6.0); Unticked (default) turns off cross links, which frequently obscure complex documents.~~ This is omitted as the link-type table (above) offers control of the view's tree and cross-link construction at per-link-type granularity.
- **Scale**: At small scale (fully to the left), the entire with of the hyperbolic disc is shown. If the view is wider than it is high, the regions near the north and south poles will be offscreen. At larger scale, more of the edges are moved offscreen so you see only near the centre of the view. Pinch-zoom adjusts scale, too, as does "smart magnify" (two-finger double-tap).
- ~~**Spread** (pre v9.6.0): Determines the angle between child nodes. Replaced as part of new force-directed layout method.~~

Navigating hyperbolic view



From v9.6.0, the spread control has been removed from hyperbolic view as a force-directed layout (that is (re-)initiated with the **Refresh** button) does this better. When force-directed layout is running, moving the centre of the display will suspend force-directed layout for a second. This avoid problems with misplacement of very distant notes, because a small misplacement of a distant note can have very large effects.

From v9.6.0, Hyperbolic view displays only notes within 8 links of the focus note. Although this means some linked notes are note currently displayed, it keeps the display within reasonable bounds and avoids numerical instability towards the edge of the display.

Dragging the background of the hyperbolic view moves all the notes, letting you examine different parts of the link structure. From v9.6.0, panning the hyperbolic view is restricted so that at least part of the graph always remains in view.

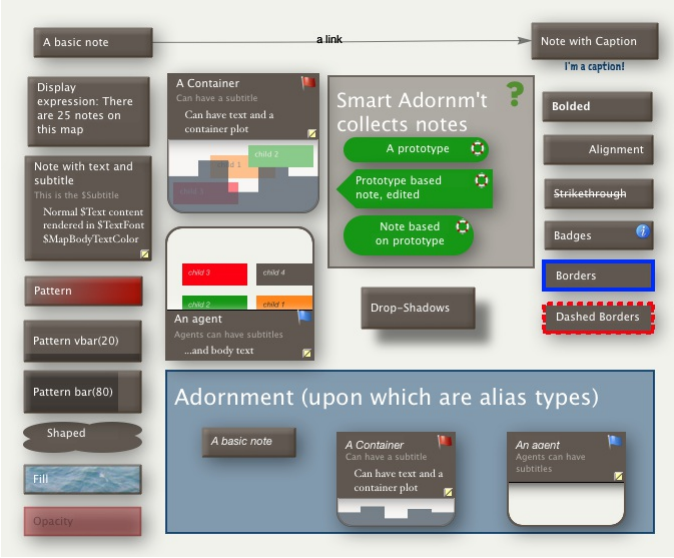
Clicking a note selects that note in the tab's text pane. Selecting a note in Hyperbolic View will move that note to the centre of the view, leaving the tree unchangedepending on the view's complexity, it may appears to change bit this simply reflects the re-centring of the view.

Double-clicking a note makes that note the view's focus note *and* will recompute the structure of the document based on that note. This may change which links are primary and which are considered cross-links.

To take select a note and take focus needs a click followed by a double-click.

A context menu (based on the selected item) allows traversal from the selected note using any outbound basic or text link. The destination note is selected but does not take focus.

Map view



The map view is the primary view type used in the View pane of a window tab. It offers he opportunity to record data in an unbounded map, leaving the structure of the layout entirely to the user.

For articles about the map view interface, scroll to the end of the article.

A map view shows notes as rectangular icons. You can move these icons around in the map however you like and drag links lines between them. A map view shows one level of the document's hierarchy, by showing the children of a note as contained within its box. A map only shows immediate siblings, not other notes at the same outline level but with a different parent note.

You can re-structure the hierarchy of the document in a map window by dragging the icon representing a note (and any icons it contains) into/onto another note's in the window. If the recipient note is not already a container it becomes one.

If notes were not originally created via the map window they may appear overlapped, or even obscuring one another, or offset some distance from the centre of the map; you may have to look around to find all the notes when first using a give map view. Over time Tinderbox has improved auto-placement of map icons.

For any container there is only ever one map, even if it is opened in two different tab main views.

Tip: when developing a big document do no use the root level for your map. Instead, first make a container at root level and start your big map inside this. The approach offers several practical benefits:

- Non-map content like prototype containers, export templates, general utility agents and such can be placed under different root level containers and thus not appear on the map.
- Addressing notes via action code is less ambiguous (easier!).

Display of links

A map view also shows many of the links between notes. If both the source and destination of the link are in the map, the link is drawn as a arrow between the boxes, with the link name at the midpoint of the arrow. By default the arrow is curved but straight arrows can be used instead (setting link link types). If the source or the destination of the link is outside the map, the link is drawn as a short arrow departing from or arriving at a box.

The map view is the only main view to show actual link lines in the view.

Moving notes in the Map

To move a note within a map, click the note with the arrow tool and drag it to another part of the map. If you move a note to the very edge of the map, Tinderbox creates 'more' map. This makes sure that a note is never hanging off the edge; you can always move the map far enough to see all of every note. Drag a note inside another note to make it a child of that note in the hierarchy. A careful hand on the mouse can drag a tiny note out of its parent in the map. New notes and dragged notes may snap to the map grid. Shift-drag constrains the drag to vertical or horizontal movement.

Map items (other than adornments are drawn in outline order. This ensure that where items overlap, the most recent item (or that highest in outline order) is drawn last.

The map cleanup feature allows for resetting some or all of the map icons to one of a number of layout patterns.

The map view offers a large number of ways to improve visualisation of data.

Using the document root for maps

Tip: when developing a big document do no use the root level for you map. Instead, make a container at root level and start your big map inside this. The approach offers several practical benefits:

- Non-map content like prototype containers, export templates, general utility agents and such can be placed under different root level containers and thus not appear on the map.
- Action code can use note names rather than paths, i.e. values for note 'NoteB' can be addressed via `$WordCount("NoteB")` rather than `$WordCount(/NoteB)`.

More on the map view interface:

Adding notes to an existing map

Historically, adding new notes to an existing map made the note the youngest sibling of the current container, i.e. it listed last in order when viewed in outline view. Behaviour is as follows:

- For notes added via a double-click on the map background, the new note is added as youngest sibling in the (map) container, i.e. last in $OutlineOrder.
- If a note is already selected, and a note is created other than via double-click, the new note is added as as next younger sibling to the previously selected note (i.e. $OutlineOrder + 1 to the selected note). If there is room the new note is placed to the right of its previous sibling
- If a note is moved into a collapsed outline container or onto a subordinate map (by dragging it, in map view, onto another note or an existing container), it is added as the first sibling.
- If a note is added to a map via an expanded container it is inserted into the map with the sibling order value appropriate for its new outline view order ($OutlineOrder) position.
- For multiple sibling notes added via drag drop, the first item in the selection conforms to either of the last two conditions above. The other items in the selection are inserted in existing their sibling or parent/child relationship within the source selection.
- A pasted note/selection is added as first sibling unless a note on the map is selected, in which case it becomes the first sibling on the map (first child of the container).
- A dropped/moved container acts as for a single note in positioning terms whilst retaining its descendants.

Note that subsequently moving the new notes around on the map has no effect on outline order ($OutlineOrder), only on X/Y co-ordinates ($Xpos/$Ypos) as was previous behaviour.

When adding notes via a keyboard shortcut:

- **Return** (↩) creates a new note to the right of the selected note, if space is available.
- **Control**+**Return** (^+↩) creates a note to the left of the selected note.
- **Control**+**Option**+**Return** (^+⌥+↩) creates a note below the selected note.

If the initially-chosen location is occupied, a different location will be chosen.

Adornments

Adornments are a **map view only** feature (though they *may* appear in Timeline views if dates are set). They are used to provide a means of adding visual elements to the background of the Map view. Some general points about adornments also described under general concepts.

Adornments can:

- act as prototypes.
- show on timelines if start and end dates are set.
- be of special types:
  - a 'smart adornment', having an agent query.
  - a 'geographic adornment', having geographical data settings.
  - an 'image adornment', displaying an image as a fill.
- use visual grids to delimit visible sections within the adornment—see adornment grids.
- display badges and flags
- use display expressions and hover expressions.
- use a subtitle.
- react to notes moving fully or partially onto ($OnAdd) or fully off ($OnRemove) the adornment. See adornment actions.

Adornments can **not**:

- have child objects (and has no child map - navigating 'down' into an adornment is not possible).
- be matched by queries.
- be referenced by designators.
- display $Text.
- have aliases.
- be exported.
- be linked to/from notes.
- use table expressions (but see grids above).
- be counted by $ChildCount or $DescendantCount, but see $AdornmentCount below.

Creating adornments

Created via the Create Adornment menu item in the Note menu or the map's context menu. A map may have one, many or no adornments. Different types of adornments—normal, smart, etc.—can be used on the same map. Additional setting are used to make a normal adornment into a smart, geographic or image type of adornment.

Positioning and stacking

New adornments are placed *on top* of existing adornments. This is the reverse of existing behaviour for notes but should be more intuitive for use. The change is achieved by always adding new adornments in $OutlineOrder in front of existi ones rather than at the end, as the lowest outline order item is always drawn on top.

Note that adornments do get counted in the $OutlineOrder, even if not shown; the are counted as the last sibling(s) child(ren) of the note on whose Map view they appear but they do not affect the normal hierarchy or the links of any notes. However, adornments do not affect $SiblingOrder, and are not included in $ChildCount and $DescendantCount. Containers have a separate $AdornmentCount for the number of adornments on their child map.

Adornments & searches

Adornments are **always** excluded from agent query matches, i.e. they are not treated as notes when matching, as well as by group designators including find(), plus the view pane's Find toolbar. Altering $Searchable for an adornment will **no** make it appear in searches.

Queries can test the inside() operator using an adornment's name to see if any notes lie *on top of* that adornment.

The boolean $IsAdornment is `true` for adornments.

Use of $Text

Adornments have a Text pane, and can have $Text. However, unlike normal notes, $Text—even when present—is never displayed in the map icon. Altering $MapBodyTextSize has no effect on this..

Adornments and export

Adornments cannot be exported. As they are excluded from child/descendant counts, normal export tests for the presence of children in a container, etc., will work properly even if there are adornments present.

Styling adornments

Adornments set to zero width or height can be used to make 'divider' lines on maps (see more).

If $Color is set to 'transparent' the adornment's icon is hidden and only the title shows on the map, although the adornment still works as usual for things like $OnAdd actions. Adornments fully respect $Opacity enabling them to be fully translucent. It is possible to have a visible border for a transparent adornment, but to do this $BorderColor must be set to a value other than 'automatic': styled borders (e.g. dashed) can be used with adornments.

Adornments can have drop shadows like note icons. The $Shadow will always default to false for an adornment regardless of the app/doc setting for notes, i.e. such defaults only apply to the latter. For that reason, making a prototype adornment with a shadow and then inheriting that is an easy way to use drop-shadowed adornments. Adornments show a small inner shadow (inside the the border line).

Adornment actions

Use of an $OnAdd/$OnRemove action

When a note is created on top of an adornment, the adornment's $OnAdd action is run; prior to this, the adornment action ran only when the note was moved onto the adornment. The adornment's $OnAdd is also run on other adornments placed in or overlapping the adornment if the former is in front; if behind no $OnAdd fires.

$OnAdd also affects adornments created or pasted into containers and the adornment's $OnRemove action is performed if a note atop it is removed or deleted.

Adornment grids

Adornment grids allow an optional gridded framework to be drawn into an adornment as background markup to any notes being arranged atop the adornment. The grid is most easily configured by clicking the 2x3 grid icon which opens the Grid Properties pop-over.

The number of cells in the gird are controlled via $GridRows and $GridColumns. The cells can each be given a label via $GridLabels, which are drawn in $GridLabelFont at $GridLabelSize. The overall grid and labels are drawn in $GridColor and with $GridOpacity.

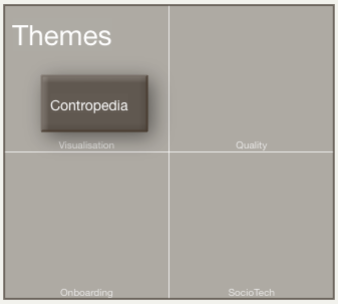
All these adjustments can be accessed via the Grid Properties pop-over which is accessed by the grid icon on any currently selected adornment.

The image shows a 2 row, 2 column grid with labelled cells; all other settings are default. Transparent adornments may draw a border and grids.

The use of a grid does not affect the OnAdd action of the adornment, which applies to a note regardless of where on the grid it is placed.

The placement of notes by a smart adornment is not aware of a grid. The smart process will use the adornment as a whole, ignoring grid assignments. Therefore grids are not ideally used with smart adornments.

The grid icon is not displayed when the adornment is part of a multiple selection.

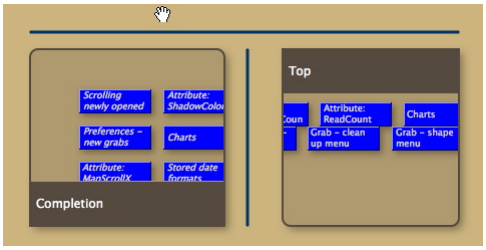


Adornments as map dividers

It is possible to set zero width or height for an adornment allowing it to server as a rule or linear divider on a map. Note that such dividers can only be of horizontal or vertical alignment, i.e. not at just any angle. It is possible to set both height and width to make a small dot.

The colour of the resulting line is the adornment $BorderColor. If that is left as the default 'automatic' value, then setting $Color for the adornment will give the intended value and may be a more intuitive way of setting it.

The height and or width of a divider is set via its $Border attribute (default =2). Thus, a zero-width adornment will draw as unit wide vertical line of height $Height with a width as if there were a persistent border of $Border value 1. Conversely, setting a wide border value, e.g. 10, causes a 10 wide block of colour. In contrast, a default adornment set 10 units wide would show 6 units width of inner colour and two 2 unit wide borders. Thus a wide border on a zero-width/depth adornment sets a band of solid single colour. If the $Border is set to 0, the adornment renders with an overall width and/or height of 1 pixel in $Border colour.



Geographic Adornments

NOTE: *Due to performance related issues, this feature—Geographic Adornments—has been retired from v9.0.0 until these issues can be resolved. It still works in v8 in v8.0.4+*

Version 9.6.0 restores the functionality of Geographic adornments, which were in abeyance v9.0.0. through v9.5.2

A map adornment becomes a geographic adornment if all of the following evaluate as `true`:

- $Latitude is non-zero
- $Longitude is non-zero
- $Range is sufficiently large

The latitude and longitude may be set directly, or may be computed indirectly because $Address has been set. The scale of the map is determined by the adornment's $Range attribute, which represents the approximate size of the map in kilometres The scale of a geographic adornment's map fill is *fixed*, and in relationship to the adornment's $Height & $Width.

Thus, dragging the adornment bigger will show more detail—i.e. appear to zoom in—as there is now more space within the adornment to plot the same $Range. The reverse applies for making the adornment size smaller. For a given adornment size, increasing $Range zooms the map out, decreasing it zooms in.

If a note that sits atop a geographic adornment has a latitude and longitude that lies within the adornment, it is moved so the note's centre lies on the corresponding place in the map. If the position is not on the map, the note is moved to the edge of the adornment. If a geographic adornment moves a note to a position that is already occupied, it will try to find a nearby place that is not occupied.

Geographic adornments can only draw maps up to about 8,000 pixels in size, which should be ample for most purposes.

If an adornment has a non-zero $Latitude or $Longitude (which it typically gains automatically if the adornment has an $Address), it becomes a geographic adornment and displays a map of the area surrounding that place. The extent of the map displayed is determined roughly by the attribute $Range, in kilometres; if not specified, the $Range is approximately 1km.

Data source for maps

Currently geographical adornments use Apple MapKit, via Tinderbox, to source the map's information.

Image Adornments

Pictures may be inserted into the background of a map view as image adornments. This is done in one of two ways:

- Copy and paste from a bitmap editor (including Preview). Open the image in the editor. Select the whole image and copy. Switch focus to a document tab using view and paste to the view pane.
- Drag-and-drop from Finder. If dragging from the Desktop, ensure the image is selected before starting the drag otherwise the image may not be imported. Drag and drop onto the map view.

Like other adornments, image adornments do not show up in any other views, and do not affect the hierarchy or the links of any notes. Locked picture adornments are ignored from drag-selections of notes.

The image format must be a bitmap image (PNG/JPG/JPEG/GIF/BMP) and *not* vector-based (i.e. not CAD, AI or EPS). Regardless of source format, image adornments store image data as JPEGs, in Base 64 form.

Just as normal adornments are not exported in HTML Export, neither are picture adornments.

Picture adornments respect $Opacity and may have non rectangular shapes, just like normal adornments.

Image size. Images are inserted and displayed at 'actual size' (i.e. for current desktop resolution). Resizing the picture's 'frame' vertically will scale the image. However, dragging the frame horizontally has no effect. But, if the frame dragged narrower (horizontally) than the image it will visibly crop it, working in from the right margin (i.e. the left side is always retained. The picture adornment does scale if the whole map is zoomed in or out.

Resizing picture adornment in aspect ratio . There is no mechanism for this, but if making regular use of picture adornments, stamps can help along with a couple of user attributes. This workaround works on storing the original pixel width and height in Number type attributes. Suggested names are `$ImgWidth` and `$ImgHeight`.The original pixel count can be obtained by opening the image in Preview and using the Inspector window to check width and height. Add 3 stamps:

Name: "Resize by Width". *Code*:`$Height = $Width*($ImgHeight/$ImgWidth);`

Name: "Resize by Height". *Code*:`$Width = $Height*($ImgWidth/$ImgHeight);`

Name: "Resize as original". *Code*:`$Width=$ImgWidth/35.3;$Height= $ImgHeight/35.3;`

The first two stamps work off the ratio of the original pixel height width. The first stamp keeps current $Width an adjusts $Height accordingly. The second stamp does the reverse, maintaining $Height and adjusting $Width.

The third stamp is an approximation as Tinderbox's mapping of pixels to map units varies slightly in scale. Ostensibly 32 pixels per unit, at normal map zoom scale a factor of 35.3 seems to work best; note that this may not work as well at different zoom scales, so change to normal zoom to use this stamp.

Locked and Sticky Adornments

Locked adornments

Locked adornments are locked ($Lock) onto the map background to avoid accidental movement. The lock state can be toggled on/off using the icons top right of the adornment; if desired the icon can be substituted with a custom icon. Locked adornments are ignored from drag-selections of notes. An adornment can be selected whilst locked but to re-size it (or drag it) is must be unlocked.

Locks a note or (more usually) an adornment to the Map to avoid being moved inadvertently.

Primarily used in Map view (indeed, intended primarily for adornments), if $Lock is `true` the adornment can be selected but not repositioned, i.e. $Xpos and$Ypos are locked. Nor can the item be re-sized (i.e. changing $Height or $Width).

Sticky adornments

Sticky adornments ($Sticky) ensure any notes within or overlapping them move with the adornment when the latter is moved. This makes it much easier to work with moving sections of very large, complex maps.

Overlapping adornments are also moved if they lie above the sticky adornment in outline order, i.e. adornments that lie partially over (above) the sticky adornment but under (behind) it. The sticky state can be toggled on/off using the icons top right of the adornment; if desired the icon can be substituted with a custom icon.

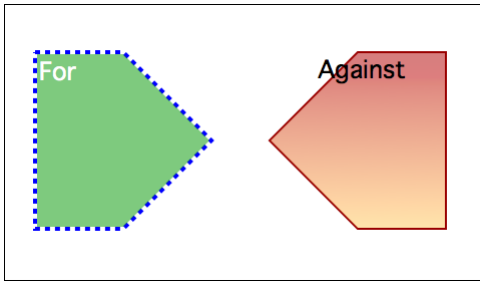
Note that, while moving a sticky adornment moves notes on top of it, items on a 'sticky' adornment can still be moved independently, either on the adornment or to move them off it. In the latter case the object is no longer 'stuck' to the adornment. Moving an item off a sticky adornment has no effect on the adornment itself.

Custom icons for locked/sticky state

The icon (ICNS-type) files for the above can be customised.

Shaped and Patterned Adornments

Adornments can use shapes. Note that due to the shapes it may be necessary to use a $NameColor setting other than 'automatic'. Adornments may also use patterns and badges. Shaped adornments are allowed, with the exception of 'cloud' and 'bubble' can be transparent and have borders; 'cloud' and 'bubble' have no border if made transparent.



Smart Adornments

Any adornment is made into a smart adornment simply by giving it a query (stored in $AgentQuery) using agent query syntax, i.e. action code. Importantly, the scope of the query in a smart adornment is always the current map, the map on which the adornment lies. All other notes are automatically out of scope. The following facts apply:

- Notes within the current map matching the query are moved onto the adornment. Unlike an agent, the smart adornment *does not create an alias* of the matched note but rather *moves the original note itself* onto the adornment, which is why the scope of the query is limited to the current map. In this context, 'original note' could also be a container, agent or an alias.
- Notes moved onto the adornment as a result of the query fire the adornments's $OnAdd code.
- Adornments do not auto-expand to encompass notes. In some cases where there are so many matching notes that they fill the adornment, some notes may be placed around the adornment.
- Matched notes are sorted as per the adornment's sort attributes.
- Matching obeys the smart adornment's $AgentQuery.
- Smart adornment ignore $AgentPriority setting. In this they are unlike normal agents. In the Query Inspector, for a smart adornment, the Priority pop-up is disabled as a result.
- Matched notes have their $Xpos/$Ypos altered as the original is being moved.
- The outline order of matched notes *is not affected*.
- Matched notes are laid out in a grid (sort) order on the adornment until the available areas is filled. Thereafter notes may be stacked on top of one another or alongside the adornment until the latter is manually resized to accommodate all matched notes.
- $CleanupAction is ignored. When a query is set the user *cannot* re-arrange items on the adornment. In the Query Inspector, for a smart adornment, the Cleanup pop-up is disabled as a result.
- Notes no longer matching the query are moved off the adornment and placed alongside it. They are not restored to their location when first moved by the adornment.
- Notes/containers/agents that are locked are ignored.
- Other attributes are ignored. Notes 'in' other adornment will be moved (even if the original adornment's sticky flag is set).
- To read or set adornment attributes via $OnAdd actions, use the 'adornment' designator (it operates analogously to the 'agent' designator used in agents).
- A note on the map can only move onto *one* smart adornment. In the case of conflict, where two or more smart adornments match a given note, the adornment with the *highest* $OutlineOrder will match the note. An adornment's $OutlineOrder can be looked up via its Get Info pop-over (attributes tab: General section). Normally, a duplication conflict resolves to the lowest outline order but in adornments the z-order and thus outline ordering is reversed (further explanation).

If a smart adornment is made larger (manually or via an action) and in doing so overlaps non-matching notes then the latter are displaced to one side so as to remain outside the adornment. Re-sizing a normal adornment the notes would simply lie inside the new adornment boundary.

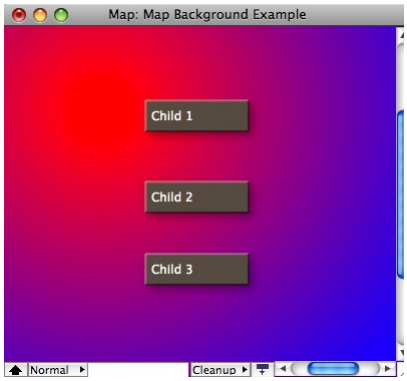
Smart adornments cannot be used to move or act on notes outside the current map. So, for instance, it *can not move* a note into a (container's) subordinate map or promote it to a parent map, or indeed pull it from another map. Put another way, it can only act on its siblings.

Smart adornments allow space for multi-line subtitles ($Subtitle), assuming that the adornment is drawn at or near standard magnification.

Background patterns

A Map view's background can use the same range of basic patterns as are available to map icons. The feature is set via a series of preferences on the Maps tab of Document Settings and settings are stored in $MapBackgroundColor, $MapBackgroundAccentColor, $MapBackgroundPattern.

The same settings can easily be customised for a given map via the Map Properties pop-over



Containers

Notes (or agents) that have child objects are termed containers.

- Aliases of Containers
- Container plots
- Container summary display tables
- Container title height
- Container viewport is scrollable
- Viewport detail in containers & agents

Aliases of Containers

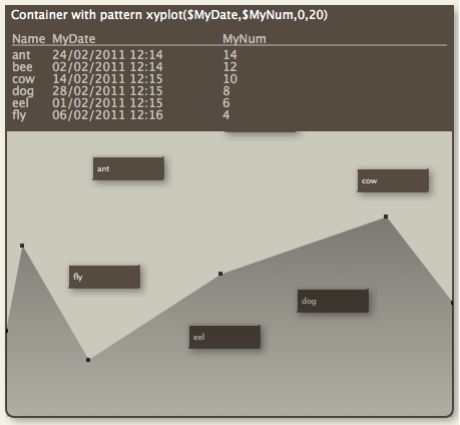
It is possible to zoom into the sub map of an alias of a container. However note that:

- This is equivalent to zooming into the child map of the original.
- Unlike the original container, the original's child notes are not drawn in the viewport of the alias of the container.
- The alias container shows any table expression or container plot used by the original, which can be useful for dashboards.
- Aliases of agent containers are drawn the same way as normal containers, i.e. with the title bar at the top. It is thus not possible to distinguish from icon layout alone whether the original is an agent or a normal container.

It is possible to zoom into an alias of a container by double-clicking its body or by selecting the alias container and pressing the down-arrow key (↓). The map then displayed is *always* the map of the *original* container. Using the up-arrow ke

(↑) or the breadcrumb bar to navigate upwards will always result in the original container being selected. Note also, that if the alias originally drilled into was on a different map, the map shown will now be a different one.

Container plots



A pattern that applies only to containers (notes & agents), is plot(). It evaluates an expression using data from each child of the containers to produce a graph; i.e. the scope is always a group, specifically 'child'.

The graph, in the forms of a sparkline-type plot is drawn across the viewport area of the container. The graph is drawn in colour $PlotColor. The container viewport is still accessible for drag/drop, etc., as if the plot were not there: think of the plot as an overlay.

For example, to graph the word count of each child note in the container, see the container's map Pattern attribute to use plot():

```
plot($WordCount)
```

An alternate plot type is bargraph(), which draws a bar graph of each child item's value.

```
bargraph($WordCount)
```

A further option is xyplot(), which offers a double variable plot:

```
xyplot($Date,$Price)
```

The pie() plot gives a pie graph based on an attribute (or expression):

```
pie($EditsMade)
```

The ring() plot displays an arc representing 70% of a complete circle. The circle is always drawn centred on a 9-o'clock position; a 50% completion would fill from 6 through 9 to 12.

```
ring(70)
```

All plot operators accept optional minimum & maximum values, e.g. plot($Attribute,min_num,max_num). Thus:

```
bargraph($WordCount,0)
```

graphs the word count of each note whilst ensuring the Y-axis is based at zero, with all attribute values including the maximum being plotted. Note that while zero is the default value of an 'blank' number type attribute, the type allows minus values. The above example would treat all of them as if their value were zero. If a negative min value is supplied, negative item values above that threshold are plotted. The above example would treat all of them as if their value were zero. For:

```
plot($WordCount,10,900)
```

the plot graphs the data from a baseline of 10 to a maximum value of 900. Values outside these are plotted appropriately as the min or max values.

Container plot-type patterns are ignored if applied to non-container notes.

A container can also apply the bar() & vbar() progress bar patterns as used with note icons; these are applied to the containers title bar area. However, as both types of visualisation use the $Pattern attribute it is not possible to use a progres bar and plot at the same time.

Where a container plot is applied, Before the graph is drawn, the space behind the plot is filled with the colour $PlotBackgroundColor with opacity of $PlotBackgroundOpacity. Previously the background was that of the child map.

Container summary display tables

Visualisation features of major views

Type	Charts	Maps	Outlines	Timelines	Treemaps
Color1	DisplayName, border	Icon face, adornments	DisplayName	DisplayName	DisplayName, border
Color2		In patterns	Swatch patterns	Duration bar	
Long Title?	Stretch / Wrap	Expand Hor./Ver.	Wrap	Wrap	
Subtitles		Yes	As HExp, if no HExp		
Has any $Text?	Dogear	Dogear	Icon fill		Dogear
Display Expression	DisplayName	DisplayName	DisplayName	DisplayName	DisplayName
Hover Expression		Yes	Yes		
Separators			Yes		
Adornments		Yes		Yes (if dates)	
Badge	Yes	Yes	Yes	Yes	
Shape		Icon shape			
Pattern		Icon face	In swatch		
Pattern – Bar		Icon face	In swatch		
Pattern – Plot		Container viewports only			
Fill		Icon face, adornments			
Opacity		Yes		DisplayName only	
Shadows		All icons/shapes			
Age since edit	Dogear colour	Dogear colour	Dogear colour		Dogear colour
Table Expression		Containers only, title bar			
Show links?		Yes		Yes	
Is a prototype?			Disc behind icon		
Uses a prototype?		Tab, when icon selected	Icon context menu		
Is an agent?		Viewport location	Icon type		
Map background colour	Yes	Yes	Yes	Yes	Yes
Map background pattern		Yes			
Outline swatch			Solid/pattern/bar		
Outline background colour			Yes		
Timeline Band				Yes	

In map view, containers and agents (and their aliases), but not ordinary notes, can display tabular summaries of their contents. Tables are drawn only if there is adequate space in the title bar area. The number of rows displayed in the table is limited by the available space; increasing the size of the note (and its $TitleHeight) will increase the size of the table. Note that the container's sort order is reflected in the table. More on a container's title height.

The $TableHeading and $TableExpression, as described below, can be edited via the Summary Table Properties pop-over dialog.

If no $TableExpression is specified but space is available in the title bar, Tinderbox displays an excerpt of the $Text of the container or agent, as it does for notes. If a TableExpression is defined, display of $Text is suppressed, regardless of the height of $TitleHeight. In other words, it is possible to display a table or $Text but not both at the same time.

When a table is displayed, the information in the table is determined by the expression attribute $TableExpression. For example, if TableExpression is

```
$Name+"|"+$WordCount
```

Tinderbox will draw a two-column table containing the $Name and $WordCount of the first few children of the container. The '|' character (often called a vertical bar or 'pipe') separates the columns of the table.

Optional column headings may be specified in the string attribute $TableHeading. Again, columns should be separated by a '|' but note that if referring to attribute names in $TableHeading, a $ prefix is not used as the name is used as a litera string. So the heading may be a any of text labels, attribute named text labels or a mix of the two. The '|' characters to delimit columns is simply inserted inline in the name string, no space padding is required. Any of the following could be used for the above table expression example:

```
Note title|Word count
Name|WordCount
Name|word count
```

Note the lack of $ prefixes in the latter two examples where attribute names are cited and lack of padding around the '|' column title delimiter.

The left column (table and heading) is always left-aligned with all other columns being right-aligned.

For agents, the table is shown beneath the title whilst the viewport remains above.

The table uses various other attributes for its styling:

- Text font: the table is drawn in $NameFont
- Colour: the table default to use $NameColor but will use $MapBodyTextColor if set differently from $NameColor.
- Text size: uses $MapBodyTextSize. Note setting the value to 1 will suppress display of table & text data.

N.B. the table heading and content have the same font/colour/size and can not differ from each other.

Container title height

Containers and agents have adjustable height title bars.

The height of the title bar may be changed by dragging the bottom of the (selected) container's title bar, or the top of the agent's title bar, to make the title area taller or shorter. Within the exposed 'title' area, subtitles, body text or table data is also drawn if there is room, as on a normal note icon.

The height of the title bar may also be changed by setting the $TitleHeight attribute. The attribute uses Tinderbox map units for its value.

To 'hide' the viewport area of a container, set the $TitleHeight to equal $Height (or for agents, `$Height-0.1`). This makes the map icon look like a normal not except the 2 bottom (note) or top (agent) corners are rounded. While the viewport hidden, it is not possible to double-click drill down but its shortcut (Cmd+Opt+Return) will work.

Container viewport is scrollable

In map views, the view inside a container (the 'viewport') reflects the container's current scroll position. So, dragging the background of the interior of a container scrolls the inside of that container and also updates the opening co-ordinates o the child map opened via that viewport ($MapScrollX and $MapScrollY). Adornments seen on a viewport map cannot be dragged.

Conversely, scrolling in a map view of the children will result in scrolling of the parent container's viewport contents.

Note that dragging a note in a container viewport outside the viewport drags that note out of the container (i.e. promotes it from child to sibling). To drag the container icon on the map, drag its title bar. If an interior note is locked, clicking on it will not begin a drag.

Viewport detail in containers & agents

A number of attributes make it easier to show detail within containers, be they notes or agents:

- $InteriorScale, which allows setting of a scaling factor used when drawing the viewport contents. This allows more note detail to be shown or conversely a larger amount of the child map contents. Changing $InteriorScale forces a re-draw of the viewport.
- $MapScrollX and $MapScrollY. These control the X/Y co-ordinates of the child map that will be displayed at the horizontal/vertical centre of the container icon. Whether that actual point is visible in the viewport will depend on the $TitleHeight of the container.

The order/location of aliases seen in an agent can be moved and seen in more detail in an agent's viewport.

By clicking and dragging the map *background* within the viewport, the child map can be 'scrolled' within the viewport. Such a change updates the $MapScrollX/$MapScrollY for the container in question. Adornments cannot be drag-moved on a viewport map, drill down to do this.

Details maintained in container viewport display: Shapes, Progress bar mark-up (bar/vbar), map background colour, adornments, icon shadows. In fact it shows pretty much everything except:

- the display of body text.
- table expression data.
- viewport data within containers, though containers on the viewport map that have container plots will still show these.

Notes being drawn in the interior of agents and containers respect both their own opacity *and* the opacity of their container. Thus, if a container is translucent, the notes inside it are also drawn translucently.

Dashboard concept

[The image here is courtesy Mark Bernstein, reproduced with permission.]

In TBX 'dashboards' leverage the map view, though there is no built-in dashboard feature as such. To make a dashboard simply plan an additional map view in your TBX whose layout is set up to illustrate key data about your project. The layout, styling and choice of data to display will vary by the needs of the TBX's subject matter.

Some attributes needed for dashboard-style display are shared with source notes, others are not, involving slightly more careful planning. Remember too, that alias titles are always italicised and that cannot be overridden.

A dashboard can use any combination of notes, aliases and containers/agents to shows its data. By making a container off the TBX root to hold your dashboard, you can more easily do things like change the $MapBackgroundColor without concerns about this colour being used in your main content where it may be less appropriate. If using agents in your dashboard do give thought to their scope and $AgentPriorty. Be prepared to adjust the latter if a first attempt results in significant increase in agent update cycle time.

Force Directed Layout: 'Dance'

Note: this feature is experimental and the result is not undoable.

In map view, View ▸ Arrange ▸ Dance (⇧⌘-D) initiates an automated layout of the view based on a physical simulation. Each link among notes in the map is treated as a spring that pulls linked notes together. The effects of Dance are undo-able (but if trying this it is best to do so immediately after the Dance, so its top of the undo stack).

The behaviour of Dance is controlled by a pop-up shown when then the feature is invoked

All notes exert a gravitation attraction for other notes. Notes that overlap repel each other. Notes that are textually similar to each other attract each other. At the beginning of the simulation, each note is subject to a random force, much as if it were heated. The force is reduced progressively over time. This process, known as simulated annealing, helps the simulation from getting tangled up in local minima.

Each adornment that has one or more values of $ClusterTerms attracts notes in which those terms appear in the text. Adornments are otherwise ignored in the simulation.

Not all maps will benefit from automatic layout; for example, the famous (Storyspace) map of Mary-Kim Arnold's "Lust" does not due to its many inter-connections. Performance may be unsatisfactory in maps with more than a few dozen notes.

Dancing automatically stops when any note is dragged. Otherwise, the effect of the drag can impart a vast, unwanted acceleration to the dragged item.

If more than one note is selected, only the selected notes will be moved by the Dance command; this is very convenient when arranging a cluster of related notes in a large map.

If no note is selected, or only a single note it selected, then all unlocked notes are moved by the Dance command.

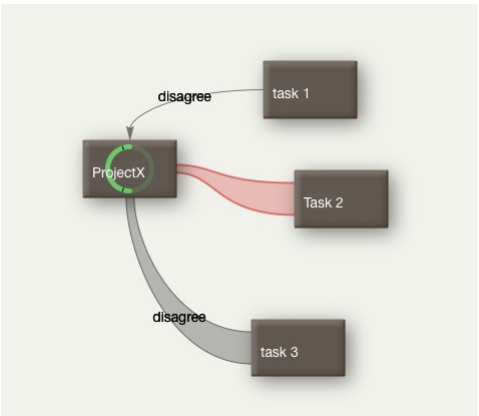
Links

- Broad links
- Draggable link anchors
- Link Curve Control
- Link controls
- Link labels
- Link stubs
- Map link highlighting
- Note creation via link dragging

Broad links

A 'broad' style for curved (i.e. bezier, non-linear) map links, set via the Document Inspector's Links tab. The link tapers in width from wider at the source to narrower at the destination. Dash/dot settings are applied as a border in broad mode, combining styles as for a normal narrow link. The border is drawn in the link's colour and the link's fill is in a lighter tint of that colour.

The broad setting applies to all links of a given type, although it can be over-ridden as a per-link style as with other settings.



Draggable link anchors

The source or destination ends of map view links can be dragged to a different face of a note/container icon; stub links, inbound or outbound cannot be altered. Dragging the link locks it to a new specific face of the note. The end of the link being dragged affects whether the source or destination attachment is altered: each can be altered independently. Both curved and linear type links can have their attachment points dragged.

Source end of link. With links(s) selected via the *source* note, click the cursor onto the link close to the *source* end of the link. Four arrows will appear over the destination note (illustration is for the case below). The link's end can now be dragged to any of the other three destination positions.

Destination end of link. With links(s) selected via the *source* note, click the cursor onto the link close to the *destination* end of the link. Four arrows will appear over the destination note (as illustrated). The link's end can now be dragged to ar of the other three destination positions.

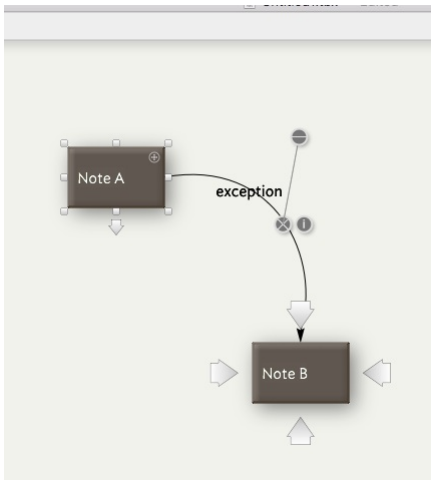
When a link attachment drag is active, the note icon whose attachment is affected will show a large arrow over each face (top/right/bottom/left). Dragging onto one of these arrows re-sets the links attachment at that face. To cancel the drag, or to re-set automatic attachment selection, drag the note onto the map background.

Once set, a user-defined start and/or end link position is stored as part of the link's data with in the document's XML data.

The default for selecting where a link is attached is 'automatic.' Tinderbox will attempt to find the optimum face of source/destination to draw a clear line. If the note is moved, Tinderbox my (automatically) shift link attachment points to maintain an optimal layout. The drag method allows the user to override this behaviour. Once a link has been manually set to use a particular face, that attachment is maintained even if the note is moved, unless automatic attachment is re-

enabled.

The attachment position of links can be seen via the Browse Links pop-up of the originating note, using the 'Start' and 'End' pop-ups. These pop-ups on this dialog can be used to alter, or re-set, the attachment for the link selected from the dialog's left-side link listing. The pop-ups default to 'automatic' but can be manually set to any of top/right/bottom/left. Changes made on the dialog update the map dynamically.

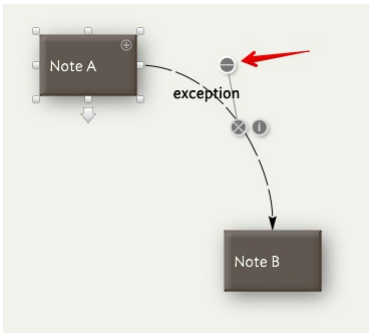


Link Curve Control

A link widget allows adjustment of the curvature for curved-style links (the default link style).

Drag the widget (indicated) to adjust the link's curvature. Movement allows both for altering the centre of arc of the curve (the 'x' point) or adjusting the bezier of S-shaped curved links.

Link attachment at the destination end is controlled separately via draggable link anchors.

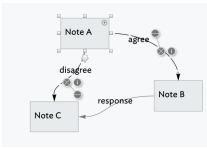


Link controls

When a note is selected in map view, all outbound (non- stub) basic and text links are animated and each such link also shows three circular controls:

- horizontal line in circle. A draggable control to alter the bezier curve of the link.
- letter 'i' in circle. Click to open the Browse Links pop-over for the current note.
- crossed lined in circle. Click to delete that link. Undo is supported, e.g. if the control clicked by mistake.

If the link has a link type label, that is also draggable.



Link labels

The Map view has the ability to visualise links between notes. These are drawn as lines between linked notes, with an arrowhead at the terminating end of the link (i.e. the destination). Where the link starts outside the map, the link is shown as a short vertical line into the top centre of the note's map icon; where the link ends outside the map a short link is drawn vertically downwards from the note's icon.

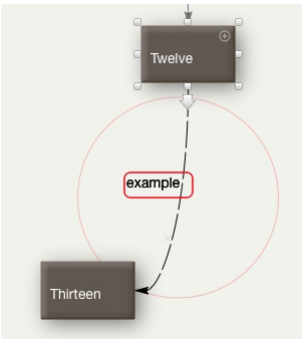
A Map Preference allow a choice of lines being drawn as straight or curved lines; there is also an option to draw/not draw all link lines.

Via the Attributes palette's Link Types pane, it is possible to control, at per linkType scope:

- The name (label) of a link type.
- Whether that link type is visible. This is trumped by the show/no show Map Preference.
- Whether, when shown, the link displays its link title. The link title is drawn in the same colour as the link arrow (previously the labels were always black, regardless of arrow colour).

The the drawing of Map link lines is altered so as to avoid link labels overwriting one another where multiple links exist between notes.

Link labels can be dragged a small distance from their associated link to aid legibility. The new position is saved through navigation up/down the outline or zooming the map. However, if either source or destination note is moved the label may move and require manual reposition; this makes sense as the line is moving as after initial repositioning the label is no longer in an auto-positioned place. When a label is dragged a circle is shown, indicating the current limit of when the label may be re-positioned.



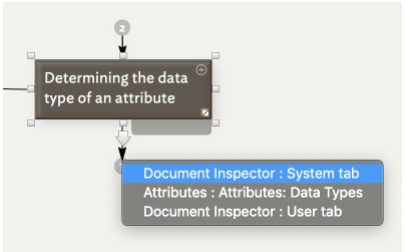
Link stubs

In the map view, a link stub represents a link to or from a note that does not appear in the map because that note is inside a different container. The stub for inbound links is drawn above the note's map icon, stub outbound links are similarly indicated below the note icon.

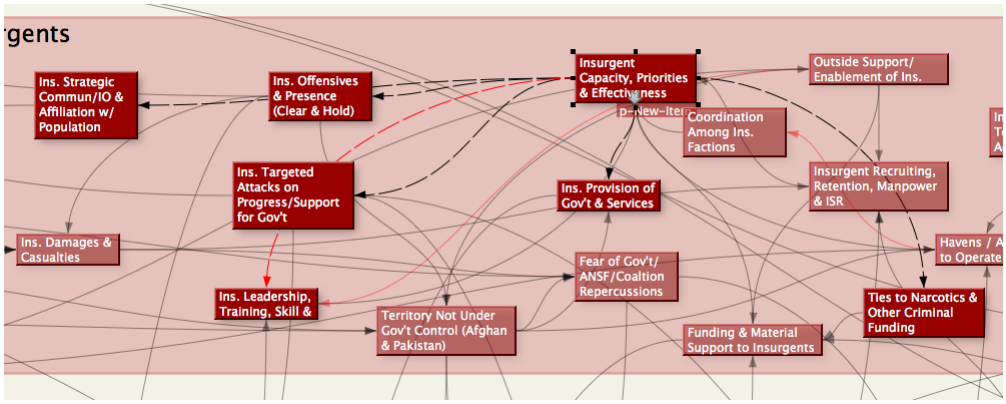
In each case, Tinderbox displays the number of stub links in a circle at the outer end of the relevant stub. Clicking this link count circle reveals a pop-up menu of destinations; clicking an item in the pop-up menu follows that link and loads that note in the text pane. This act *does not* move the map view focus to the map containing the selected note.

The attachment points of link stubs are not draggable.

The pop-up menus of the inbound-stub and the outbound-stub show the *display names* of linked note, prefixed by the display names of their containers. The extra information may help disambiguate situations where it is convenient to have several notes with identical names, or when linking to different aliases of a note.



Map link highlighting



If no notes are selected, or if there are no links in the map, all notes are drawn normally.

Highlighted notes are shown in an animated dashed style, with the gaps cycling along the link from source to destination. By Cmd+clicking additional items on the map the number of animated links can be increased, e.g. indicating a path through a set of linked notes. It is not intended as general eye candy: trying to animate every link on a large map may start up the computers fans and/or leave the map unresponsive.

In complex maps, link animation can consume significant computational resources. Link animation is slowed in large maps to manage load.

Note creation via link dragging

When dragging links in map view, releasing the mouse when not pointing to a note will create a new note at that location, opening the Create Link pop-over.

If you change your mind and do not name the new note (or close the pop-up using ⎋), the newly-created note and its link are automatically deleted.

Map Composites

Only seen in map view, composites offer a means of treating a group of notes as a consolidated collection.

Map Coordinates

Any object on a map will have a position defined by the location of the top left corner of the item, stored as $Xpos and $Ypos; the icon is then drawn to $Width and $Height from that location.

Tinderbox map co-ordinates differ slightly from normal X/Y graphing. The X axis is the same: positive to the right of zero, negative values to the left. However, the Y axis is flipped with values up the page from zero being negative and downwards being positive. Why the change?

It makes sense if you take into account the fact that Tinderbox starts a new map by setting the top left corner of the map window to {0,0} adding noted into the bottom right quadrant of the map, populate it right-wards and downwards. In Tinderbox's co-ordinates this quadrant uses positive X and Y values rather than positive X and negative Y in a more conventional graphing method.

Thus the first items on any map are using positive values for $Xpos and $Ypos. The accompanying image is thus a bit misleading as Tinderbox map will, without any manual user intervention, fill the bottom right quadrant of the map. A slight exception to this is that if a note is dropped onto a (non-container) note this first child item is placed at {-2,-0.5} on the new child map. However subsequent content is added down and right, i.e. in the bottom right quadrant of the map.

In general use, Tinderbox's 'reverse' polarity of the Y axis will be hidden from the user. The facts are only really important if the document is being scripted and items need to be precisely positioned. This co-ordinate system also allows for potentially infinite maps. Tinderbox can 'add' more map in any direction as needs be without having to reset the frame of reference.

Map co-ordinates are also referred to as map units, which are used by a number of attributes to store portion or size related data.

At normal zoom level 1 map unit equates to approximately 32 pixels (this may vary on more modern systems).

Map grid & guides

When moving objects on a map, their boundaries 'snap' to the nearest 0.5 unit of map measurement. In addition alignment guides are shown to enable grid-like placement if so desired. The guides can be turned off if they are distracting, via View menu, Guides.

Map view guides have been simplified and improved. Additional guides help to:

- align the vertical and horizontal centres of notes
- maintain uniform spacing between adjacent notes or place notes next to each other
- align the edges of notes
- maintain spacing between notes and the edges of the adornment that contains them
- place a note symmetrically between two horizontally-adjacent notes
- create square notes and notes using the golden ratio. These aspect ratio guides show as additional on-screen guide label when a relevant ratio is achieved.
- align left or right edges of notes to the midpoint of notes above or below them.

Guides look for equal spacing between adjacent sets of notes. If there are two adjacent notes, A and B, and C is then moved next to B, the guide will make the space between B and C the same as the space between A and B.

Guides examine the neighbourhood around notes with a circular shape, looking for other notes that are nearly the same distance from the circular note as the note being dragged.

The guide for centring a note between two other notes formerly considered only the notes closest to the dragged note. It looks for matches among all pairs in the vicinity of the dragged note.

If the shift key (⇧) is depressed while dragging this also disables guides, which are drawn in grey rather than the usual blue-green to show that they are inactive.

Whilst dragging a composite various guides including the aspect-ratio guide, that affect properties internal to the composite, are not displayed.

Map View endeavours to be smart about which guides it displays. For example, no amount of dragging an object can change its aspect ratio, so aspect ratio guides appear only when resizing an object.

A new map guide notices when a note or adornment is approximate the same size as one of its neighbours, and snaps the note size to match the neighbour. If no nearby note is approximately the same size, the guide looks for notes with approximately the same height or width.

A new map guide looks for 45° diagonal alignments between nearby notes.

The edge alignment guides examine notes within a radius based on the size of the dragged note, rather than based on a fixed radius. Diagonal alignment guides have a smaller range.

Grids and close-vs.-far objects

Guides can only assume so much. Sometimes it is desirable to align two items whose relationship is obvious to the eye but less so to the guide logic. For example, aligning the left edges of two important notes that are separated by many other notes. By comparison, guide logic is invoked on the relationship of the closest objects. If guides do not show the desired alignment prompt, use the explicit commands on the Arrange menu.

Automatic gridded arrangements

These grids are for manual positioning use. To alter the default layout behaviour of maps inside agents (i.e. their aliases) see re-arrangeable agent maps.

Map overview mode

To see an overview of the entire map, press and hold the Control, Option, and Command keys (^+⌘+⌥) simultaneously. The overview has been improved to ensure the entire map is visible.

To zoom to a different part of the map from the overview, move the mouse cursor to the area of the map in which you are interested before you release the ^+⌘+⌥ keys.

Map object icons

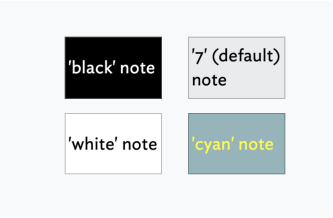
- Auto-coloured note titles
- Badges on icons
- Captions
- Content Dogears
- Icon dashed borders
- Icon drop-shadows
- Icon fill textures
- Note Flags
- Opacity & Transparency
- Note progress bars
- Note, Container and Agent icon layout
- Notes accommodate titles
- Notes displaying body text
- Note icon size and prototypes
- Notes with images in text
- Shaped Map notes
- Subtitles

Auto-coloured note titles

The titles of map view note icons are rendered in $NameColor, whose default value is " automatic". This means that in *most* cases the note's title is rendered in black (#000) or white (#fff) according to which offers the best contrast. In some cases, where neither black nor white may offer good contrast, titles use a yellow colour.

These automatic colours cannot be redefined, though $NameColor can be set to user-defined colour though this does mean the title colour will not change if $Color changes in a manner lessening contrast.

Automatic colouring was defined before more recent inclusion of new colour schemes, i.e. the 3 default 'automatic' choices were not optimised for all colour schemes.

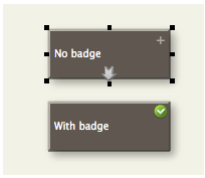


Badges on icons

Notes may be decorated with a ' badge', a small icon that appears to the left of the note name in outline view and in the upper right hand corner of the note in map view (or lower right corner for containers/agents).

Badges are displayed in a slightly different location according to the type of icon (as illustrated):

- Note: top right of icon
- Note container: bottom right of icon
- Agent: top right of icon



The reason for this is that in the case of the containers, the part of the icon holding the title is comparatively small and a badge could intrude on that.

Badges are not shown on icons displayed in the viewport of note and agent containers.

Badges may be selected from the badge picker pop-over by clicking on the area where a badge is displayed (see list above).

The note's badge is controlled by a string attribute $Badge, which holds the name of the badge. The value of $Badge may be set through inheritance, rules, or actions as well as well as manually via the pop-up list. Additional attributes contro monochrome display ($BadgeMonochrome) and the exact size ($BadgeSize).

When a note is selected in map view, a subtle "+" is drawn where the badge would appear if the note had a badge. This helps users discover the location of the contextual badge menu.

Large badges with $BadgeSize > 32 respect the shape of the note and are clipped to its border. Small badges, as before, may extend outside the drawn region of the note.

Captions

A new text field, $Caption, can appear beneath a note or an adornment, in map view only. Captions are intended to be additional annotations or comments about a note.

Caption appearance is controlled by $CaptionFont, $CaptionOpacity, $CaptionSize, and $CaptionColor.

Captions be edited by selecting the note and then clicking on the caption. Captions may also be edited in the Caption pane of the Name inspector, or by setting $Caption via action code.

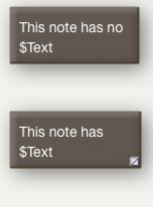
The caption's font may be changed using the font palette (by default the bundled Sketchnote font is used). Note that the font selected is used for the entire caption (no mixing of fonts).

Content Dogears

In Map views, Tinderbox uses a 'dog-ear' (like the folded page corner) to denote the presence of content; Outline view uses a different method, with content-related variable note icons.

The dog dog-ear is not interactive, it simply indicates that a note has content. In Map and Treemap views, the dog-ear is bottom right of the note icon, in Charts it is at bottom left of the note icon.

The background colour of the dog-ears is the same variable colour used for outline icons to indicates the age of the note (since last edit). Originally, the background colour of the dog-ear was the view's background colour.

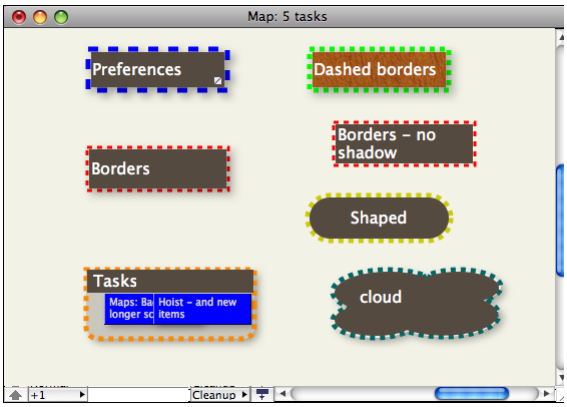


Icon dashed borders

Map note icons may display dashed borders if:

- The icon is simple note and not an agent, container or adornment.
- The $BorderBevel type is 'plain' (N.B. not the default setting).
- The $BorderDash has a value greater than 0.

The $BorderDash controls the length of each dash; the Border inspector sets the dashes to 5 pixels (use actions or Info view to set other values). The width of the dash is drawn as per the $Border value.



Icon drop-shadows

Notes may cast a shadow when drawn on maps.

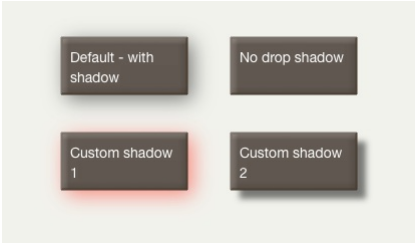
The following new attributes control the appearance of shadows:

- $Shadow (boolean): if `true`, a shadow is drawn
- $ShadowDistance (number): the offset of the shadow
- $ShadowBlur (number): the blurriness of the shadow
- $ShadowColor (color): the hue of the shadow

On very large maps (and/or less powerful computers) it can help with performance of Map view is shadows are disabled by setting Shadow to `false`. There is no preference to do this app/TBX-wide but stamps to agents can help. With agents consider use of the inside() and descendedFrom() queries with $OnAdd action of `$Shadow=false`. To reverse the setting use the same query but with `$Shadow=true`. Or write a toggle action:

```
if($Shadow){$Shadow=false;}else{$Shadow=true;}
```

Of course, being a toggle, a stamp like above would only be useful if all notes selected had the same $Shadow state!



Icon fill textures

Map icons can use texture fills, for containers and agents the fill is used within the caption area of the icon but not the viewport. These textures are set via the $Fill attribute and the opacity of the fill texture can be further controlled by the $FillOpacity attribute.

If $NameColor (the colour of the title text) is set to the default of 'automatic') then the icon's title will be black or white text depending on the colour value of $Color and not the colour of the fill texture image. Thus if using fills it may be necessary to also set an explicit $NameColor value in order that there is sufficient contrast to read the title.

The texture is used to fill the whole face of notes and adornments. For containers and agents, the texture fills the part above/below the viewport as used for the title & text/table.

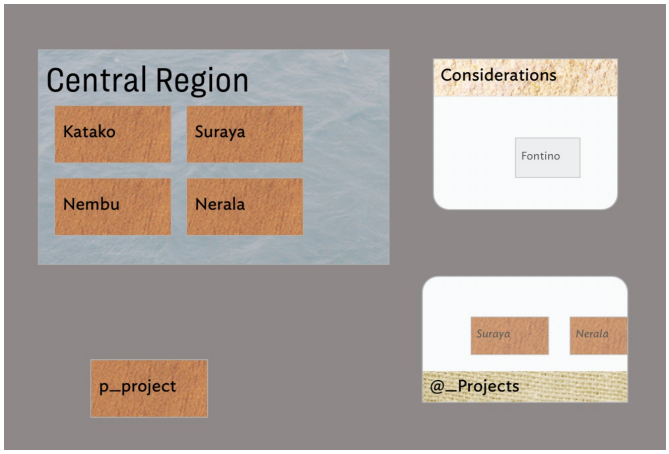
Built-in textures have the following name values:

- linen
- sandstone
- steel
- wood
- water

Users can supply additional custom fill artwork; only JPEG format files are supported.

When setting $Fill, Tinderbox accepts a file name with or without an extension. If the user file folder contains the file "TestFill.png", then setting $Fill to "TestFill" or to "TestFill.png" is equivalent.

If a note has a fill, ⌘⇧-drag inside the note to move the fill image. Such a the fill image is scaled to fill the width of the note, and the centre of the image coincides with the centre of the note. $FillOffsetY moves the image vertical centre up or down from the note vertical centre.



Note Flags

Map view allows optional 'flags' to assist in qualitative analysis and coding. One common Tinderbox task is qualitative analysis of existing materials, such as letters, surveys, diaries, and personal papers. An important preliminary step in this work is coding—identifying occurrences of special interest for the study. For example, if we were analysing a collection of nineteenth-century diaries to study what people recorded about food and drink, we might want to code where the food was consumed. We might mark every passage that discussed eating at home with the code P1, eating at the residence of another family member with the code P2, eating at a pub with P3, and so forth. We might also note places where money is discussed: C1 might indicate that the writer paid for their meal, C2 that someone else explicitly paid for the writer's meal, and so forth.

Flags offer a convenient and flexible way to foreground selected codes in map view. Flags are defined via $Flags, a List-type attribute. When not empty, small "flags" are displayed above the note in map view. (N.B. flags do not appear in other views). Note that as $Flags is a List-type attribute, actions can add new flags with "+" or remove flags with "-". Adding a flag to a note that already possesses that flag has no effect.

$Flags is an intrinsic attribute; it is not inherited from prototypes and an alias may have different flags from its original note.

For simple coding tasks, using $Badge may be adequate. Flags provide a wider range of visual cues, and new flags can be improvised quickly when coding needs change.

Defining Flags

Flags are described using a concise textual shorthand, and are illustrated in the screen-grab. (A visual flag editor is planned for the future.). Whilst it is generally easier to use named Tinderbox colours, hexadecimal values can be used to define flag colours as in the first example below. Possible design variations . Note the possibility of combining styles:

- Single solid colour. A red flag: `red`, or `#00ff00`, or `#00f`
- Horizontal stripes. List the stripe colours separated by hyphens: `red-white-blue`
- Vertical stripes. List the stripe colours separated by the vertical bar character: `blue|white|red`, or `green|white|blue|white|green`
- Diagonal stripes. List the stripe colours separated by the '/' symbol: `yellow/black`, or `green/yellow/blue`
- Chequered flag. List the two colours separated by the '$' symbol: `black$white`
- Cross symbol. List the cross colour, a '+' sign and then background colours: `white+red`, or `white+yellow/black`
- Saltire symbol. List the cross colour, an asterisk '*' and then background colours: `white*light blue`
- Diagonal line. The '%' sign adds a diagonal line: `green%white`, or `white%light blue/red`
- Chevron. The '>' symbol adds a chevron: `green>red`
- Pall. The '}' character adds a pall: `white}red`
- Pall and Chevron. The pall and chevron work together: `white}green>blue`, or `yellow}red>blue-green`
- Text caption. A period '.' adds a short textual annotation (1-3 characters) on top of any other colour/pattern: `A1.red`, or `42.green>red`, or `BZ.yellow$light green`. The characters `>*+.$%}` are treated as literal characters if they are used the first three characters of flag code using the period marker, so `+.red` draws a plus sign on a red flag.
- Text colour. Though text is normally white, a colour may be specified: `black:C.lighter blue`, or `#ffd700:F14.black`
- The text may be an emoji: `.green`

Progress bars in Flags

From v9.5.2, Flags permit several additional expressions, useful for showing progress bars.

- bar(value[,min,max]). Draws a horizontal progress bar, using the note's $Color and $AccentColor values. If the minimum and maximum values are not specified, they are assumed to be 0 and 100. More on the bar() pattern.
- vbar(value[,min,max]). Draws the a progress bar vertically, using the note's $Color and $AccentColor values. If the minimum and

maximum values are not specified, they are assumed to be 0 and 100. More on the vbar() pattern.

- pie(value[,min,max])). Draws a pie chart, using the note's $Color and $AccentColor values. More on the pie() pattern.

When drawing flags using bar() or vbar(), Tinderbox draws the flag outline in $AccentColor. This should be helpful when $Color contrasts poorly with the view's background. From v9.6.0, Flags using the pie-chart progress indicator draw their outline in $Color, and so are easier to read.

Using Text in Flags

if using the text method above, avoid using colon, semi-colon and period. These get interpreted respectively as a dictionary key delimiter, a list delimiter and the flag style definition delimiter. So, if wanting to use '3.1' try '3-1' instead. Also, the default 'fill' colour is the same as the text so do defined a colour. So not `3.1` or `3-1.green`, but `3-1.green`. Don't forget text is limited to only 3 characters or fewer. If more are supplied, the result is the first two characters and an ellipsis characters. In the previous example, 3-1' is *3 characters*, as the hyphen still counts as one.

Defining multiple flags for a note

As $Flags is a List-type attribute, if defining multiple flags for a note, use a semicolon between each flag's code. Thus, this code ' `white}green>blue;yellow}red>blue-green`' defines *two* discrete flags for a *single* note.

Multiple Flags and icon width

The default map icon will display up to 4 flags, or the first four listed flags if more. By dragging the note wider, all flags can be displayed (as illustrated).

Colours and Tinderbox Colour Schemes

As Tinderbox colour schemes files allow a named colour to be any value, it may be safer to use use colour values (e.g. `#00ff00`) rather than named colours (e.g. `red`) if you are not using the Tinderbox v7 standard scheme (as used to illustr here) or consider switching between colour schemes. Another approach is to make new 'new' colours for use in flags. So, if your favourite colour scheme renders the colour named 'green' not as green, you could make a new colour 'fgreen' and set the desired (visual) value of green and then use 'fgreen' instead of 'green' when setting $Flags. The exact name of the colour doesn't matter as long as it is not the same as an existing named colour.

If adding such extra named colours, be aware that they only existing in the current TBX document. Applying a different colour scheme to the document should not affect such custom colours (unless the new scheme includes a same-named colour with a different defined colour value). Thus a good idea, after adding extra colours, is to save the documents current colours as a colour scheme file ensuring it is possible to re-assert the colours if something changes their value in the current document and have the possibility to apply the same scheme to other TBX documents.

Flags in Outline view too

From v9.5.2, in Outline view notes can display their *first* flag (only) instead of a colour swatch: see more detail.

Opacity & Transparency

Map note icons and adornments but not the title part of containers/agents can use transparency. The transparency allows background items to show through where notes overlap.

Note transparency is set via the $Opacity attribute, with 100 (default) being opaque and 0 being totally transparent. Unless the $Border is also set to 0, a note always retains a border regardless of transparency.

An alternate and easier way to get a transparent (invisible) map icon for a note or adornment is to set $Color to 'transparent'. Doing this suppresses borders and drop shadows, but the title is left visible.

However, if $BorderStyle is set to 'automatic' Tinderbox is clever enough to suppress borders if $Opacity is set to 0 (transparent).

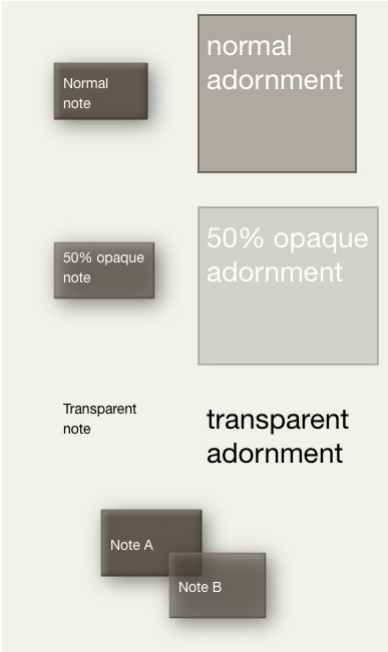
Transparency is effected within the normal stacking order of a map. Thus if transparent note A overlaps opaque B, if B has the higher $OutlineOrder, B will simply overlap A. If the $OutlineOrder is reversed A will both overlap B and B will be visible within the overlap.

Adornments, by default, already have a built-in fixed level of transparency. This is because adornments are often used in overlapping layout and the semi-transparency enables the areas of overlap to be seen more clearly. If fully opaque adornments are desired, an option on the Maps tab of Document Settings allows this, but note that it is a global setting affecting all adornments and overriding $Opacity settings. If it is desired to leave the default document setting for (semi-)opaque adornments, then for adornments desired fully opaque it will be necessary to set their $Opacity value to greater than 100 (a value of over 260 may be needed).

The opacity of a note's title and subtitle can be set separately from overall note opacity via $TitleOpacity and $SubtitleOpacity. Be aware that $Opacity still trumps the latter two attributes, if their values are greater than that of $Opacity, the latter value is used for all. Thus if $Opacity is 100, $TitleOpacity my be 50, half that of the overall note. But, if $Opacity is 50, $TitleOpacity my be 100 the title is drawn at 50% opacity.

In short, the latter ends up being generally what might be expected. It does preclude having a 'invisible' note with a visible title. To do the latter, set $Border to zero and $Pattern to "none" and $Color to the parent container's $MapBackgroundColor.

Automatic colours, such as are used in note titles, obey $Opacity settings.



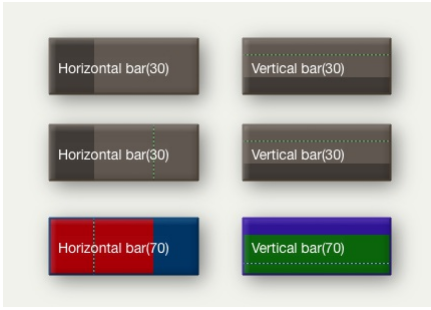
Note progress bars

Notes and containers in map view can make use of patterns that simulate horizontal (bar) or vertical (vbar) progress bars. As the patterns can use evaluated arguments as well as fixed numerical values this form of mark-up allows the progress of tasks to be monitored and displayed in the background.

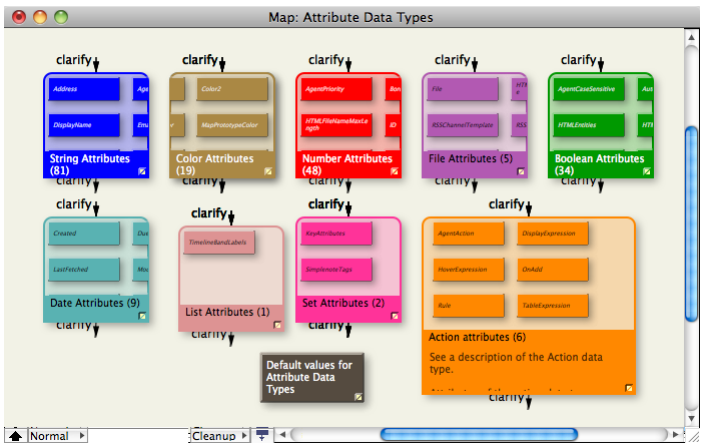
Containers (including agents) can use progress bars too though the progress bar is drawn across both title bar and icon, not just the title bar. Use of progress bars is not compatible with use of container plots as both features use the Pattern attribute to store their value.

This progress bar can also be displayed in outline view using swatches.

Progress Bars accept an optional fourth argument, a target value, casing a thin line to be drawn at the target position.



Note, Container and Agent icon layout



In Map view, notes and agents are differentiated similarly to an outline, where note containers have the title bar at the top and for agents it is at the bottom. However, whereas agents always show up as a container, regardless of whether they contain anything, a note only shows as a container if it has children.

Unlike Outline view, Map view agent icons do not give an indication of $AgentPriority status.

Notes within agent maps are always aliases. Setting $CleanupAction allows re-arrangeable agent maps.

Notes accommodate titles

Note windows can be resized easily in order to accommodate the full title of the note. This is most useful in Map view when taking notes in Map view using the titles only to capture content (e.g. rough notes during a talk).

The feature is accessed via the Note menu. There are two options:

- **Expand Horizontally** (Map view only, otherwise greyed out). Tells Tinderbox to attempt to widen the note's map icon to display the whole note Name (title). The selected note's icon is expanded at the right side, maintaining existing height and X/Y origin. May be used in conjunction with Expand Vertically (below). The revised Map note width is retained for the test of the session and persisted if the TBX file is saved. Keyboard shortcut.
- **Expand Vertically** (Map view only, otherwise greyed out). Tells Tinderbox to attempt to increase the depth (height) of the note's map icon to display the whole note Name (title). The selected note's icon is expanded downward, maintaining existing width and X/Y origin. May be used in conjunction with Expand Horizontally (above). The revised Map note height is retained for the test of the session and persisted if the TBX file is saved. Keyboard shortcut.

Also consider the feature to display a note's body copy.

A preference is available to automatically apply one of the above (or neither) as a default for new notes.

Containers and agents also display multi-line titles (and even body text), if sufficient space is available.

Notes displaying body text

In map views, large notes display (a portion of) the note's body copy text ($Text) as well as the title. The note's text only appears if:

- The note has some text.
- The note icon is sufficiently large more than the title and (option subtitle).
- For containers, including agents, the title height ($TitleHeight) area is sufficiently large.
- The note is not an adornment or picture adornment. Adornments do not have text. More strictly, they do have $Text but even if it has a value it cannot be displayed like note text.
- The note does not use a shape for display. From v9.6.0, text thumbnails are now drawn in rounded (shaped) notes as well as normal rectangular notes.
- The note has no $TableHeading or $TableExpression set. See below re table expressions.
- $MapBodyTextSize is not set to '1', so as to suppress table/text display.
- Notes with no title will display body text if present; previously body text display was suppressed in such notes.

Styling

Do note that the text that is displayed is **not styled**; think of it as a $Text 'preview'. Also, any embedded images within the text are not displayed.

Text Size

An attribute $MapBodyTextSize lets a specific font size for the body text be chosen, including suppressing any display of text. $MapBodyTextColor sets the colour used to display such text.

Setting $MapBodyTextSize to a value of 1 will suppress text display even if there is body text.

Font

The font that used to render the title is set via the Maps document setting Map Font but can be overridden at note level by setting $NameFont to a different value.

The font that used to render the text is set via the Text document setting Text Font but can be overridden at note level by setting the $TextFont to a different value.

Alignment

Displayed text is aligned according to $TextAlign (default: "left") whilst title and subtitle alignment uses $NameAlignment (default: "left").

Once a map note displays body copy its title is no longer vertically centred; there is no setting to override the latter.

$TableExpression vs. $Text

A table expression is drawn in the same part of the map icon used to display a note's $Text, therefore both cannot be shown in the same map icon. If $TableExpression or $TableHeading are set, $Text will not be displayed. More on container summary display tables.

Images in $Text

Inline images within $text will be rendered if there is sufficient space. If an image is the very first piece of $Text, a different render, whereby the image is used as a fill—occurs.

Note icon size and prototypes

When a note adopts a prototype, the note's $Height and $Width are initially changed to match those of the prototype. However, this is a one-time effect $Height and $Width are not inherited; the change affects only the initial dimensions of the note.

When assigning a note to a new prototype, the note's initial $Height and $Width are set to match the prototype's height and width *provided that the note's dimensions have not already been changed from the default*. If the note has been resized, those dimensions are preserved.

Notes with images in text

If a note begins with an image, that image is drawn on the face of the note, as if it were a fill. If the note's $NameColor is automatic, the colour of the note is changed based on a colour sampled from the image, so that a contrasting title colour may be chosen automatically. If the title is still an annoyance set the $NameColor to 'transparent'. This is most easily done using the 'Color' control on Text Inspector's Title tab.

This 'fill' display process will not occur if there is any text (including spaces) *before* the first image in the text. If such exists, normal rendering of note text/images will occur.

Shaped Map notes

Notes and adornments may be set to use differently shape icons in Map view, i.e. other than the normal rectangle. Whilst adornments can use shapes, if using shapes it may be necessary to use a $NameColor setting other than 'automatic'.

The shape of a note is determined by its $Shape attribute, and is inherited from its prototype (shape name values are case-sensitive). The $Shape submenu of the Note menu lets you change the note's shape conveniently.

The 'rounded' shape is new to v9.5.0.

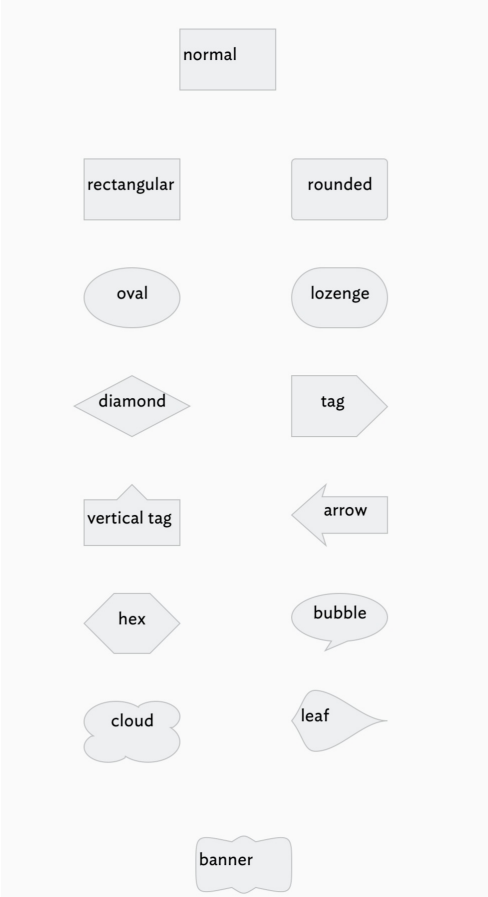
Shapes are still used for notes displayed in the viewport of an agent or container.

The shape of agents and containers is fixed; they ignore the $Shape attribute if it is set by the user. Thus a visible shape can only be set for a note with no children.

With the exception of 'cloud' and 'bubble' shapes can be transparent and have borders; 'cloud' and bubble' have no border if made transparent.

Shapes are probably best reserved for special occasions/purposes, in order to emphasise exceptional or unusual notes. Otherwise, maps may become a little busy.

Notes with non-rectangular shapes may use the bar, vbar, pie, and ring progress-bar patterns.



Subtitles

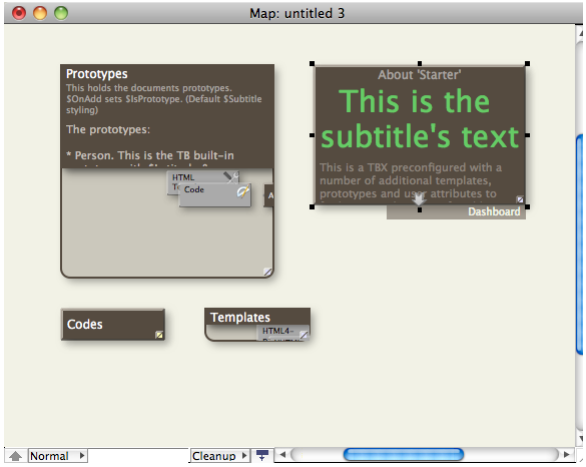
Map icons, including adornments, can show a subtitle ($Subtitle) below the main title ($Name) and above any displayed $Text. Further attributes control the colour, opacity and text size of the subtitle. The alignment of subtitles is linked to that of the note's title, i.e. $NameAlignment controls *both* title *and* subtitle.

Even if a subtitle is set, it is only drawn if the map icon affords enough (vertical) space to draw it.

Subtitles can be edited directly when Edit-in-Place is active.

The subtitle's text is not part of the $Text. However, it could be. For instance, by using $Subtitle action code to set it as the the first sentence or paragraph of $Text or by setting $Subtitle to the extracted first paragraph of $Text.

The illustration shows default styling on the left and the ' Dashboard' built-in prototype's styling on the right.



Map Units

Tinderbox uses the same units for a number of Map view related attributes:

- icon sizes
  - $Width
  - $Height
- map co-ordinates
  - $Xpos
  - $Ypos
- viewport positioning co-ordinates
  - $MapScrollX
  - $MapScrollY
- container title bar depth (vs. viewport)
  - $TitleHeight
- fill configuration
  - $FillOffsetY

When used for position, attributes use two unit values to give Map co-ordinates.

Whereas co-ordinate values may be positive of negative, for basic sizing positive values should always be specified.

Note: *the following applies to v5.x or earlier and will likely not be accurate with 4k displays.* To map Tinderbox units to pixels use this formula: (N*36)-4 where N is the map unit figure. Thus 3 map unit = (3*36)-4 = 104 pixels. The '-4' adjustmer is because Tinderbox allows a 2 pixel 'outer' buffer all round. As this buffer area never overlaps during automatic spacing of notes, it ensures at least a minimum 4 pixels gap between map icons.

Navigating map views

Why do some icons look different?

Although the actual colour of each note's icon in a Map View may vary by the colour-related attributes you set, it will always appear in one of two basic styles:

- Notes without children. A solid block of the colour $Color with at note title in contrasting black or white text.
- Notes with children. These are notes are termed 'containers'. They also have a section of solid colour at the top with the note title but in addition the icon is extended vertically and a grey area inset into it, referred to below as the 'viewport'.

Also

- Alias notes/containers. As for their originals but the title text is **always** in italics (and this cannot be overridden).
- Containers on the (child) sub-map do note show (grandchild) items but if they have a container plot then this is displayed.

What are the blocks of colour in the grey area of the container notes?

These represent child notes inside the container, i.e. you are seeing down one level into the the child notes' map. By default the child notes in the viewport will show their titles as well. If the sub-map in the viewport does not have a grey background that is because its $MapBackgroundColor has been set to a non-default value. Badges and Shapes are also shown, as indeed are most of the map visualisation attributes.

Why is the title area at the bottom of some containers and not the top?

These indicate the container is an agent1. To clarify, in a Map view a normal note container has its title area at the *top* of the note whilst an agent container has its title area at the *bottom* of the note.

How do I navigate down / open a container / drill down a level?

The three preceding questions are effectively all the same. To drill down into any container, you double-click the viewport area of the note and you then navigate down/open/drill down, whichever euphemism you prefer. Should you mistakenl click the top (title) part of the note icon you will simply set focus on note. If the latter happens by mistake, do not worry, simply close the note and make sure you double-click the lower portion of the note. This method works for both normal (note) and agent containers. An alternate method is to select container and press the Down Arrow key (↓) or use the View menu ▸ Focus view (⌘+⇧+↓).

How do I navigate up a level?

To shift the map view's focus to the to the parent map ('drill up') use the Views menu ▸ Expand view (⌘+⇧+↑), or just use the Up Arrow key (↑).

How do I quickly get back up to whole-document outline focus?

- Click the left most segment of the pane breadcrumb bar.
- Use View menu ▸ Expand view (⌘+⇧+↑) repeatedly until at root level (view does not change focus).

How can I tell if my current outline view has whole document focus?

The view pane breadcrumb bar is not displayed. Below root level is shows the ancestor containers of the current scope.

Can I interact with the child map in the viewport?

Yes you may, though it often helps to drag the parent container's icon a bit larger to increase the size of the viewport. You can do two things:

- 'Scroll' the child map. To do this click onto any *background* area of the child map, i.e. not on a note/container icon, and drag. This will scroll the *child* map within the parent's viewport. A container's viewport will remember the section the last-used (visible) section of the map. The date is stored in the attributes $MapScrollX and $MapScrollY.
- Promote notes/containers from the child map to the current map. Simply click on the note or agent in the child map and drag it out of the viewport onto the parent map.
- Demoting note/containers from the current map to the child map. Click on the note or agent in the current map and drag it into a container's viewport. It is not possible to nest two item on the child map, you must drill down to do that.

Can containers show note body copy text, display tabular info, etc., like non-containers?

Yes. For containers, note or agent, the attribute $TitleHeight controls how much of the map icon is devoted to the viewport. The only difference between note and agent containers in this regard is that the body text or $TableHeading/$TableExpression information is always displayed beneath the icon title, which makes sense when you see it on screen, regardless of the relative position of the viewport. To set $TitleHeight manually in map view, simple drag the bottom (note container) or top (agent) of the solid colour area of the icon and drag it up/down to alter the relative allocation of space to title/text viewport. To provide more space overall drag the corner of the container's note icon; as the icon grows/shrinks in size, the title/viewport allocation scales accordingly allowing the $TitleHeight to then be re-adjusted. Being an attribute, $TitleHeight can also be set via action code.

Positioning newly created notes

When Tinderbox needs to choose a map position for a note (e.g. in an outline when the note is created, or dragged into a new container), it is careful to avoid colliding with existing notes. Notes created outside Map view view are automatically assigned plausible map positions.

[Ctrl]+[Opt]+Return (^⌥-Return) creates a note below the selected note. Return creates a note to the right of the selected note, and [Ctrl]+Return (^-Return) creates a note to the left of the selected note. If there is already a note where Tinderbox would place the new note, Tinderbox seeks a suitable location.

Note: in earlier versions of Tinderbox, the app would sometimes give up the search for a plausible position and place all new notes at {0,0} (in map co-ordinates). This should no longer occur but the issue may be seen in old TBX files created using older versions of Tinderbox.

Map Posters

Posters let Tinderbox incorporate all sorts of visualisation tools into the map view. Consider posters as a way to use powerful visualisation libraries inside Tinderbox, opening up a powerful canvas on which to draw. It is a mistake to think of this web view as an intended way to view web pages—although that *is* possible.

Any note can have a poster, or more precisely, *be* a poster. Notes that have a poster display a small web view in front of the note icon. The poster cannot be moved so as to show the normal map icon. In a map, any note is either a normal note icon *or* a poster.

Be aware that:

- posters are displayed *only in map view*. (They behave normally in other views.)
- a note's poster obscure its normal map icon.
  - a note is **either** a normal map icon **or** a poster. If wanting a poster *relating to* a note that needs to remain visible at the same time as the main note itself, make a sibling (poster) note as this allows both to be visible at the same time.
- a note that has a poster takes responsibility for drawing and updating itself.

The poster feature is supported by the ' Poster' group of system attributes.

Template-based posters vs. posters of line URLs

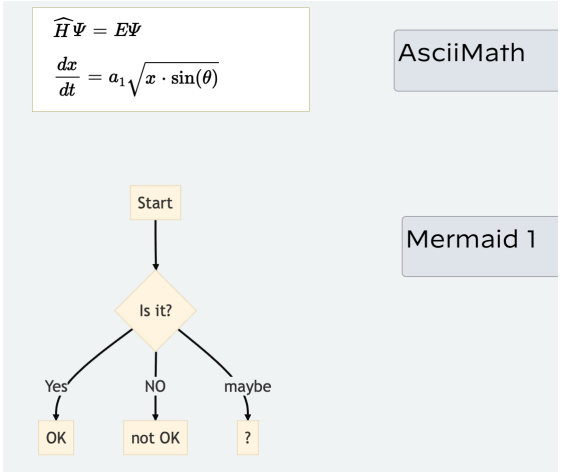
A poster can be defined either via a Tinderbox template or by supplying a URL, though the latter is primarily there for less common cases where the template method cannot be used. If, by mistake, the user specifies both a template and a URL, the template approach is used.

From whence comes the poster data?

In the most simple example the post note holds any necessary data, at least as regards data held in the TBX as opposed to being drawn from the web or local disk resources.

Javascript or other languages?

This has some nuance. As a poster essentially shows what might be seen in a web browser. So all, or at least some of the code needed with be JavaScript. But, in principle, anything a browser can show (e.g a Java applet) *might* possibly work. The more the envelope is pushed in terms of types of code in use, the more the user should start by testing a web browser before migrating the code to a poster note. The more complex/exotic the mix of code the greater the likelihood of meeting unexpected issues, especially relating to (OS) security—i.e. the poster code trying to access information outside the TBX that the OS believes should not be accessible for safety reasons.



Poster of a note vs. note as a poster

An easy misconception is to think of a poster as the visualisation of a note and assume that the poster can be moved to also show the 'normal 'map' icon. *This is not so*! In such a case, it is necessary to make a *new note* to 'host' the poste as the two are now discrete entities that can be positioned independently on the map.

Other notes about posters:

- Creating new posters
- Updating posters
- Posters and performance

Creating new posters

Importantly, the user is responsible for understanding how the visualisation works and is configured. Tinderbox is merely providing a means to display a *correctly configured* visualisation within a Tinderbox map view.

If making a poster from scratch, as a opposed to adding new data to a pre-made poster, there are some points to consider:

- if unfamiliar with the visualisation tool, consider make a working copy in in a web browser before trying to implement in Tinderbox. Why? Making a working browser-based test will:
  - identify any libraries, e.g. JavaScript files,
- $PosterTemplate is independent of the note's $HTMLExportTemplate. The former affects only the in-app poster, the latter the export of the note. Thus a note can still export even if it has a poster.
  - For debugging, it may be handy to set the $HTMLExportTemplate (temporarily) to be the same as the poster template. Doing this enables use of the Text/Export pane to review and correct the HTML being passed to the poster.
- Generally, set the template such that the plot area fills the entire page)i.e. the whole poster)
- A likely problem on first use is the appearance of unwanted
  elements in the middle of Tinderbox data, e.g. `^text^` passed into the poster's Javascript code. Do not forget to use either the built-In Code prototype for notes using posters, or otherwise turn off paragraph markup.

Updating posters

Posters reload themselves:

- when the map view is laid out
- when the poster is selected
- when a note scrolls into view.
- Posters loading via $PosterURL update if $PosterCSS is changed

Thus, changes to the poster's data might not be reflected immediately.

The action changed(), allows the designated note—default is the current note—to update itself, allowing a stamp to call the poster to update itself. That is useful if some calculation needs to be done before the poster is displayed.

Posters and performance

Posters in general do not pose an issue, but there are two aspects of use of which to be aware

- some visualisations may either/both use a lot of computational effort or call on very large datasets. Tinderbox can use lots of processing power; a more likely limit may occur in terms of the overall CPU power/RAM of the host Mac and the demands of other currently running programs.
- posters can make significant demands inside the map view, which is already quite demanding. Some performance optimisations are already in place, but it may be good to understand them. If a note is well off-screen, i.e. outside the currently visible area of the map, it loses its poster (i.e. the poster is not calculated/drawn) and regains the poster only as the note approaches the viewport (i.e. so it is drawn ready when the poster note is scrolled into view).
- any visible poster is refreshed when the map is laid out or refreshed.
- to reload a poster, click the tab label area of the current tab, which asks Tinderbox to re-draw the current map. Additionally, if a poster is asked to reload its data, it will wait a minimum of five seconds from the most recent reload—this avoids unnecessary re-calculation due to rapid multiple clicks, e.g. double-clicking the tab rather than single-clicking it.

In summary, when planning poster use consider both the effect of each individual poster *and* the number of posters in use within a map—or at least the number visible on screen at the same time.

Re-arrangeable Agent Maps

By default, a map of an agent cannot be re-arranged. This is because at each agent update cycle, the aliases are refreshed adding/deleting aliases as their originals match, or cease to match, the agent query.

However, if the agent is turned off ($AgentPriority or Action Inspector Query sub-tab or Get Info agent tab.) if is possible to then manipulate the map's contents. When 'off' the agents ceases updating but it still retains its contents from the las active agent cycle. If the agent is turned on again, the map is re-arranged to the normal agent-organised layout.

Maps within an *active* agent can be rearranged by the user, which attempts to get around the above problem. Note that this flexibility applies to the *agent-created* aliases on the map; the user *cannot add additional items*. If the user deletes alias it will be re-created next agent cycle. Also, like normal aliases, it is not possible for the agent's aliases to contain other notes so no nesting is possible.

The feature is controlled by the agent's $CleanupAction. In the default mode, the agent's contents in map view will be organised in a grid, as was previously the case (see above); the order of items in the grid being set again, as before, via the sort attributes for the agent container. A number of cleanup layouts are supported (seen in the Cleanup Action pop-up list), with items being ordered via outline order ($OutlineOrder) unless the parent container has no sort ($Sort) set:

- **grid** (default). Aliases are arranged in a grid layout, left-right/top-bottom in sibling order.
- **row**. Aliases are arranged in a single row, left-right in sibling order.
- **column**. Aliases are arranged in a single column, top-bottom in sibling order.
- **box**. Aliases are arranged in a an open box (square) arrangement, clock-wise in sibling order from top-left.
- **none**. Auto-layout is suspended, the user is free to arrange the items within the agent containers as they would any other map. Whilst in this freeform mode the normal Map view Cleanup pop-up menu items may also be used. In "none mode Tinderbox adds new aliases to the agent map by the logic used for a normal map so it should avoid overlapping/hidden icons, etc.

Changing the Clean-up order

By default, items are arranged ordered via outline order ($OutlineOrder) unless the parent container has no sort ($Sort) set. If a sort is set, the items are listed in ascending sort order. This will vary by data type e.g. number-type: lowest first, date-type: earliest first). This can be reversed by setting the $SortBackward attribute or ticking the 'reverse' box on the Sort Inspector.

Adornments

User-added adornments are allowed in agent maps—an exception to the only aliases forming the child objects of an agent. If the agent sort is enabled, i.e. $CleanupAction is "grid", this trumps the effect of smart adornments; smart adornments themselves ignore $CleanupAction completely. Adornments can be dragged or pasted into agent maps.

If an adornment is added inside an agent, the agent's $CleanupAction is set to "none".

Agents

Agents do not force a default cleanup on file load. Thus agent maps using "none" correctly maintain their layout across different sessions.

Should the agent be re-enabled, the custom icon positions will be lost as the agent rearranges the icons.

Regardless of whether an agent is enabled it is always possible to link to/from aliases on the map of the agent's contents.

Scrolling newly opened maps

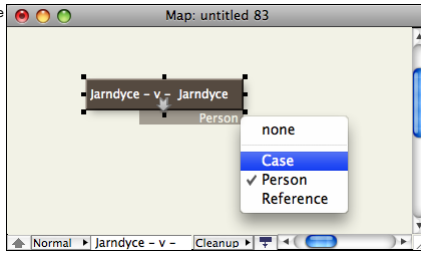
Container maps record their map scroll position in two attributes, $MapScrollX and $MapScrollY. (Very early versions automatically scrolled new map views to place the first child in view.)

These co-ordinates are floating-point numbers, and use the same relative co-ordinates as $Xpos and $Ypos. If a map view of a note is closed and later reopened, it will scroll to that map's most recently viewed position.

The scroll position is also reflected in the interior view of a note container viewport (as before an agent's contents are fixed). For a note container, data at $MapScrollX and $MapScrollY is drawn in the vertical/horizontal centre of the container viewport.

The viewport within a container can be moved via click-drag within the viewport area. Dragging the child map background scrolls the child map, dragging an icon moves that icon on its own map.

Selecting a prototype in Map view



In TBX documents with prototypes *already defined*, when a map icon (note, container, agent but not adornment) is selected a tab in $Color is shown bottom right of the icon. Otherwise, the tab is not shown, regardless of whether a note is selected. This method makes it much easier to leverage Map view for fast note-taking.

If a prototype is set for the currently selected note, the tab shows the prototype name or else the tab is blank. Map adornments do not show a prototype tab when selected.

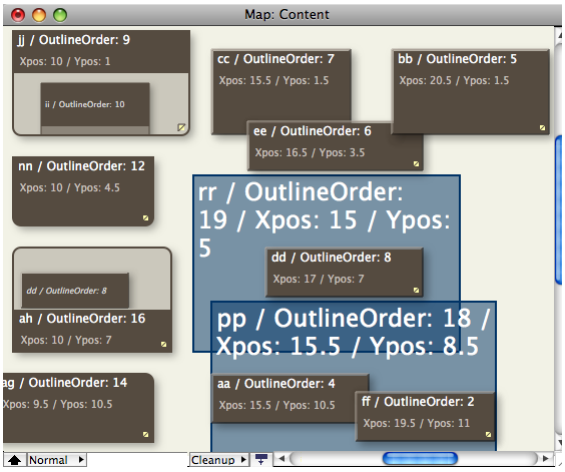
For a *selected* note click-holding or right-clicking on the icon's prototype tab gives a pop-up list of all prototypes available in the current TBX allowing the user to set, change or clear the note's $Prototype value. Such a menu is only displayed if there are prototypes defined in the TBX. Setting a prototype on an alias sets the *original note's* Prototype value.

If multiple notes are selected, using the above method on any one of the selected items' prototype tab will result in all items changing their prototype assignments.

This is just one method of setting a note's prototype.

The text colour used in prototype tabs is normally the background colour of the map ($MapBackgroundColor). This usually works well, but can be difficult to read in low-contrast colour schemes where the note colour is close to the background colour. In the latter scenario, if the two colours have similar luma, Tinderbox instead chooses either black or white for the prototype tab text.

Stacking and overlapping items



When two map note items overlap, the item with the lower $OutlineOrder value will stack on top of the other item and obscure it (unless note transparency is used).

Adornments follow the same principle between adornments but all adornments are drawn behind all notes/containers on the map.

If working in map view and the order of items in outline view is not important, the Note menu offers a number of functions (all with shortcuts too) to alter the $OutlineOrder of a not so as to affect its stacking order:

- Move to Front. Sets $OutlineOrder lower than any other note/container on the map.
- Move Note Up. Sets $OutlineOrder one number lower.
- Move Note Down. Sets $OutlineOrder one number higher.
- Send To Back. Sets $OutlineOrder higher than any other note/container on the map.

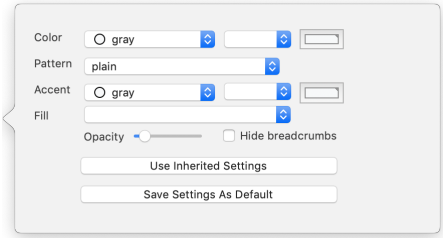
N.B. $OutlineOrder is *read-only*. The user cannot make the above adjustments using action code directly on $OutlineOrder.

Also see a more detailed discussion of Outline vs. Map view for further discussion of stacking.

Map Settings pop-over

This is the configuration pop-up for the current Map view:

- **Color**. Colour controls to set the parent object's $MapBackgroundColor.
- **Pattern**. Uses Pattern pop-up to set the parent object's $MapBackgroundPattern.
- **Accent**. Colour controls to set the parent object's $MapBackgroundAccentColor.
- **Fill**. Uses Fills pop-up menu to set the parent object's $MapBackgroundFill.
- **Opacity**. Slider control to set the parent object's $MapBackgroundFillOpacity.
- **Hide Breadcrumbs**. Toggles view's breadcrumb bar (default: ticked).
- **Use Inherited Settings**. A button to reset all the above attributes to current document's default values.
- **Save Settings As Default**. A button to set all the above attributes as the current document's default values.



Outline view

An outline view shows the hierarchical structure of the entire document as an outline. An outline is useful for seeing many notes in one window. It is easy to restructure the hierarchy of the document in an outline window, by dragging parts of the hierarchy to other parts of the hierarchy. It is generally the most useful view to use when laying out notes for export.

Containers (and agents), notes containing other notes, can be expanded or collapsed to show child notes (and there are shortcuts to assist with this).

Note titles can be double-clicked and edited in situ. Use **Return** [↩] to close the edit, or click off the edited note: use **Escape** [⎋] to abandon an edit. Long titles will wrap to a second line.

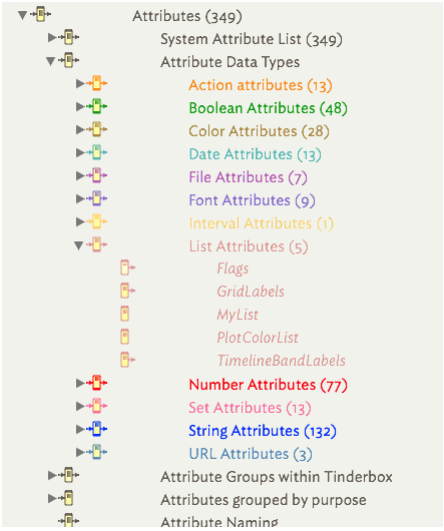
Using the **Tab** (⇥) key with or immediately after the **Return** (↩) key will result in a new, *indented*, 'untitled' note.

As with a normal new note, the indented note is presented with the note's title in edit mode.

An indented (i.e. child note, can be promoted as sibling of its parent (as next item in outline order) using Shift+Tab (⇧**Tab**). It is not possible to outdent the parent note of a hoisted outline (as some early releases allowed). If the hoisted parent note is selected, ⇧Tab is ignored.

The right-arrow key (→) expands the selected note

The left-arrow key (←) collapses the selected note if it is expanded. If not, it selects the parent of the selected note if the parent is visible in the view.



Note icons

Notes and containers have icons which illustrate:

- the degree of content (i.e. amount of $Text)
- the type of note, e.g. note/alias/agent
- the age of the content (time since $Text or Displayed Attributes were last $Modified)
- the presence of in- or out-bound links: see below
- the note's colour ($Color). Like the note title, the icon is drawn in $Color

Note title

The icon and the note title ($Name) are drawn in the objects' $Color attribute (i.e. the face colour of the object's map view icon.

To help with legibility, there is also a ' Darker colors' option in The Outlines tab in Document Settings that, as it implies, slightly darkens the colours used in the Outline view from their defined $Color value. The feature is 'on' by default, but is best turned off if using Outline as the main working view and using a wide range of colours for $Color.

Separators add the ability to delimit sets of sibling notes and are not drawn in Map and Chart views. In the outline they are similar, in a visual sense, organiser device in the way adornments are used in a map (and only seen in a map view).

Links

An outline view does not show any link lines (such as are seen in a map), but the note icon will show an inbound and/or outbound stub link if the note has one or more such links in the given direction; web links count as outbound links.

From v9.6.0, the icon symbol no longer shows an outbound link if the note has only Web links.

Untitled notes (within Outline views) are not auto-deleted if they have outbound links, or if they have inbound links other than prototype links.

Other features

Outline view can be enhanced by the optional display of item checkboxes and column display of attribute data. Here is a list of other features:

- Outline Settings pop-over
- Altering Scope of Outline views
- Checkboxes
- Colour Swatches
- Coloured Backgrounds and Selections
- Column view: displaying and editing column data
- Copying column view data as tabular data
- Double-clicking Outline background
- Drag re-arranging notes in Outline view
- Flags in outlines
- Horizontal scolling
- Outline filter
- Selecting Prototype via Outline icon
- Separators

In outlines:

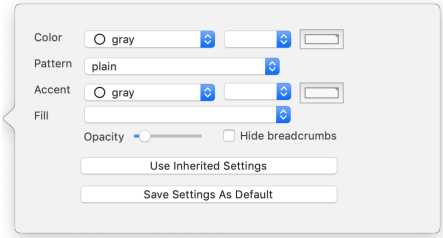
- the right-arrow key (→) expands the selected note
- the left-arrow key (←) collapses the selected note if it is expanded. If not, it selects the parent of the selected note if the parent is visible in the view.

Auto-scrolling:

- View ▸ Arrange ▸ Move Note Up (⌘-↑) and View ▸ Arrange ▸ Move Note Down (⌘-↓) now scroll the view pane if necessary to ensure that the moved note remains visible.
- If you use the arrow keys, fn+arrow keys, or create a note that is not visible in the window, Tinderbox scrolls the view pane to make the selected note visible.

Outline Settings pop-over

This is the configuration pop-up for the current Outline view:



- **Color**. The parent object's $MapBackgroundColor value.
- **Pattern**.The parent object's $MapBackgroundPattern value.
- **Color 2**. The parent object's $MapBackgroundAccentColor value.
- **Fill**. The parent object's $MapBackgroundFill value.
- **Opacity**. The parent object's $MapBackgroundFillOpacity value.
- **Hide Breadcrumbs**. Toggles views breadcrumb bar (default: ticked).
- **Use Inherited Settings** button: click to reset the document defaults for Outlines.
- **Save Settings As Default** button: click to apply the current view's settings as the document default.

Altering Scope of Outline views

Just as in a Map view it is possible to alter the focus (scope) of an Outline view. By default, a new TBX opens as an Outline view with every note in scope (even if folded away from view within a container).

Note: the commands here can also be applied to Treemap and Chart views. Map views have a few slight navigational differences.

Can I have an outline of just a part of my document?

Yes. Select the note to take focus and View menu ▸ Focus view (⌘+⇧+↓). If you'd prefer the new outline to be a separate window, instead use View menu ▸ Tab sub-menu ▸ New tab to make a new tab. This new tab opens in the current vi (which will thus be another outline).

Can I shift focus up/down one container level, as with a map?

Yes. To move up a level there are 2 methods:

- View menu ▸ Expand View (⌘+⇧+↑).
- Click the parent container of the desired new scope in the breadcrumb bar.

To move ('drill') down, there is one option only. Select the note to act as the new focus, then use View menu ▸ Focus view (⌘+⇧+↓)f you use the opposite shortcut to that for Expand View (i.e. 'drill down' shortcut) is merely expands the curr container in the view without shifting focus. This makes sense as there may be more than one child container and via the shortcut Tinderbox has no way to tell which sub-container is the users desired new focus.

How do I quickly get back to whole-document outline focus?

- Click the left-most segment of the view pane breadcrumb bar.
- Use View menu ▸ Expand view (⌘+⇧+↑) repeatedly until at root level (view does not change focus).

How can I tell if my current outline view has whole-document focus?

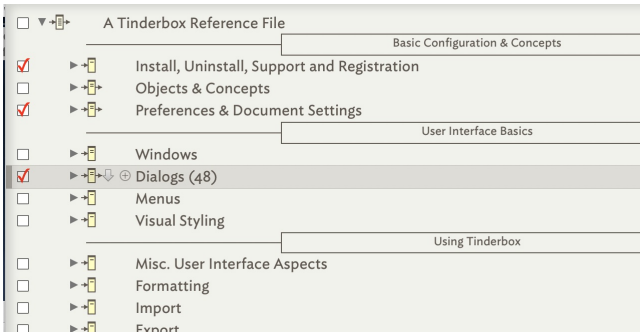
The view pane breadcrumb bar is not displayed. Below root level is shows the ancestor containers of the current scope.

Checkboxes

Outline views have a (non-default) option to display check-boxes for each item. This is enabled via the View menu. There is no document level preference for this setting as each discrete Outline view window has its own checkbox visibility state. Ticked boxes show a red-coloured tick.

Checkboxes are drawn to the immediate left of a note's icon, at the indentation level of note, i.e. the boxes are not drawn in a single vertical column but rather follow the indenting of the outline.

The ticked/un-ticked state of a box is stored in the Boolean the system attribute 'Checked'. The action of changing the tick state of the box via code or the UI has no direct effect other that that which is seen. However, Tinderbox code can be used to test and act on the $Checked values of notes.



Colour Swatches

From v9.5.2, swatches are deprecated and discontinued in favour of showing the first (if any) of a notes defined flags ($Flags), as described here.

This feature and its associated system attribute are now essentially non-functional.

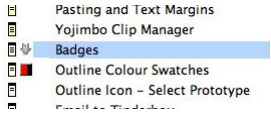
Legacy use

Outlines can show a square colour swatch, including border and pattern information, in the area reserved for the link widget between the note's icon and its badge. Because of the swatch's positioning, when a note with a swatch is selected, the link widget temporarily replaces the swatch.

The main aim of this feature is to allow the display of progress using the 'bar' and 'vbar' patterns that display a progress bar in map note icons.

Swatches are activated via the $OutlineColorSwatch attribute.

This feature can be toggled on/off via the Appearance Inspector's Outline tab.



Coloured Backgrounds and Selections

Outline items can have a solid coloured background. The attribute $OutlineBackgroundColor lets the user specify a colour (named colour or hex value). The colour then forms the background colour of that note's listing in Outline view; it does not run full width (rather like Finder item colouring) but does include check-boxes and column display if those features are turned on.

Outline layout has been revised to use $Color more attractively. If a note uses a $Color other than the document's default colour, the entire area of the note is tinted translucently with that colour. $OutlineBackgroundColor to tint the background behind the note. The tint is also varied in opacity so the degree of colour decreases from top to bottom of the selection highlight.

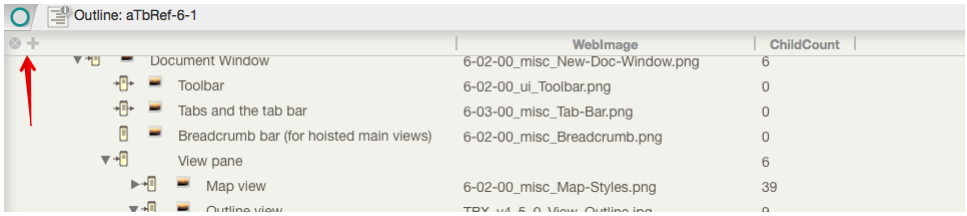
A selected item focus highlight uses a tint of $Color. The highlight is drawn on top of the outline background colour if any. These aspects are shown in the image where the first two and last items are selected.

The image also shows that thought needs to be given to contrast between $Color and $OutlineBackgroundColor or titles may become hard to read.

In documents that use 'Darker colors' in outlines, the selection highlight also uses the darkened colour.

To help identify selected item(s), a selected item's highlight has a solid vertical bar of the tint colour of the highlight $OutlineBackgroundColor.

Column view: displaying and editing column data



Outline views have a (non-default) option to display user-chosen attribute data in columns to the right of the note titles. This feature is enabled via the View menu. There is no document level preference for this setting as each discrete Outline view has its own column visibility state. As with $Name, any editable attribute value can be edited via Edit-in-Place.

Column data is drawn in the source note's $Color, and the current row of data is highlighted (as with the actual note title). With Edit-inPlace turned on (the default) column data—for any non read-only (i.e. most) attributes—can be edited in place. As with the normal note title edit, click and hold to enter edit mode.

When the columns option is turned on, a column header row is displayed. At the left end are two controls:

- '**X**' button. This turns off column view (the View menu can do the same). Any currently defined columns are remembered.
- '**+**' button. This adds a new column, to the right of any existing columns. The default name is 'Attribute'; as this is not a valid attribute name no data is displayed. To configure the column, click on the column head for the appropriate column and use the column format pop-up. The pop-up also allows a given column to be deleted.

Deleting a column has no effect on the data it was displaying. There is no set limit to the number of columns that can be displayed. The view will support columns displayed off-screen in which case the view pane will show a horizontal scroll bar. If all columns are deleted, Column view is turned off.

Separators still draw normally and therefore do **not** display column data.

Adding more columns will shrink the width of existing columns rather than grow the view window. Dragging the view window wider *after* adding columns does not re-expand the columns, so if planning to use use multiple columns it is a good idea to re-size the view window first.

Each tab with an Outline view will remember the columns used (such data is stored per view). Customisation details are stored internally in the TBX file. Any tab showing Outline view with columns that is open when a TBX is saved & closed will be remembered and restored when the file is next opened.

Column widths may also be altered by manually dragging the right-side divider bar of the column's header. The left-right order of columns can be altered by clicking on a column heading (away from its edges) and dragging left or right. There are some limitations to column manipulation. Column data does not wrap if wider than the current column, the data is simply only shown in part (with nothing to indicate more text).

Boolean attributes are displayed as tick-boxes.

Double-clicking a displayed column value allows a value to be edited (or entered into a blank 'cell'). If editing a value in an outline column, clicking inside the edit field places the insertion point is placed appropriately.

When a selection is copied, displayed columns values are copied in addition to the title and text in tab-delimited form to assist with export.

Copying column view data as tabular data

When copying from an outline view with columns, Tinderbox copies all the column data in tabular form. This allows easy and simple sharing with spreadsheets and other programs that use tab-separated values.

To copy just note titles, turn off column view, copy items and re-enable column view.

Double-clicking Outline background

Double-clicking an Outline view background creates a new note at the end of the outline, even when a title is in edit. This makes it easy to set up new separators.

Drag re-arranging notes in Outline view

A note, or selection of notes, can be re-arranged in Outline view by dragging the selection to the desired location. Keeping the cursor over the right part of other note names in the view, the user will see a green rule between existing notes indicating where the dropped item will be placed. The degree of indentation to the left edge of the rule will indicate where in the hierarchy the dropped note will be placed (i.e. as a sibling or a child of the note above it).

Note that dragging a hierarchical selection will flatten it at point of re-insertion in the outline; the selection collapses into outline order sequence.

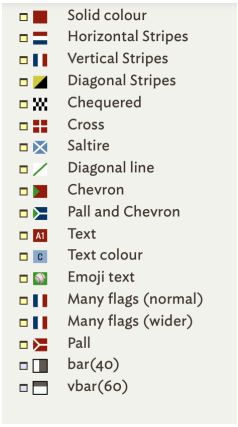
Dragging note(s) will alter their $OutlineOrder and other outline-position related attributes.

Flags in outlines

In place of seldom-used colour swatches, from v9.5.2 outlines instead draw the first (only) element from $Flags to the right of the outline icon of notes that are not selected and have a value for $Flags.

When a note with flags is selected, the flag is replaced by the link widget.

The syntax for setting flags, and the styles available are described in detail under their use as as note flags in map view.



Horizontal scolling

If Outline column view uses more columns than can be displayed within the current width of the view pane, the view can be scrolled horizontally.

Note: whilst this allows more columns to be displayed care should be taken not to overload the view by displaying very large numbers of columns. There is not limit, but if the view becomes sluggish it may indicate that it is sensible to remove a few columns from the view.

Outline filter

Outline view can be filtered using a query, as defined via the filter toolbar.

Selecting Prototype via Outline icon

Right-clicking on a note's icon in outline views opens a pop-up menu for selecting a prototype for the note (setting the note's $Prototype). Such a menu is only displayed if there are prototypes defined in the TBX.

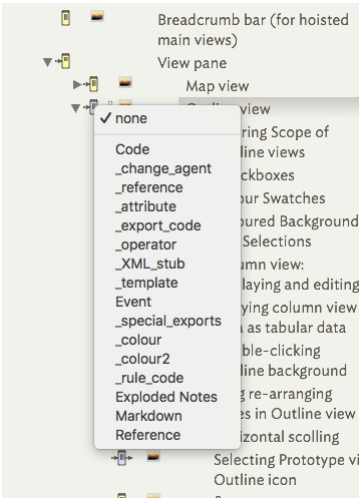
If multiple notes are selected, using this method on any one item will update the prototype assignment of the whole selection (including any separators.

Although separators have no visible note icon, right-clicking a selected separator at the left-most end, where the icon would otherwise be, will show the prototype pop-up menu.

Setting a prototype on an alias sets the *original note's* Prototype value.

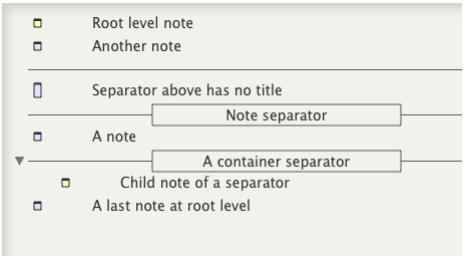
This method also works in Chart, Attribute Browser and Timeline views.

This is just one method for setting a note's prototype.



Separators

A note may be designated as a separator. A separator is drawn in outline view as a horizontal line. The note's name, if any, is shown centred in a box-out in the middle of the separator line. If the note has no Name value or it is "untitled' the separator shows as a solid line. For separators no duplicate name warning is given if the Name value is left blank. A note's status as a separator is held in $IsSeparator.



Just as adornments are only visible in Map view, separators are only shown as separators in Outline view (and exceptionally in Treemaps). However, in all other views except Map view, a separator is visible as if a normal note. By the same token children of a separator are hidden in Map view, as their parent container is not drawn on the map. The latter can be a useful way to 'hide' content, either notes or containers, in a map without removing the content from the overall document.

Note that separators were not originally designed to be able to be containers but rather like a (non-smart) adornment for separating groups of siblings in outline views. However, they can be containers; even agents may be a separator. Because of their possible container role, treemap views always show separators (as if they are normal notes). No note icon is shown for a separator note so the age/level of content/age cannot be judged although the separator is a fully functional note. In major views other than Outline and Map views, a separator is shown as a normal note.

A separator can still have normal note features like a Rule and OnAdd action and become a container by having notes added to it.

If data columns are enabled for the Outline view, separators still draw normally. Thus the data columns are not displayed for separator notes, unlike all other items in the outline.

Even if not needed as a separator for Outline purposes, making a Map item into a separator can be a way of hiding it (temporarily) from Map view; note that when hidden like this it cannot be selected from the Map either. To make the item visible to the map it is necessary to find and select it in a view other than map and remove the separator status.

Separators, but not adornments, are included in $SiblingOrder wheres both are included in $OutlineOrder. This makes sense when considering export is where SiblingOrder values are most regularly used and where adornments do not appear. See more on separators in a map vs. outline context.

A separator can have inbound and outbound links. It can also have note text; in such cases Treemap views will show a 'dogear' symbol denoting text content.

To designate a note as a separator, check the Separator checkbox in the Prototype sub-tab of the Properties Inspector set the value of $IsSeparator to `true`.

Separator notes default to exporting as HTML (only) if either:

- they contain both a title **and** have note text (i.e. have $Name **and** $Text values)

...OR...

- they contain child notes

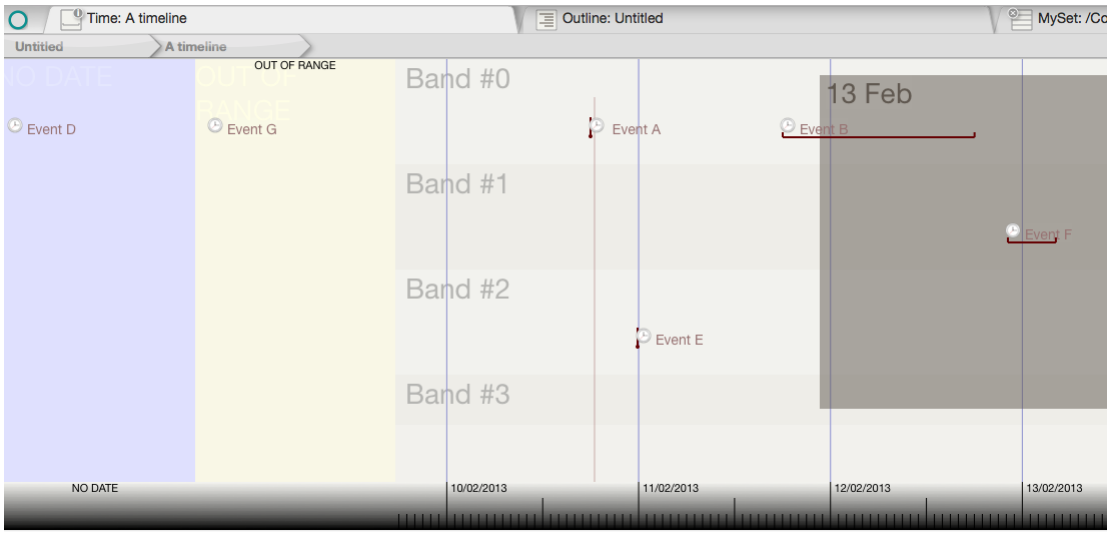
If exporting Separator notes with no text (i.e. with children) ensure either the template will link through to the child content or add a ^childLinks^ into the $Text of the separator note.

Making a note into a separator does not alter any of its font-size related attributes but when drawing an outline view Tinderbox draws separator titles very slightly smaller than for normal notes. Likely this is done so as not to expand line height i.e. allowing for the box around the separator title. This effect is more noticeable as zoom level is decreased.

Duplicated separators retain their original name rather than receive the name suffix " copy" making it easier to clone unnamed separators.

Separators do not appear in hyperbolic view or in the attribute browser.

Timeline view



A new major view displays the notes inside a container as a horizontal timeline. Each note is positioned along the timeline based on its $StartDate. The note's width is based on the value of $EndDate, if specified. There is limited support for negative (BCE) dates.

Most of the configuration controls for the view's controlling attributes are accessible via the Timeline Settings pop-up. This can be accessed from the 'i' icon on the tab's label or via the 'Change settings' in the view's context menu.

Double-clicking in the timeline creates a new note. The note will automatically be assigned a StartDate corresponding to the timeline date of the clicked location.

Events may be assigned to distinct horizontal bands in the timeline. For example, one band might describe political developments and a second might describe artistic landmarks, or one band might specify development milestones while another would involve documentation or marketing. The band for an event is specified by the attribute $TimelineBand; notes may be dragged between bands which will reset the attribute accordingly. Dragging a note beneath the bottom-most band creates a new band; empty bands are automatically deleted or not drawn if higher-numbered bands exists (i.e. only bands with data are drawn).

If notes have no start date, they do not appear in the timeline. Instead, they appear in a section labelled "No Date" to the left of the timeline. If notes do have a date, but that date lies outside the boundaries of the timeline, they appear in a section labelled "Out of Range" to the left of the timeline.

Labels for each band are taken from the container's attribute $TimelineBandLabels, a semicolon-delimited string. For example, the string "Paris;London;Madrid" would label three bands, "Paris", "London", and "Madrid".

Resizing or moving events in the timeline will change their start and/or end dates.

Links can be dragged, from the selected note, in the view pane via the link widget drawn to the right of the selected note's title.

Opening a timeline on a note with no children opens that note's container as this is the more likely intent.

More detail on the Timeline view:

- Adding, deleting or moving events
- Alternative Date attributes
- Colouring the Timeline's main view
- Dragging event notes
- Item styling, markers & duration
- Items undated or outside current date range
- Link visualisation
- Scope of inclusion for notes
- Timeline adornments
- Timeline bands
- Timeline customisations: items vs. view level
- Timeline scale
- Timeline Settings pop-over

Adding, deleting or moving events

Timeline view is not a 'view-only' format. Items shown in the main panel or either of the sidebars (no date, out of range) can be editing and items either added or deleted.

To edit a note (or notes) simply select the item(s). This becomes the scope of the Inspector, quickstamp, stamps, and all other controls. Normal shortcuts, such a spacebar, will open the text window.

Dragging a note horizontally within the main view will update the $StartDate (and $EndDate if one) according to the moved-to position.

Dragging the event's title wider than the default width will alter the duration by updating $EndDate. If no $EndDate was set previously, an appropriate $EndDate will get added and set.

Dragging an event vertically, will move it to a new timeline band (if present) or dragging to the bottom of the current view will create a new band. Note that bands use alternate colour backgrounds to make them easier to tell apart.

Dragging an event vertically *within* a band is not possible. To alter the listing order, open an outline view.

Shift-dragging vertically will ensure the $StartDate does not change, e.g. whilst moving a note into a new band.

Deleting a selected note in the timeline, deletes as normal, i.e. from the TBX and not just the current view.

See dragging undated and out-of-scope items to/from the timeline.

Alternative Date attributes

Sometimes a note may hold significant dates in Date-type attributes other than $StartDate and $EndDate, or may thus may store several pairs of dates.

It is possible to specify an alternate pair of Date-type attributes for plotting timeline events, instead of using the default $StartDate and $EndDate. The desired start and end attribute names are specified via $TimelineStart and $TimelineEnd.

Colouring the Timeline's main view

The main background colour is $MapBackgroundColor, whilst alternate even-numbered timeline bands, where present, will use tints of $TimelineColor. Vertical grid lines indicating major time periods (varies by scale in use) are shown in $TimelineGridColor. If used, timeline band labels are coloured automatically using tints of $TimelineColor for all odd-numbered bands. The even numbered bands (including band zero) use $MapBackgroundColor.

Dragging event notes

Dragging an event within a Timeline view changes the $StartDate to the reflect the date indicated by the new; if $EndDate was also set, that too will update, i.e. the whole duration will move. This effect applies to selections of notes as well as individual notes.

Notes can be dragged from/to the "NO DATE" sidebar. Dragging onto the main timeline sets *both* $StartDate and $EndDate to the same data/time; dragging to the sidebar sets both to 'never'. However, be aware that the "NO DATE" sidebar only shown if there are any descendant notes with no $StartDate value.

Notes can be dragged from, but not into, the "OUT OF RANGE" sidebar. Dragging onto the main timeline sets *both* $StartDate and $EndDate to the same data/time.

Event drags are undo-able.

Dragging note(s) between existing timeline bands will move the note(s) into that band. Holding the Shift key (⇧) down during a vertical drag will ensure dates are not changed.

Dragging note(s) below the bottom existing timeline band will move the note(s) into that band.

Item styling, markers & duration

An item in the timeline is drawn with the left edge of the icon vertically above the exact date and time [sic] of the item's $StartDate. This left edge is drawn as a line in $AccentColor. Alternatively, if an item has both $StartDate and $EndDate duration bar, scaled to the duration between the dates is drawn along the bottom edge of the icon from the left corner, in colour $AccentColor and the upright left-edge line is omitted.

For very expanded scales, it may be desirable to use some action code to set events with no relevant time component to a consistent time so same-day events list vertically in line.

When item(s) are selected a 'marker' line is drawn down from the left edge of each selected item to the scale bar at the bottom of the view. This marker is drawn in a tint of $AccentColor. The $TimelineMarker boolean can be used to make marker lines persistent, i.e. even when items are not selected.

Link visualisation.

If the note is read-only no resize handles appear when the note is selected and the note cannot be moved.

If an item has a $Badge it is displayed at left between the marker line and the $Name. $Name may be styled via normal attributes. The $Name text is drawn in $Color.

The notes are drawn reflecting $Opacity.

Items undated or outside current date range

Items descended from the Timeline View note that have not $StartDate are placed in a column at the left side titled "NO DATE". Notes can be dragged to and from the timeline, in doing so setting or deleting their $StartDate.

If the Timeline note has $TimelineStart and/or $TimelineEnd set then any items in scope but with dates earlier or later than these limits are placed in a separate left-side column called "OUT OF RANGE". Events can be dragged from, but not to this column.

These sidebar columns are only drawn if events of that type are present, otherwise they are not drawn.

If both columns are in use, the "NO DATE" column is drawn first (furthest left).

Within the columns, notes are placed within their respective timeline bands, if bands are in use.

Link visualisation

Any links that both start *and* end at items within the main timeline are drawn.

Unlike maps:

- Link labels are never drawn (regardless of link label visibility settings).
- Links between items at different outline levels will be drawn.

Links ignore timeline bands. The link line simply links across the band(s) as required.

Links are drawn in using $TimelineBandLabelOpacity.

Links are drawn between notes and descendants, and not just siblings as previously. Thus a link is drawn if source and destination of the link are both within the main timeline (i.e. excluding sidebar listings).

Scope of inclusion for notes

Opening a Timeline view plots data of all descendants that have a $StartDate defined. For instance, if a container has child containers—without dates set—titled to represent years, and these contain notes with $StartDates then the view will show the events for all years whilst the year containers will be listed in the 'No Date' sidebar. This is a powerful feature as the non-plotted year containers can help with data/note management without affecting the actual timeline visualisation. If a note also has an $EndDate the note will plot as having a duration in the timeline. An event's $DueDate is not acknowledged by timelines.

It is possible to specify different non-default Date-type attributes for plotting events in a timeline.

It is not possible to define any other attribute than $StartDate as the inclusion criteria for the timeline, or to use numerical or other sequence data (as opposed to Date-type data).

Items without dates are placed in a " No Date" column at the side of the view.

Remember Tinderbox's built-in prototypes as these include an ' Event' prototype that is ideal for timeline data items.

By default, the timeline will encompass all dates in scope but it is possible to set $TimelineStart and/or $TimelineEnd in the container note to set a specific duration within which data is displayed. If these boundaries are set, outlier data is placed in a "Out of Range" column at the left side of the view (and to the right of a 'No Date' column, if present).

Dates with an $EndDate only or where $EndDate is before $StartDate will plot, but incorrectly, as such configurations are effectively invalid.

By default, timelines show all descendant notes and in-scope aliases with qualifying dates. Either of these categories can be excluded via $TimelineDescendants and $TimelineAliases being set to the timeline's container note.

Timeline adornments

Timelines can display adornments, if the adornment has both a $StartDate and $EndDate or both dates as defined by $TimelineStart and $TimelineEnd. These adornments are in fact shared with the map view of the same container, and use their colour, border, etc., values. The 'smart' elements of any such shared adornments (querying, etc.) only work in map view and so are ignored in timeline view. Adornments in maps of child/descendant containers are *not* shown. Thus a given map will always show all its adornments but the timeline of the same item will only show adornments with start/end dates.

Adornments are drawn from the top of the timeline view to the bottom of the bottom timeline band. Adornment staking order (if overlapping) is as per map view), though this can be still be adjusted in timeline view using the same controls as for map view.

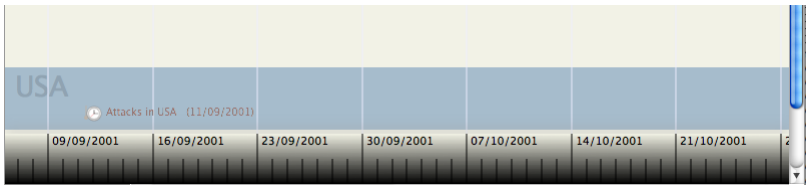
Adornments draw in their map colour (i.e. $Color) and draw vertically across all bands on top of any band colour. The width of the adornment is controlled by its start/end dates as with notes.

Dragging a selected adornment right or left will move it and adjust $StartDate/$EndDate. Dragging the left of right drag-handle of a selected adornment allow separate adjustment of $StartDate/$EndDate.

As the Create Adornment dialog has no way to set $StartDate or $EndDate, it is advisable to create new timeline adornments in the map view of the timeline's container, even if said map view is not used for any other purpose.

Although the timeline view context menu allows you to add an adornment, it is better to create adornments in map view, then use their Info view to set $StartDate and $EndDate. Setting the latter will then make the adornment show in timeline view.

Timeline bands



By default, all events are shown in a single band. However, notes may be dragged down to create a second, or even more bands, by dragging to the bottom of the main view area. The timeline band to which an (event) note belongs is stored in $TimelineBand, noting that bands number from zero, not 1.

Timeline bands may have a label, drawn at left end of the band, that is stored as a list in the $TimelineBandLabels attribute of the container holding the timeline. In the simplest scenario, of a container holding event children (and not grandchildren), $TimelineBandLabels is set in the container and $TimelineBand in the children. In other words, the band labels are set in the parent or some other ancestor of the events being shown and never in the events themselves (unless an event note is itself used as a source of a timeline view. Using excessively high values for $TimelineBand (as might occur through data entry errors) may result in timeline band labels being suppressed due to the sheer number of bands to be drawn. In the latter event, query for the largest $TimelineBand value in use and adjust it if necessary.

Bands are drawn down as far as the lowest with data or lowest named label defined. Empty bands are drawn at a fixed height as long as there is at least one higher-numbered band (i.e. drawn lower down in the view) with data. This behaviour makes it easier to show data like lecture theatre or operating room scheduling, where it is expected that at times a given band (e.g. given room) may have no events (bookings). Empty bands below the last band with data are not drawn, to reduce the vertical height of the view.

All even-numbered bands, with numbers starting from *zero*, are drawn in $MapBackgroundColor; all odd-numbered bands are drawn in a tint of $TimelineColor. For a given view these colours are those of the view's container, i.e. the same place (note) as with $TimelineBands described above.

$TimelineBandLabelColor and $TimelineBandLabelOpacity allow control over the colour and opacity of timeline band labels.

$TimelineBand is intrinsic for aliases, allowing aliases to be placed in a different timeline band to their original.

Timeline customisations: items vs. view level

Some aspects of the timeline view are altered per item and some in the container used as the view source.

Item (note) level:

- $EndDate
- $StartDate
- $TimelineBand
- $TimelineMarker

Container (view) level:

- $TimelineAliases
- $TimelineBandLabelColor
- $TimelineBandLabelOpacity
- $TimelineBandLabels
- $TimelineColor
- $TimelineDescendants
- $TimelineEnd
- $TimelineEndAttribute
- $TimelineGridColor
- $TimelineScaleColor
- $TimelineScaleColor2
- $TimelineStart
- $TimelineStartAttribute

Adornments: these use map view colour, border, etc., but smart (query) elements of adornments are ignored. Where they overlap, adornments follow map view stacking order, though this can still be adjusted in timeline view using the same controls as for map view.

Overall background colour is the container's $MapBackgroundColor, and thus is common to the container view across all major view types. The optional alternate colouring & pattern settings used for maps are ignored in timeline view.

Any aliases in scope, i.e. descended from the container (unless aliases and/or descendant display is disabled), are drawn like normal notes except that an italic font is used for the title. The note $Name/$DisplayName is drawn in $Color and the marker in $AccentColor with the marker line to the scale using a tint of $AccentColor. Timeline view uses display expressions for items but does not show their subtitles or hover expressions.

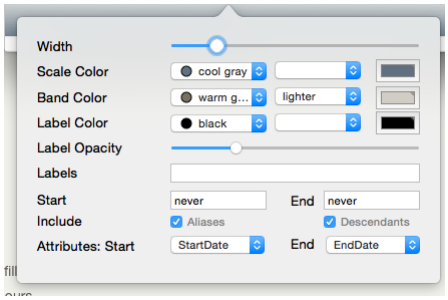
Timeline scale

At the bottom of the view is a scale bar whose graduation at labelling will vary as the view scale is changed using the scale scrubber control at top left of the view. The bar uses a graduated colour fill from $TimelineScaleColor at the bottom $TimelineScaleColor2 at the top. The latter defaults to the same colour as $MapBackgroundColor to give a smooth transition into the main view. Thus if the latter is changed, it is probably a good idea to set $TimelineScaleColor2 to the same value.

The periodicity and exact scale bar positioning of scale captions are set by Tinderbox and vary with the scale as set via the scrubber. The user cannot modify these labels.

From every major scale marker (normally those with labels) a vertical grid line is drawn vertically up across the timeline, in $TimelineGridColor.

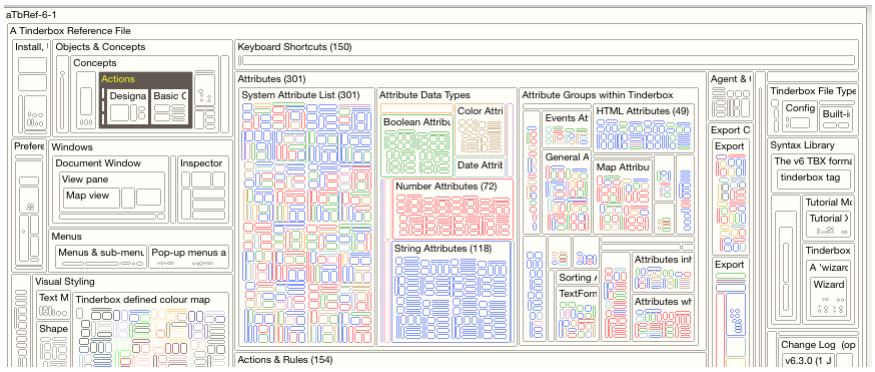
Timeline Settings pop-over



This pop-over gives a visual method for setting a number of timeline-related settings for the current Timeline view:

- **Width**. This adjusts the scale of the bottom scale, i.e. how much horizontal distance there is between days or months, etc.
- **Scale Color**. A trio of standard colour controls set the main colour used in the scale bar at the bottom of the timeline. $TimelineScaleColor.
- **Band Color**. A trio of standard colour controls set the colour a tint of which is used to draw alternate timeline bands ($TimelineColor). The actual colour set is a 'lighter' shade of the colour set in the controls.
- **Label Color**. A trio of standard colour controls set the colour used to draw timeline band labels ($TimelineBandLabelColor).
- **Label Opacity**. Sets the opacity with which timeline band labels are drawn ($TimelineBandLabelOpacity).
- **Labels**. Sets the labels used for timeline bands ($TimelineBandLabels).
- **Start**. Sets $TimelineStart.
- **End**. Sets $TimelineEnd.
- **Include**:
  - **Aliases**. Sets $TimelineAliases.
  - **Descendants**. Sets $TimelineDescendants.
- **Attributes**:
  - **Start**. Choose and alternative Date-type attribute for plotting event start instead of $StartDate.
  - **End**. Choose and alternative Date-type attribute for plotting event end instead of $EndDate.

Treemap view



Treemap view enables more notes to be visible even in comparatively small spaces and to reveal quantitative relationships between them.

Treemap Properties may be adjusted by clicking the "i" button in the Treemap tab. These controls weighting of items such that the area used to draw the items represents some value. By default no weighting is applied and all sibling notes draw at the same area within their parent; note their shape may vary to allow the map algorithm to fit everything within the parent. In addition notes are drawn with a border in $Color and a fill of the Treemap 'start' colour.

As well as an expression to set weighting, the Treemap Properties also allow a separate weighting to be set such that the fill of an item uses a a shade between customisable start/end colours. A custom border colour can also be set to override use of $Color.

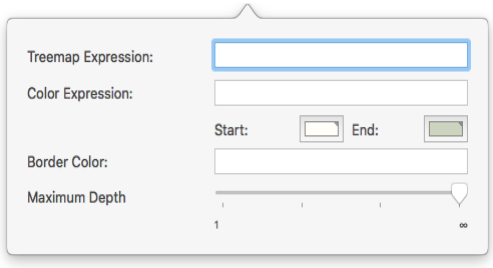
The Actions and Dashboards tutorial PDF contains a useful section on configuring treemap view.

Treemaps do not support the dragging of links from the view pane. Use the link widget on the text pane to drag links form the selected note.

Links can be dragged from the link parking spaces onto notes in the treemap to create links. To create a link from a treemap note, use the text pane link tool making sure there is no $Text selected; this creates a basic link from the selected note.

- Treemap Settings pop-over

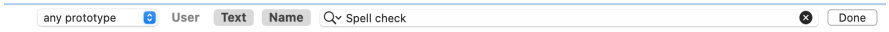
Treemap Settings pop-over



This is the configuration pop-up for the current Treemap view. Note that none of these setting are saved as attributes but the are saved as part of the Tab's data.

- **Treemap Expression**. Enter an attribute name or action code expression. Used to derive a number used to set each item's area. Items with a value of zero are not drawn at all. Non-Number attributes return a value of 1 if they have a set/inherited value: i.e. string-based are not empty, booleans are `true`. The default is no no expression code.
- **Color Expression**. An action code expression use to colour each note. The colour used is a shade between the start/end colours set below. The minimum value uses the start colour, the maximum the end colour with other values using a shade between the two. The default is no no expression code.
- **Start**. The starting colour used of shading treemap items. The colour chip opens a standard OS colour palette. The default is an off-white #FFFEF9.
- **End**. The ending colour used of shading treemap items. The colour chip opens a standard OS colour palette. The default is a muted light green #CCD4C0.
- **Border Color**. Enter the Tinderbox colour (named or hex value) used to draw the border of each item. Or use an expression resolving to a named or hex colour value. If not specified, items draw with a border of $Color. The default is no value. (i.e. it uses $Color).
- **Maximum Depth**. Allows display of only the **N** upper levels of the treemap. This can sometimes improve clarity of presentation.

Find toolbar (view pane)



Toggling on the Find process exposes a toolbar at the top right of the view pane. Three independent buttons allow setting scope to **User** and/or **Text** ($Text) and/or **Name** (note's title: $Name), or both. By default only Text and Name are pre-selected.

Click the **User** button to turn on user attribute search, and select the attribute of interest from the user attribute table. Dismiss the popover and the label of the "User" button becomes that of the chosen attribute and operates as a toggle like the other two buttons.

Double-clicking the chosen attribute's button is a convenient shortcut to searching for the currently entered find string in that attribute.

As only one attribute selection is allowed, selecting a new user attribute will replace the existing choice. The feature is reset to 'User' at the beginning if each session.

From v9.5.0, a new prototype-based filter control is offered. The pop-up list displays all the document's currently defined prototypes. Selecting a prototype, filters the find results to only items using the prototype.

Enter a search term in the box and press Return to see a pop-up list of matches. Click the 'x' button in the input box to clear the search. Click on the magnifying glass icon to see a pop-up list of previous searches (up to 10 are stored).

The search term box's pop-up menu offers an option for case-sensitive search, and an option to turn off regular expression search.

The Find toolbar appears after you press ⌘-F with focus in the the view pane. If the Find toolbar is already visible, ⌘-F moves the keyboard focus to the Find toolbar's search field. [Esc] dismisses the Find toolbar.

Find offers suggested related words as possible autocompletions in the Find bar and in torn-off Find windows. These are available only in macOS 10.14 and later, and only in selected languages.

To close the Find bar click the **Done** button.

View filter toolbar

This feature is currently only implemented in Outline view

The view filter toolbar is hidden by default, but can be opened from the View menu, or via a shortcut.

Filter expressions are queries, and are saved with the tab.

When the filter bar is active, outlines do not draw collapse widgets and all containers are implicitly expanded. Closing the filter bar disables the filter and restores the previous expansion state

The filter list is alphabetised.

The filter bar displays the number of items that match the filter's query. Note that this is not identical to the number of items displayed in the outline, since an item appears in a filtered outline if it matches the query or if one of its descendants matches the query.

The Search field of the find bar grows larger if the window is sufficiently wide.

Autocomplete suggestions. The Find Bar no longer displays a menu of related words as autocomplete suggestions, instead such suggestions may now be requested by pressing F5.

Text pane

The Text pane forms the right side the right side of any document window tab, i.e. everything to the right of the vertical spitter bar. By default the pane only shows the Text tab of the text pane (more on the content of the Text tab).

In fact, the Text pane has three tabs: Text, Preview and HTML, with the latter two hidden by default, as mentioned above.

As many Tinderbox projects do not need to export HTML (or any kind of marked-up text) at all, the selectors for these three tabs are hidden by default. To reveal these tab controls Use the Window menu ▸ Shown/Hide Text Pane Selector will toggle the visibility of all sub-tabs.

From v9.5.0, the Text Pane Selector is now visible unless the user hides it; previously, it was hidden until you showed it. This change reflects increasing use of the preview tab as a presentation mode whereas originally is was designed for checking the render before HTML export, and thus on no need for those who do not use HTML export.

When first opening the document, if the text pane selector is initially hidden, hiding it is deferred slightly to allow seeing it sliding away, helping to confirm that it is deliberately hidden (and suggesting that there is a way to recover it).

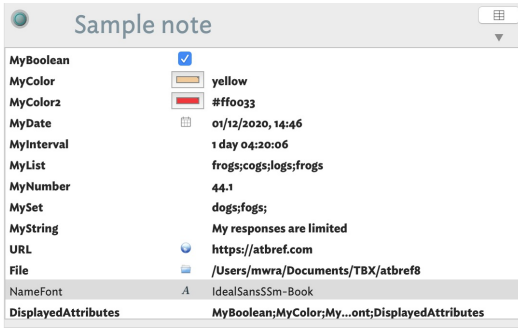
Printing is possible from all 3 sub-panes (the Preview pane from v9.5.0).

In all three tabs, the text pane's title offers a tooltip containing the path of the note.

The sub-tabs are:

- Text tab
- Preview tab
- Export tab

Text tab



This is the default tab of the Text pane and shows the selected note's $Text and, optionally, its Displayed Attributes. By default the text area is empty and there is no Displayed Attributes table displayed.

Title bar

The top grey section of the tab contains a number of controls:

- **Link Widget**. Displays a white 'T' when there is a selection of $Text. See more on the use of the link widget.
- **Error warning** indicator. *Shown only if errors are found* . Click to display the Error list. See more on the error list. See more on the error list. The error flag has a text title to make the meaning more obvious.
- **Title**. The note title ($Name). The string is always $Name and not $DisplayName such as will be seen in the View pane at left. The title is drawn in $Color using $NameFont. The title is always editable as soon as the control takes focus. Using the Return key to complete the edit will close title edit mode and set focus in the View (left) window pane. The title is rendered in $Color unless the colour is "transparent" in which case black is used. Notes using lighter colours for $Color *may* render the title in black as opposed to $Color in order to give sufficient contrast. The colour used to render the title obeys the Document Settings, Outlines tab's for darker colours or always using black or white titles .
- **Add Displayed Attributes** button. This opens the Add Displayed Attributes pop-over, allowing addition and editing of the Displayed Attributes table.
- (**disclosure triangle**). This can be clicked to show/hide the Displayed Attributes table. The triangle is not shown if the current note has no Displayed Attributes configured. The show/hide state is stored in $HideDisplayedAttributes.

If multiple items are selected in the View pane the lower part of the text pane, below the title bar, is replaced with a grey fill and a caption showing the number of notes selected.

Displayed Attributes

This table is only shown if Displayed Attributes ($DisplayedAttributes) are set for the current note. The table's listing and wording or listing is controlled via the Add Displayed Attributes pop-over accessed from the button in the title bar.

See more detail on configuration and use of the Displayed Attributes table.

Text area

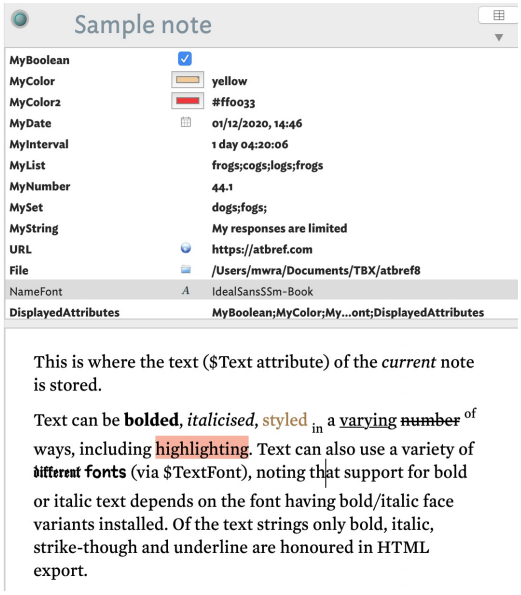
The Text area is the main section of the tab and holds the text ($Text) of the note. The text is RTF text, though note that not all RTF features export to HTML. A ruler (tab bar) for the text area can optionally be toggled on/off (see the Text sub-menu of the Format menu).

Editing text updates the current note's rule and edict (if present).

More on the Text pane:

- Displayed Attributes table
- Text pane multi-item view selection
- Text pane editable multi-select
- Text area
- Text area - Links panel
- Text area ruler
- Find toolbar (text pane)
- Find results toolbar (text pane)
- Find & Replace toolbar (text pane)

Displayed Attributes table



NOTE: 'Displayed Attributes' replace the old now-deprecated 'Key Attributes'—see explanation.

A note's 'Displayed Attributes' are an optional display of a user-defined tabular layout of attribute title/value cells. A Displayed Attribute is a display feature and *has no effect on inheritance and does not accord an attribute any special status* other than being easily viewed/edited for a note's text pane.

Attribute names are shown in the left column, attribute values in the right column. The width of the first column of the Displayed Attribute table adjusts itself to accommodate long attribute names. The list can be re-ordered, items deleted and attribute value edited. An item is added by:

- using the 'Add Displayed Attributes' button in the note's text pane's title bar (button, to right of the title), and then using the Add Displayed Attributes pop-over the contents and order of the table; pre-v8.9.0 the button icon was a '+' sign. As well adding/removing existing attributes, new attributes names can be typed here and on dismissing the pop-up the user will be asked to confirm this is a new attribute and its data type (default:string)
- setting the note's $DisplayedAttributes.
- setting the note's prototype's Displayed Attributes (if a prototype is used)

The vertical order of displayed (key) attributes can be altered via drag-drop of the attribute name label (left column cell), or altering the order of the list stored in $DisplayedAttributes. An item can be deleted from the Displayed Attribute display by dragging its label cell out of the table and dropping onto the $Text pane.

Row height. From v9.5.2, String-type (only) attributes can have a bigger line height. The default row hight for String-type attributes remains one line, but this can be extended to a maximum of seven lines. Within that space line breaks in the String's value are honoured but any overflow is clipped (i.e. more content cannot be scrolled). Line height is set per-attribute using The Document Inspector's System and User tabs. From v9.6.0, multi-line attributes are also permitted for Sets and Lists as well as String type attributes.

Items in bold are set locally for the note. Items in lighter colour are read-only (normally dynamically calculated values like $ChildCount; pre v6 these were in italics). All other rows in normal text use the default or inherited value. The full meaning of the styling of different rows (bold, strikethrough, etc.) is explained here.

This pop-up can be called via the View menu as well as from the text pane.

At the extreme the right of the panel each string, List or Set attribute has a pop-up list of values, allowing quick setting of values already in use for that attribute (or suggested values, if defined).

When a user attribute is renamed, Displayed Attributes referring to the new attribute are updated to use the new attribute (and attribute values are transferred).

Boolean-type Displayed Attributes are always shown as a tick-box; ticked equates to `true`, un-ticked to `false`.

Number-type data. Very large or small numbers can be entered/shown using exponential form, e.g. `0.000001` can be entered as `1e-06`.

Color-type data. The data is shown both as a colour chip (showing the currently set colour) and a string showing the value set with is either a Tinderbox named colour or otherwise a hexadecimal value string ("#330099").

Date-type data. Shows the data and time, with the date/time format in the host OS' short date format, so this will vary by the user's locale and their choice of OS settings. Document Settings offer a small selection of variations on the OS format. An alternative date-time string format can be used for all Date-type attributes in KA by setting $DisplayedAttributeDateFormat at document or note level using a date format string. On entering a date, if no time is specified, the time element of the attribute value defaults to current system time. A date-picker control is offered to the left of the value box. When editing dates in the Displayed Attributes table, from v8.2.3 the date string is changed before editing to its value in normal format (medium date, short time). This avoids ambiguity in short dates, where the default US style uses 2-digit years: i.e. does 12/7/41 represent a date in 1941 or 2041?

String-type data. Although generally intended to hold short text values, String attributes can hold larger amounts of text and including line/paragraph breaks. When not selected for edit, all text paragraphs run together. In edit more, the value edit box will only show the first paragraph of text. Use the right arrow to reach first paragraph text overflowing the edit box; use the down arrow key to access paragraph #2, etc. It is not possible to type a line break character in the edit box bu it is possible to copy/paste one into it.

URL-type data. A form of string attribute, URL attributes show a globe icon to the left of the data value edit box. Clicking the button will open the URL, stored in the value box, in the user's default web browser. Files can be dropped onto a URL-type attribute (e.g. to give a file:/// type of URL). The URL button remains enabled even if the note is read-only.

File-type data. A form of string attribute, File attributes show a folder icon to the left of the data value edit box. Clicking the button will open the file (or folder) path set in the value via Finder. Folders will open in Finder and other file formats in whichever app is registered as the default app for that file type. The File button remains enabled even if the note is read-only. The Browse File… menu option is not available for read-only notes.

Font-type data. Font-type attributes (which are only ever system attributes) show an 'A' button which opens the OS' Font dialog, making it easier to set the correct font name for a font. N.B. altering the text font size does not alter the font size attribute (where pertinent) in Tinderbox.

List-type and Set-type data. All discrete list values are shown, as a single string of semicolon-delimited values.

Email-type data. From v9.6.0, Email-type attributes such as $Email, show an icon which allow the creation of email from inside Tinderbox. If a note has no $EmailTemplate, the 'email' icon/button in the Displayed Attributes table sets the body of the new email to the styled text of the note (i.e. ^text^).

A note's title ($Name) can be edited directly via the title bar above the Displayed Attributes table so there is no need to add it to the table.

Displayed Attributes are sometimes referred to with the shorthand abbreviation 'DA' in help forums, etc.

The attribute $ReadOnly is exempted from being read-only, even when set to `true`. This allows you to turn off ReadOnly from the displayed attributes table or from Get Info ▸ attributes tab.

Show/Hide Displayed Attributes

A disclosure triangle allows the visibility Displayed Attributes table to be toggled, e.g. to allow more space for showing $Text. The control is not shown if there are no Displayed Attributes for the current note. The 'add Displayed Attributes' button still remains available. The show/hide state is stored in $HideDisplayedAttributes.

Cycling/Editing Displayed Attributes via keyboard

To cycle through Displayed Attributes, use the **Return** key (↩). Using the **Return** key on a selected row sets that item in edit mode. Clicking **Return** again closes the edit and selects the next item in the table (looping to the top when the bottom is reached). When a boolean item is selected and in edit mode, use the **Spacebar** key to toggle the `true/false` setting, otherwise other data types are editing via normal keyboard input.

If the Return key has not yet been used since the note was selected, the **Up-Arrow** (↑) and **Down-Arrow** (↓)keys can be used to move the selection in the table (but the selection does not loop at top bottom). If **Return** key has been used t **Up**/**Down** arrow keys simply select the table row above or below and set it in edit mode.

To cycle between the Displayed Attributes table, the $Text are and the View pane, use the **Opt**+**Tab** keys (⌥+ ⇥).

In Outline/Chart/Treemap view, to move the View pane selection (i.e. change the contents of the text pane) up or down one item in $OutlineOrder use **Cmd**+**Opt**+**Up-Arrow** (⌘+⌥+↑) or **Cmd**+**Opt**+**Down-Arrow** (⌘+⌥+↓).

Editing Displayed Attributes values

Once a row in the table has focus (either via keyboard, as above)or by a direct trackpad/mouse-click the value box in the right column can be edited:

- Enter edit mode. Either double-click (mouse/trackpad) or use the **Return** key (keyboard). The value box contents can be edited. To tick/un-tick a Boolean attribute via the keyboard use the spacebar. For data types offering it, the pop-u value list of existing/suggested values (see above) can be used to set a new value without typing.
- Complete the edit and save the new value. Click out of the table cell (mouse/trackpad) or use the **Return** key to toggle out of edit mode.
- Use the **Escape** key (⎋) to abandon an incorrect edit.

Displayed Attribute characteristics such as enabling or disabling the URL and File buttons are updated after editing the Displayed Attribute's textual value.

Changing the text of a Displayed Attribute immediately updates that Displayed Attribute's pulldown value menu.

When a Boolean attribute is selected in the Displayed Attributes table, pressing the spacebar will toggle its value.

The table displays an ellipsis (…) if the text value extends to multiple lines, even if the visible line is not truncated.

The menu commands Format ▸ Text ▸ Insert Date/Time/Date and Time now insert the requested text at the insertion point, rather than replacing the current value with the requested text.

Number of Displayed Attributes shown

The height of the Displayed Attributes table can grow to as much as half the height of the text panel, reducing the need to scroll long lists of Displayed Attributes if using a sufficiently-large display. As previously, the table shows scroll bars if not all Displayed Attributes can be accommodated. In older versions, scroll bars show if there are over 15 Displayed Attributes, regardless of text pane height.

Resetting default/inherited values

For displayed String/Set/List data typed attributes only, the default value can be reset by using their pop-up value lists, by selecting the value 'normal'. For all other data types, to repair inheritance select the note in question and do either on of:

- click on the line with the attribute to reset, right click and use the 'Use Inherited Value' option in the pop-up menu.
- open the Quickstamp Inspector, select the desired attribute and use the reset buttons.

Font/Font Size for the Displayed Attributes table

The Displayed Attributes table can be set to a user specified font, for attribute names and their values, via $DisplayedAttributesFont.

The font size used to render the current note's table can be altered via the Displayed Attributes replace Displayed Attributes sub-menu of the Window menu. The latter offers a small range of suggested sizes (default is 11pt). The actual size is stored is the point size number, i.e. 11 for the default in $DisplayedAttributesFontSize. It is possible to set values not available in the menu by editing the attribute value directly. To change the value for the entire document, use the Document Inspector General tab to alter the the default value for $DisplayedAttributesFontSize.

Evaluating an attribute

The current value of the Displayed Attributes can be evaluated (as action code) and written back as the attribute's value. The Evaluate function is accessed via the table's pop-up context menu.

Changes to Displayed Attributes affect $Modified

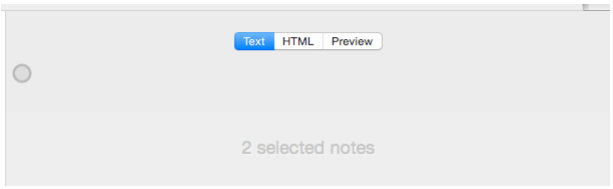
Changing an attribute value in the displayed attributes table or in Get Info's attributes tab updates $Modified. Changing an attribute value in a stamp (including Quickstamp) or an action does not update $Modified.

Needing to show even more Displayed Attributes?

The maximum number of table items equates to c.50% of the $Text area's height, so dragging a taller overall document window can help. Other approaches to consider:

- set Displayed Attributes via prototypes. That means that it is easy in such notes to add new display items and delete others, e.g. to debug and immediate problem and easily reset them ('reset' button on the Displayed Attributes configuration pop-up) to the prototype-inherited list. Forget to reset at the time? The reset button still works at a later time. Or, simply make and agent to find all notes using that prototype and hasLocalValue(), e.g. `hasLocalValue("DisplayedAttributes")`. Then reset $DisplayedAttributes via the agent action.
- Have the prototype store one or more alternative lists of attributes, e.g. normal set, a debug set, a set for a specific process, etc.

Text pane multi-item view selection



When there are multiple items selected in the view pane, the note title is replaced—as on the Inspector—with a count of the selection size.

Text pane editable multi-select

From v9.5.0, if several notes are selected at once, they may now be viewed *and* edited. It is possible to edit any of the selected notes in the text pane.

Editing is permitted inside any note, but not in selections that contain more than one note. Nonetheless, the text format may be modified across note boundaries, facilitating changes in font or type size across an extended document.

Text area

This is the main working part of a note and where is body copy ($Text) is edited and displayed. The text is RTF and supports RTF styles. The available text area customisations and controls are a sub-set of those that will be familiar from Apple's TextEdit app.

Altering the look of note text

A number of text area functions are controlled by/stored as attributes. These are system attributes mostly in the TextFormat and Textual groups. It is important to note that all text is styled. Thus these note-scoped attribute values controlling the text font and size have no further effect once the text area is first edited. However, the Format ▸ Style submenu has option for resetting selected text back to the attribute stored values.

To set non-inherited font face (the 'name' of the font) or size or colour use the OS Fonts dialog. Styling like bold, italic, or underline use normal macOS shortcuts and can also be accessed via the Format menus, as can things like sub-/super script, and enabling rulers and tab-bars. Colour and highlights colour values may also be set via attributes.

Some further options are available via the right-click context menu with options varying depending on wether there is text selected or no selection.

The Text Inspector's 'Text' tab controls a number of *note-level* text settings so in not the place to look for in-text style controls.

Displaying note text in map view icons

This aspect of text display is described under Map view, see here.

Text Selection

Text selection methods:

- Select the adjacent line: click in the left margin of the text pane.
- Select the adjacent paragraph: double-click in the left margin of the text pane.
- Extend the existing selection: shift-clicking and shift-double-clicking extends the current selection to encompass the adjacent line or paragraph.

Rules and Edicts are updated when the text of a note is edited.

Find & Replace

The text area has its own discrete Find and Replace features including highlighting of Find results. A ruler can be displayed for the text area.

Exporting Styled Text

Though only a subset of the full range of RTF styling is supported for HTML export, fully styled RTF text export is supported via text export (note the latter only supports a limited range of export codes).

Text Font, Size and Colour

When $Text is first edited it uses its inherited of set $TextFont, $TextFontSize and $TextColor, i.e. the defaults. Importantly, this results in RTF text saved using those values. As a result, changing those defaults *has no effect on existing text* the $Text. To restyle (selections of) the existing $Text or to re-impose the defaults see the Format ▸ Style or Format ▸ Font menus.

Avoid Importing Source Text Style

When importing test, to avoid source styles carrying across (e.g. copying from a web page) rather than use the **Paste** (⌘+V) command use **Paste and Match Style** (⌘+⌥+⇧+V). This will retain bold and italic styles but lose font type/size/colour from form source.

Thus when importing test from outside Tinderbox it is generally best to use **Paste and Match Style** unless it is certain that external formatting is needed.

Action Code and Styled Text

Several string operations can be applied to styled text from $Text, or to text destined for $Text, in order to modify the (rich) text styling. Note that while any string may have its style set, this styling only has an effect when viewed in $Text.

When an action assigns the $Text of one note to another, Tinderbox preserves styling.

```
$Text = $Text("/configuration/example")
```

When the text of two notes are appended, Tinderbox preserves styling.

```
$Text = $Text(/a)+$Text(/b)
```

A string may be emboldened or italicised:

```
$Text=$DisplayName.bold
$Text=$MyString.italic
```

The font size of a string may be changed:

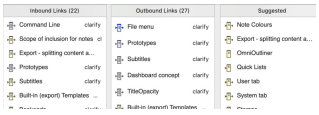
```
$Text=$MyString.fontSize(36)
```

The following operators also respect style information:

- String.replace()
- String.substr()
- String.paragraphs()

Text area - Links panel

A collapsible Links panel has been added to the the bottom of the text pane. When invoked it slides up from the bottom of the $Text area. To invoke the panel choose menu Window ▸ Links (or use the keyboard shortcut, ⌘+7). Using either method, the call is a toggle that shows or hides the pane depending on its current state.



Lists are empty when the pane is displayed but more than one note is selected.

The panel shows three lists. These display for the current note, in order left to right:

- inbound links
- outbound links (excluding web links)
- suggested link targets.

Inbound and Outbound links

The panel shows the source or destination display name and the link type. For compactness, the link anchor appears in the tooltip when hovering the cursor over the link.

When the panel is visible, double-click any inbound or outbound link to follow it and select its source and reveal the link's source (or destination) in the current view (scrolling the view if necessary). Selecting a link and pressing the Delete key (⌫) will delete that link.

Suggested links

These are notes to which Tinderbox assesses the user might usefully create new links. At present, are based exclusively on similarity though this will change as the underlying tests are improved (in app, not in the current document session). The underlying assessment is that used for the Get Info/similar dialog.

To make a text link to a suggested note, drag a link from a parking space to the note in the suggestions list. If text is selected in the text pane, the link with be a text link. If no text is selected, the link will be a basic link. Double-clicking a note i the suggestions list inserts a new text link, using the note's name as a link anchor.

The Suggested links list slides offscreen if there is insufficient space.

From, v9.6.0, suggested links exclude notes in the Hints container.

Linked item previews

Selecting a note from the inbound list in the Links pane previews the *source* ($Text only) of that link. Likewise for selecting a note from the outbound link list previews the *destination* of that note. In other words regardless of link direction, the preview is of text from the other end of the link. Selecting a note in the suggestions list displays a preview of the suggested linked note's title and text in the view pane.

The text preview window has a label at its foot that explains how to dismiss the window or drag it to create a new text window. The hint panel in the link's text preview pane slides out of view after three seconds. Moving the mouse over the link's text preview displays the hint again, as does previewing a different note.

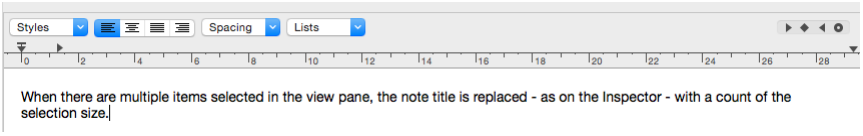
From v9.6.0, right-click on any item in the Inbound or Outbound lists for a contextual menu, allowing you to select that note.

From v9.6.0, when double-clicking a note in the Links drawer to select it, Tinderbox will expand any collapsed ancestors of that note in outline view.

Tear-off previews. Dragging the text preview, creates a new text window for the note.

Dismissing previews. Click the preview, the text pane, or the view pane to dismiss the preview. Pressing the [Esc] key will also cancel the links pane text preview. The text preview window has a label at its foot that explains how to dismiss th window or drag it to create a new text window. The hint panel in the link's text preview pane slides out of view after three seconds. Moving the mouse over the link's text preview displays the hint again, as does previewing a different note.

Text area ruler



A ruler can be displayed for the text area showing margins and tab stops and also including a set of RTF style controls.

The ruler is hidden by default at the start of every session and turned on/off globally for the document; i.e. if shown they are shown for every note. The ruler is shown either via the Format ▸ Text sub-menu or a shortcut.

The ruler allows the setting/editing of margins, hanging indents and tab stops.

Controls above the ruler allow the setting of stored styles. These apply at paragraph rather than note level, so are best applied to text selection. There are further controls for text alignment, line spacing, and list styling.

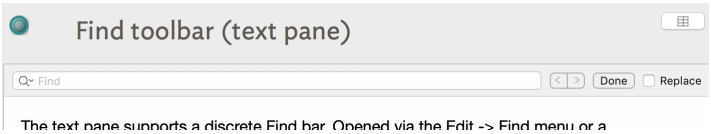
Styling of lists and other ruler-set styles

The Tinderbox area (i.e. where $Text appears) uses Apple's Rich Text engine. As result is that built-in styles use a default font of Helvetica 16pt. Although a note has a $TextFont (and $TextFontSize) is is not applied to $Text until a given note's $Text is first edited. At that point the current $Text font becomes the 'base' font for that note's text.

If inserting styles from the ruler, such as a list, and $Text has not yet been edited, the style may apply its (Helvetica) default, as $TextFont is not yet set in the stored RTFD data. If this occurs, either use the Format ▸ Style menu's, Standard Font command (**^⌥⌘+T**) to reset to font. Or, before using any style in a new note, type a few characters in the note to 'set' the base font (the characters can then be deleted). Thus:

- Make a new note, place the cursor in $Text and then add a numerical list. The font of the list numbering will be Helvetica (the stored format). *Not desired!* So,
- Do the same but *first* add at last one character before adding the list and the font used is $TextFont.

Find toolbar (text pane)



The text pane supports a discrete Find bar. Opened via the Edit ▸ Find menu or a shortcut, this adds a one-row toolbar to the top of the text pane (and beneath any Displayed Attributes). If Find & Replace is required, the bar expands to 2 ro

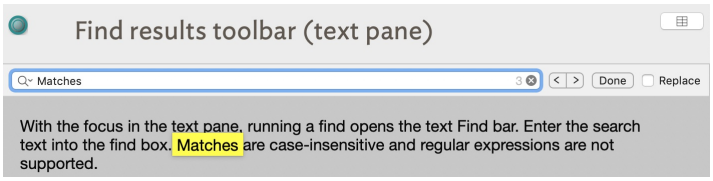
Matches to the find string are case-insensitive and regular expressions are not supported.

The results if a find in $Text will highlight all matches in the current text and lists the total count of matches (if any) at the right-hand end of the search string input box. The two chevron button aloo the cursor to be cycled through the matches in the $Text.

Ticking the Replace box toggles additional find & replace controls.

The **Done** button closes (hides) the Find toolbar. Note that the current find string is remembered during the current session so can easily be re-used.

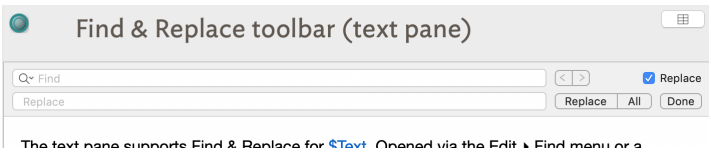
Find results toolbar (text pane)



With the focus in the text pane, running a find opens the text Find bar. Enter the search text into the find box. Matches are case-insensitive and regular expressions are not supported.

Once a Find has been run on on the text ($Text), the Find input box shows a match count at its right-hand end. The $Text is greyed over and all matches are highlighted. The chevron buttons to the right of the Find search box can be used to cycle through each of the matches in turn.

Find & Replace toolbar (text pane)



The text pane supports Find & Replace for $Text. Opened via the Edit ▸ Find menu or a shortcut, this adds a two-row toolbar to the top pf the text pane (and beneath any Displayed Attributes).

Existing text Find controls can be used to cycle through the current note's Find matches and replace them (**Replace** button) with the Replace text. Or, use the **All** button to replace all current matches.

In the Text pane, Edit ▸ Find ▸ Use Selection for Find loads the find string for both the text pane search bar and the view pane search bar.

Preview tab

The preview tab gives a live web-browser style preview of the current note as it will appear when exported. If the export code uses style sheets, images, etc., these will work in preview if such assets are already exported/stored in the appropriate location within the document's export folder.

The Preview tab can also be used simply for viewing a note in rendered form even if there is no intent to export, e.g. when using Markdown in the actual note $Text.

The Preview pane has been rewritten for v9.5.0 and modernised (to use a WebKit-based control). Note that, while in the past Preview did not write to disk, it now writes temporary files. These files are written to the documents (HTML) export location (as viewed/set via the Export Inspector). Also:

- $Text-embedded images appear in previews.
- Images are automatically exported when a note with images is previewed.
- JavaScript is enabled in the Preview pane.

Export tab

This alternative view for the Text pane shows the current note's data as marked up for HTML Export. Although 'HTML' is described, this method of export can be adapted for use with other favours of markup.

Different colours represent different sources:

- Blue: text, i.e. $Text via ^text^.
- Red: includes of children's data.
- Blue-green: includes of data from other notes, e.g. via ^include()^.
- Grey: code from (export) template.
- Black: anything else.

For some bespoke mark-up tasks it is not necessary to do a full page export; simply swap in the desired export template and select/copy the code from the HTML tab.

The code in this view is rendered in $HTMLFont at $HTMLFontSize. These attribute can be set by invoking the OS Fonts dialog via the view's context menu and setting the font typeface and/or size.

If the document currently has no HTML template set, this sub-tab will show the Template pop-up menu to allow selection of an export template to use (assuming one or more templates are defined in the document).

The Export tab displays the selected note's $HTMLPreviewCommand, *if it has one*. A checkbox allows temporarily disablement of the preview command and Markdown processing. This makes it possible also to see the exported content as i is *before* it is post-processed by the preview command. The toggle is only used within the view, i.e. viewing the export code 'live', and thus has no background setting fro the tick-box.

Secondary windows

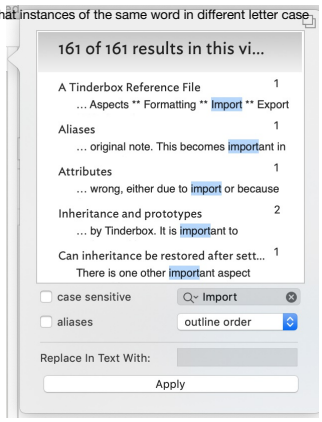
These are primarily 'torn-off' windows.

- Find results stand-alone dialog
- Get Info stand-alone dialog
- Roadmap stand-alone dialog
- Text window
- Text preview window

Find results stand-alone dialog

The Find results for main view searches are shown in a pop-over. The pop-over can be torn off to give a more persistent record of the search results. The torn-off Find window retains its selection when clicking on a note. Its title reflects the number of notes found in the document, and the number in the scope of the current view. Selecting a note in the torn-off find window will select that notes in the view, if the note is available.

[list of matching notes]. The list are shows notes matching the Find criteria. Results are listed by note title ($Name) sorted in the default, lexical, sort order. A lexical sort is used means that instances of the same word in different letter case do not sort together: `Ant, Bee, ant` and *not* `Ant, ant, Bee` as a non-coder might assume. To have greater control over the sorting of results or to get a count of matches, use an agent instead of Find.



For notes where the match is in $Text, the (first) match context is sown in a second line of listing with the match highlighted. Matches for note title or user attribute value use a single line listing.

Clicking on a list item makes that note the context of the front window's text pane. Listed notes are rendered in using $Color. Text for notes not in the current view scope are shown as a preview - see 'Preview windows' below.

The pop-over has a series of controls:

- search input box (from which a new search can be run). The box's pop-up menu offers an option for case-sensitive search, and an option to turn off regular expression search.
- a tick-box to alter the case-sensitivity of searches.
- a tick-box to allow inclusion of aliases (by default they are filtered out).
- sort order pop-up with the following choices:
  - outline order (default)
  - Creation date
  - last modified
  - name

The controls work live on the current search results (both in pop-over and tear-off form) without the need to re-run the Find.

Found items are grouped into two groups. The first grouping shows items within context of the current tab's view. Such items might not necessarily immediately be seen within the current view area, e.g. because they are outside the display part of a current Map or inside a collapsed container within a current Outline, etc. The second group lists all other matches that cannot be seen in the view pane without changing the context of the current view, i.e. setting its scope differently from the current settings. Clicking these out of scope list items invokes a preview window (see below).

The Find window can match outline separators and map adornments.

A contextual menu lets the right-clicked-on list item be opened in a new tear-off text window or in a new tab. Shift-double-click on a listed item will also open the result in a new tab.

Results from the Find results pop-over can be dragged into a text pane. The display name of the dragged item will be inserted into the text and linked to the corresponding note.

Results dragged into the view pane generate an alias of the dragged result's source note.

Find offers up to 7 suggested related words as possible autocompletions in the Find bar and in torn-off Find windows. Note: the related text feature only works on macOS 10.14 or later.

$Text replacement: this feature is only available in the pop-over mode of Find results.

Preview windows

After selecting a note in the Find window, if the note cannot be selected and previewed in the current view, Tinderbox displays a preview of the note's text.

Once displayed, either click the preview to close it, or drag it to tear it off as a stand-alone text preview window. Preview closing instructions are shown briefly, at the bottom of the preview pane, on first opening.

Closing the dialog

Click the dialog window's 'close' (red) button. This will close the dialog and any preview windows open.

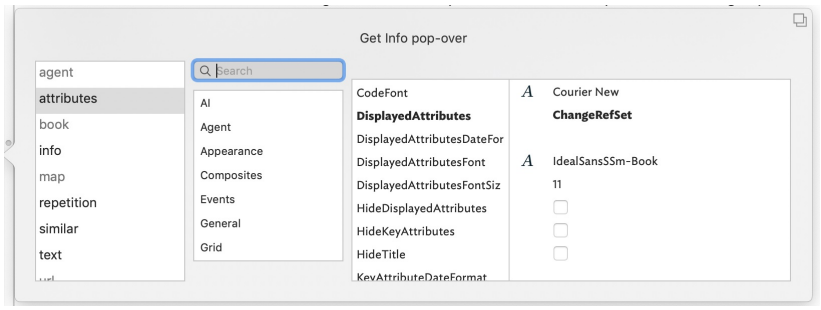
Dragging aliases

Results dragged into the view pane generate an alias of the dragged result's source note.

Dragging text links

Results from the Find results pop-over can be dragged into the text pane. The display name of the dragged item will be inserted into the text and used as the anchor of a text link to the corresponding note.

Get Info stand-alone dialog



The Get Info pop-over provides a series of tabs showing information about the currently selected note. Dismiss the pop-over by clicking anywhere outside it. The pop-over will position over the non-active pane of the main window, i.e. the main view or text pane.

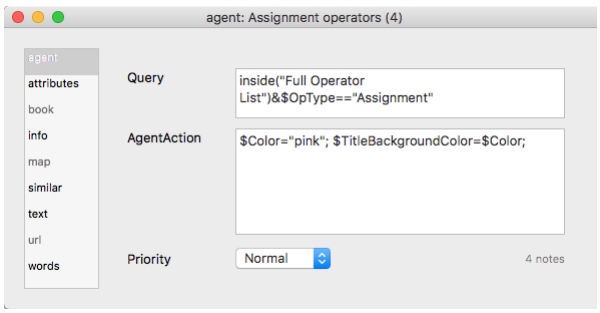
Dragging the pop-over will result in a stand-along dialog. This allows the Get Info information to be viewed and used when the source note is no longer selected. It is possible to have multiple Get Info dialogs open at the same time.

The sub-tab selected on first opening will generally default to the last one used in the current session.

The dialog has the following tabs:

- agent tab
- attributes tab
- book tab
- info tab
- map tab
- repetition tab
- similar tab
- text tab
- url tab
- words tab

agent tab



This tab controls basic agent functions. This replicates information found on the sub-subs of the Action Inspector. Edits made on the this dialog are live as soon as the Return key (↩) is pressed or focus shifts from the current input box on the pop-over.

This popover is shown by default as soon as a new agent is defined.

Agent title (only shown if the current object is an agent). The $DisplayName for the agent is displayed above the input boxes.

Query. The agent's query (stored in $AgentQuery). Auto-complete is offered for action code terms and system/user defined attributes (if starting with a '$' prefix, as is best practice). Editing this code is the same as if editing the Action Inspector's Query tab.

Action. The action to be applied to each child alias (stored in $AgentAction). Auto-complete is offered for some input, as described above. Editing this code is the same as if editing the Action Inspector's Action tab.

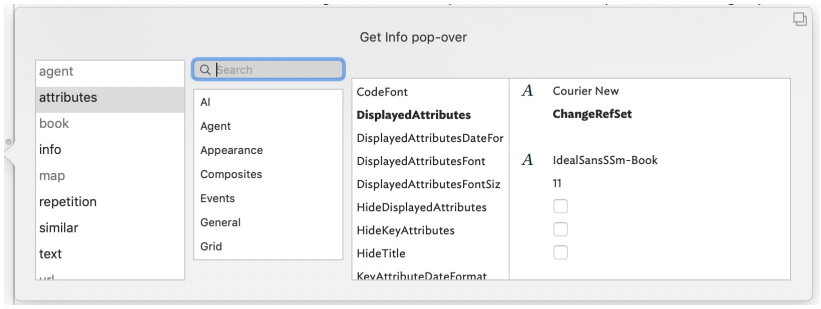
Priority. This opens the Agent Priority pop-up list, allowing the agent to be turn on, off or to a non-default working state (e.g. on, but at lower priority). Setting is stored in $AgentPriority. Using this is the same as using the Action Inspector's Query tab's **Priority** pop-up.

Result count (only shown if the current object is an agent). The count of matches to the current query is shown in grey text bottom right of the pop-over.

Dragging the pop-over will result in a tear-off window that will persist until the end of the current session (i.e. the document and/or app are closed). A torn-off window may be closed during the current session if no longer needed.

From v9.5.0, the agents query and action code boxes receive action code syntax colouring.

attributes tab



This tab shows attribute data and is the most direct successor to the Get Info dialog in previous versions of Tinderbox. The tab has 3 sections.

Search. This works as per the Document Inspector, System tab, in seeking to auto-match any input to a currently defined attribute (here for system or user). If multiple matches are found, these are displayed in a pop-up listing allowing the user to select the preferred match. If a match is found, the correct attribute group is selected in the listing and its contents loaded. From this the matched attribute is also selected. Here, $BorderColor has been searched for and matched.

Attribute Group listing. A list of all the defined groups of System attributes and any defined User attributes.

Attribute data table. A list of attributes within the currently selected group. The left column shows the attribute name. The right column shows the current local or inherited value. Display and editing of attribute values is as described for the Displayed Attributes table. String, Number, Set and List type attribute will display a pop-up menu showing values currently applied to that attribute elsewhere in the document.

Row height in the table. From v9.5.2, String-type (only) attributes can have a bigger line height. The default row hight for String-type attributes remains one line, but this can be extended to a maximum of seven lines. Within that space line breaks in the String's value are honoured but any overflow is clipped (i.e. additional content cannot be scrolled). Line height is set per-attribute using The Document Inspector's System and User tabs. From v9.6.0, multi-line attributes are also permitted for Sets and Lists as well as String type attributes.

Some attribute data types show an icon between the attribute name and value cells. The function of these is described under the notes on the Text pane's Displayed Attributes table.

The meaning of the styling of different rows (bold, strikethrough, etc.) is explained here.

Dragging the pop-over will result in a tear-off window that will persist until the end of the current session (i.e. the document and/or app are closed). A torn-off window may be closed during the current session if no longer needed.

Editing the $Prototype value in this dialog is just one way of setting a note's prototype.

The attribute $ReadOnly is exempted from being read-only, even if set to `true`. This allows you to turn off ReadOnly from the Displayed Attributes table or from Get Info ▸ Attributes.

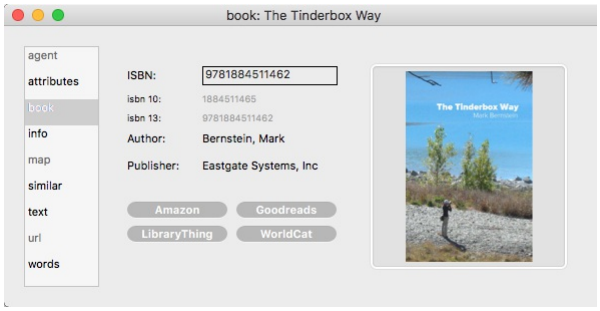
Changes to Displayed Attributes affect $Modified

Changing an attribute value in the displayed attributes table or in Get Info's attributes tab updates $Modified. Changing an attribute value in a stamp (including Quickstamp) or an action does not update $Modified.

book tab

The book tab shows book-related data. For notes with an $ISBN value, Tinderbox will assume the note is about some from of book and the book tab will:

- look up author and title, if not already known, from ISBNDB.com
- look up the cover image from amazon.com
- provide useful links to Amazon, Goodreads, LibraryThing, and WorldCat for this book.



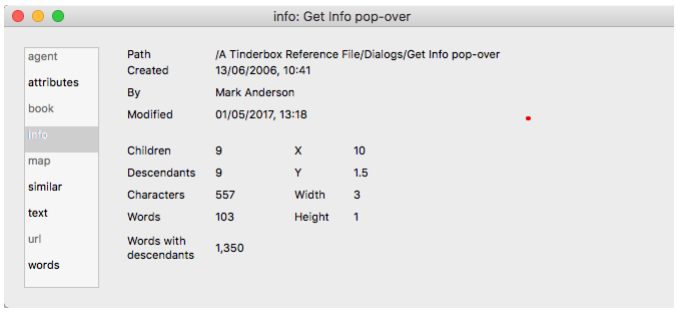
The **ISBN** box shows $ISBN data or allows it to be set directly by editing in the box.

Dragging the pop-over will result in a tear-off window that will persist until the end of the current session (i.e. the document and/or app are closed). A torn-off window may be closed during the current session if no longer needed.

info tab

The info tab holds basic metrics about the current note, mostly relating to attributes:

- **Path**. $Path.
- **Created**. $Created.
- **By**. $Creator.
- **Children**. $ChildCount.
- **Descendants**. $DescendantCount.
- **Characters**. $TextLength.
- **Words**. $WordCount.
- **Words with descendants**. $WordCount plus the sum of descendants' $WordCount.



Dragging the pop-over will result in a tear-off window that will persist until the end of the current session (i.e. the document and/or app are closed). A torn-off window may be closed during the current session if no longer needed.

map tab

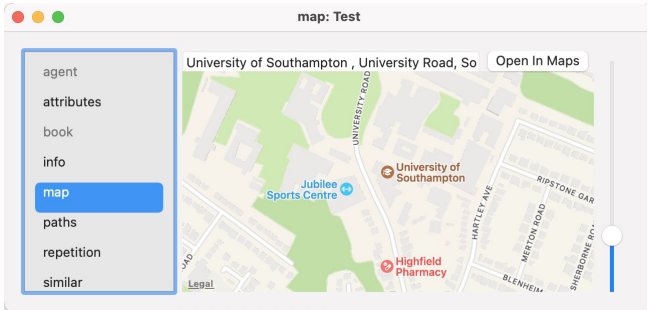
The map tab checks to see if $Address is populated and if it is will attempt to resolve the address into Google Maps, populating $Latitude, $Longitude and $GeocodedAddress.

The top input box shows data in $Address, or adding data to the box will set there attribute.

If a map location is found, the map can be zoomed via the slider control to the right of the map.

Dragging the pop-over will result in a tear-off window that will persist until the end of the current session (i.e. the document and/or app are closed). A torn-off window may be closed during the current session if no longer needed.

From v9.6.0, the map tab has an **Open In Maps** button that opens the (macOS Apple) Maps application.

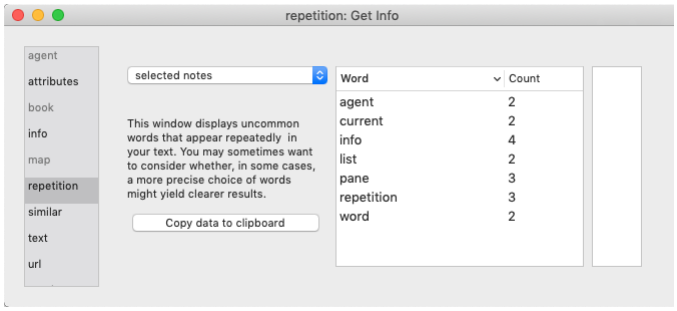


repetition tab

The left list-box shows the repetition count of each repeated word. The right listbox shows the context of occurrence of the currently selected repetition (i.e. left list-box selection) with the text excerpt highlighting the repeated word. Clicking on an item in the right list-box opens that note in the text pane of the front document window. The data in the list-boxes can not be copied.

The **Scope** of inspection is variable, selected by a pop-up menu with the following choices (the last option is the default):

- **document** (whole current/front document). Use this option with caution in a very large document, it may take a little while to complete scanning the existing text.
- **parent & descendants**
- **note & descendants**
- **selected notes** (default)



The **Copy data as clipboard** button copies the word/count list data to the clipboard as a tab-delim list.

A table of results shows the discrete words and the count of said words. The occurrence list can be sorted on any of the columns (default is 'word' in alphabetical order).

The repetition tab thus offers insight into words that are used repeatedly in the selected notes, sections, or in the entire document. Consistent usage may be needful or desirable, of course, but noting repetition can call attention to opportunities

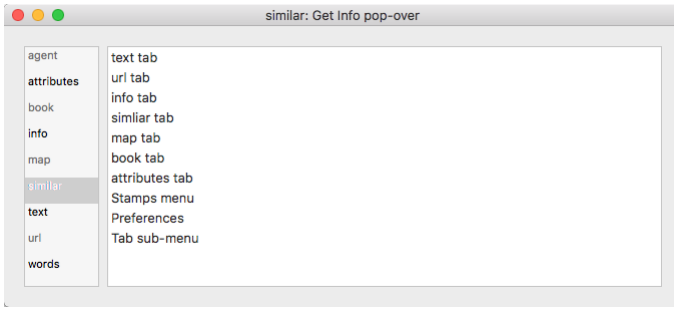
to adopt more precise language.

The pane lists words that occur two or more times except:

- words with fewer than four characters
- words that appear in the built-in 'stoplist' of 100 common English words
- words that appear in an option customised stoplist.txt in Tinderbox's Application Settings.
- words that appear in a note in the current document named 'stoplist', if one exists. The note title is case sensitive, but the note can be placed anywhere in the document (most likely away from the primary content).

The repetition indexing process tries to treat words derived from a common stem as repetitions, so plurals and verb conjugations are often handled intelligently.

similar tab



The similar tab uses Tinderbox's 'similar to' function to list (up to) 10 items it adjudges most similar to the current note. It replaces the stand-alone minor view in older Tinderbox versions. Similarity is based on several factors, including:

- the text ($Text) of the note
- the note title ($Name)
- any text contained in user attributes (i.e. attributes of String type or which are string-based)

In addition, weighting is applied for:

- notes having the same prototype
- notes having roughly similar amounts of text

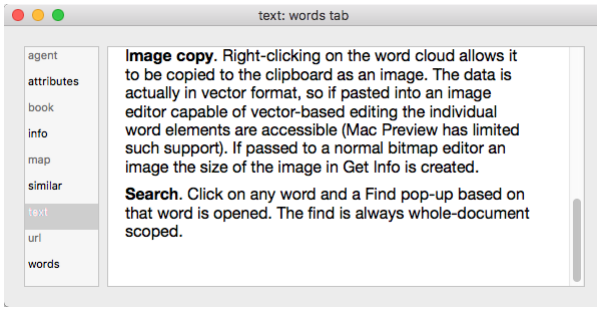
Similar notes are listed in order from top to bottom, starting with the most similar note at the top.

Double-clicking any list item sets the clicked note as the focus of the document window's text pane and dismisses the pop-over.

Dragging the pop-over will result in a tear-off window that will persist until the end of the current session (i.e. the document and/or app are closed). A torn-off window may be closed during the current session if no longer needed.

This information can also be accessed as an action code similarTo() and an export code ^similarTo()^.

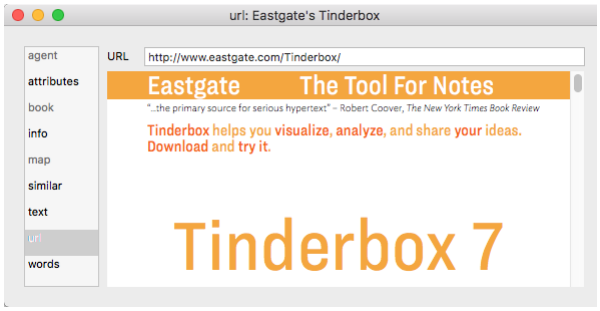
text tab



The text tab shows the $Text space of the current note. Dragging the pop-over will result in a tear-off window that will persist until the end of the current session (i.e. the document and/or app are closed). A torn-off window may be closed during the current session if no longer needed.

This feature is offered in a richer form by opening a stand-alone text window for a note. Links in $Text are non-functional in this display.

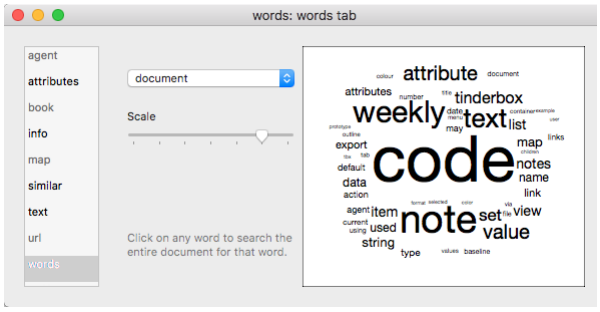
url tab



If the note's $URL is populated, the url tab will preview the linked webpage. The web content in the preview is live and links can be followed (but it is not intended for general web browsing).

Dragging the pop-over will result in a tear-off window that will persist until the end of the current session (i.e. the document and/or app are closed). A torn-off window may be closed during the current session if no longer needed.

words tab



This tab visualises the most common words in the $Text and $Name of in-scope notes. The word cloud acknowledges both the built-in stoplist and document's own stoplist.txt list, if present.

Word clouds and aliases: If viewing the word cloud of a note and its descendants, and if the selected note is an *agent*, the word cloud indexes each alias found by the agent.

The tabs controls are:

'scope' pop-up. The scope of examination within the (current) document can be set to one of:

- **document**: whole document (default)
- **parent & descendants**: parent note of the current note and all the parent's descendants
- **note & descendants**: current note and all its descendants
- **selected note**: currently selected note

The scope shown in the pane is equivalent to the export output of ^documentCloud^.

'Scale' slider. The view can be magnified or reduced by dragging the dragging the slider.

Image. A word cloud of the most common words found in the current scope. The words are drawing in $NameFont. Re-sizing the pane causes the layout of the contents to be re-arranged.

Image copy. Right-clicking on the word cloud allows it to be copied to the clipboard as an image. The data is actually in vector format, so if pasted into an image editor capable of vector-based editing the individual word elements are accessible (Mac Preview has limited such support). If passed to a normal bitmap editor an image the size of the image in Get Info is created.

Search. Click on any word and a Find pop-up based on that word is opened. The find is always whole-document scoped.

For note and section, each distinct occurrence of a word is counted. For the document level view, Tinderbox counts the number of notes that contain a word 1 or more times. The display contains up to 100 words - less if there are fewer valid words in scope.

Unlike the 'similar' notes tab, only $Text and $Name are interrogated. Other sources of words such as textual user attributes are thus ignored.

If the dialog is torn-off as a discrete window, subsequent changes to in-scope $Text are not reflected; re-select the window sub-tab to refresh the view. Similarly, the 'selected only' scope only maps the note selected at the time the view was opened, and does not track changes in selection while the tab is active.

This tab replaces the 'Common Words' view in very early versions of Tinderbox.

Roadmap stand-alone dialog

The Roadmap dialog is created by ' tearing off' a Roadmap pop-over. The Roadmap lets you see all of the internal links leading to or from a selected space—the local area of a complex hypertext (web links out of the document are not included. Aliases have their own Roadmap, not that of their original. The Roadmap not only gives and overview of inbound and outbound links but also allows exploration of the local hypertext (i.e. the TBX's content). Links out of the current document, e.g. web links, are not visualised in the Roadmap. RTF-based 'Smart' links are also omitted from the listing.

The left column lists all notes that link into the current note note; the column header lists a count of the inbound links. The right column lists all the note's outbound links, again with a count in the column header. To change the focus of a Roadmap, double-click any item in either column. The Roadmap will change to focus on the selected note, with links leading in and out of it in the two columns. The current window's main view will update to the selected note, *if it is in scope*; note that if the pop-up is torn off the text pane stops following focus of the view.

From v9.1.0, Roadmap opens with the initial focus on the outbound links list if there are any outbound links. Otherwise, the inbound links list gains the initial focus.

Roadmap allows editing of the properties of the selected link. At the top of the Roadmap, Tinderbox additionally shows the *display name* of the note for which inbound and outbound links are currently being shown. The tooltip of this label is the full path of that note (which will include the *name* of the note.

Pressing the **Delete** key (⌫) will delete the selected link.

Tab (⇥) cycles from first listed inbound link → first listed outbound link → link type → first listed inbound link, etc. Use **spacebar** to follow a link (or **Return** (↩) from v9.1.0) and set window focus on the currently selected list item. Use *blin*

type to select a source or destination.

Note: if the main view is a map, selection/focus can move to other items on the current map but not to notes on other maps. In the latter case, only the Roadmap updates.

The Roadmap title shows the $DisplayName of the notes whose data is being displayed. Each listed link item shows:

- Line one. The $DisplayName of the linked note (in $Color) and at right its $Badge (if one is set).
- Line two. The link type of the link and, for text links only, that link's anchor text.
- Roll-over tooltip for either line: the full path of the linked note.

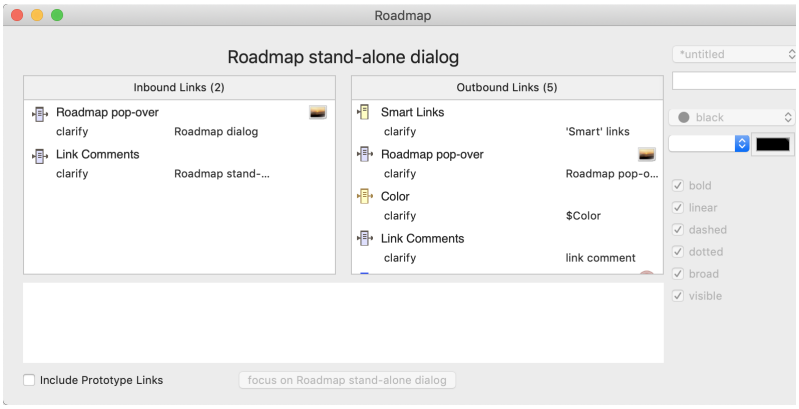
If the Outline Document Settings for 'Darker colors' is ticked, the item is rendered in a darker tint of their $Color.

Link Comment box. Below the in-/out-bound columns a box shows the link comment (if any) of the selected link; the comment can be edited. This box is not shown when the dialog is a pop-over,

Include Prototype Links. A tick-box (ticked by default) at bottom left allows prototype-type links to be filtered out. Prototype assignments are stored as links, in the TBX's data, but are not normally used in maps and excluded from link count attributes. However, leaving such links in the last can help tracing prototype-based relationships in a document.

Button **focus on [title of calling note]** . Located at middle-bottom of the dialog, this refocuses the dialog back onto the note from which the dialog was first opened. Or, if the user has shifted the selection of the current tab's view, the dialog updates to show the links for the currently selected note.

As stand-alone dialogs the Roadmap dialogs last only until the end of the current session; they cannot be saved to last across sessions.



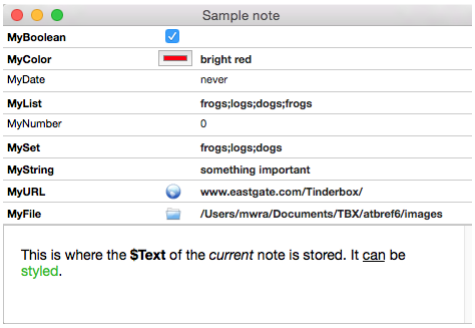
Text window

Any note (but not agents, etc.) can be opened as a 'tear-off' stand-alone window. The window display the note's $Text and Displayed Attributes table. Both are editable and will update the source note in the main window. There is no title bar or HTML/Preview tab. Note that drag-drop or link-dragging *between* windows is not supported.

When a document is reopened, any 'tear-off' text windows that were open when the document was saved are also reopened. These text windows are moved in front of other windows lest they be hidden and forgotten.

A standalone window can be created via the View menu or a shortcut (⌘⌥**X**).

The meaning of the styling of different rows in the Displayed Attributes table (bold, strikethrough, etc.) is explained here.



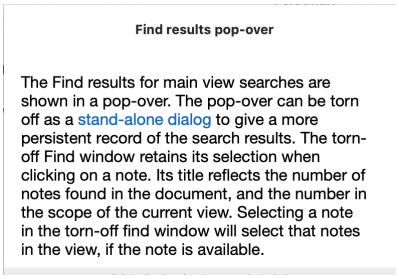
Text preview window

In some cases it is useful for Tinderbox to be able to preview the text, e.g. to confirm the note's $Text content. In such circumstances, a number of Tinderbox features (listed below) will generate a preview window.

Such preview windows are a form of pop-over. They show only the $Text of the target note (i.e. no Displayed Attributes or links panel). Clicking either on or outside the preview will dismiss it, but clicking+dragging the preview will result in creating a stand-alone Text window.

The following features use text preview windows:

- Text pane links panel.
- View pane, Find results pop-up.



Inspector

The inspector dialog has TBX document scope, i.e. you can open one for each open TB document. Its outcome works on the current selection (of the parent document). By default it opens with the Quickstamp pane selected.

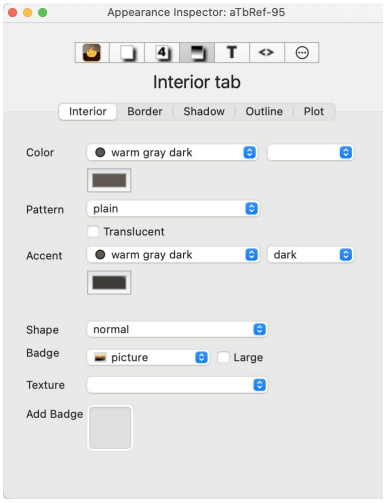
In keeping with Inspector-type windows in other apps the Inspector always sits in front of any other Tinderbox window. The Inspector is invisible when the Tinderbox app loses focus; it does not close but is simply not drawn on screen when another app is in use.

An Inspector's window title will show the name of the parent TB document, a colon, and either the $DisplayName of a single selected or the count number of notes in a multiple selection.

The Inspector may be resized. Code fields, in particular, may thus be expanded at will to accommodate more complex logic when needed.

In general documentation and community usage, the tab name or full name ate used interchangeably. Thus the 'Links Inspector' is the same as the 'Document Inspector's Links tab', etc. Likewise, the main Inspector name is assumed to refer to its default (first) tab: thus Appearance Inspector may be assumed to mean the Appearance Inspector's Interior tab unless a different tab name is given.

- Tinderbox Inspector
- Document Inspector
- Properties Inspector
- Appearance Inspector
- Text Inspector
- Export Inspector
- Action Inspector

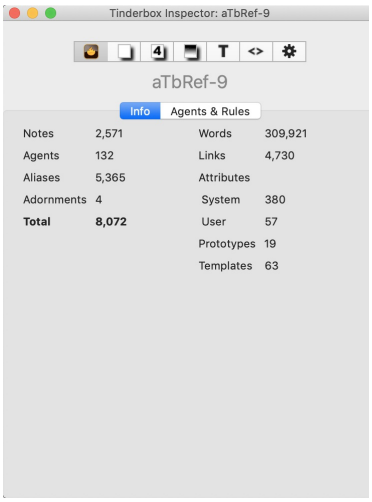


Tinderbox Inspector

The Tinderbox Inspector data is document-scoped, and displays the file name document for the current (frontmost) document if more than one is open.

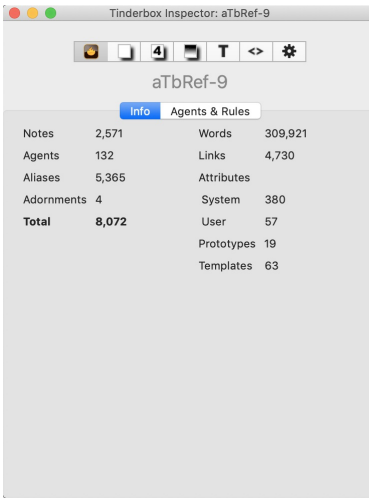
The Tinderbox Inspector has these sub-tabs:

- Info tab
- Agents & Rules tab



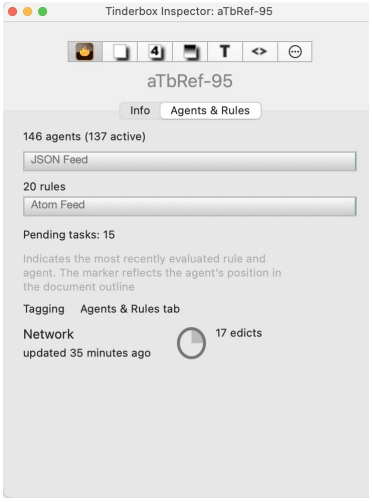
Info tab

The Hypertext Status view gives an overview of the current documents contents. It lists:



- **Notes**. The number of notes in the current document (containers count as notes, as do separators).
- **Agents**. The number of agents in the current document.
- **Aliases**. The number of aliases in the current document.
- **Adornments**. The number of adornments in the current document.
- **Total**. Sum of notes, agents, aliases and adornments.
- **Words**. The total word count for all notes.
- **Links**. The number of links in the current document (excluding 'prototype' type links).
- **Attributes**. The number of attributes defined in the current document:
  - **System**. The number of System (built-in) attributes. Hidden System attributes are not counted since the user cannot see them.
  - **User**. The number of user-defined attributes.
- **Prototypes**. The number of prototypes defined in the current document.
- **Templates**. The number of export templates defined in the current document.

Agents & Rules tab



This shows the total number of agents and rules defined in the document. This includes currently disabled agents and rules. An alias of a note with a rule adds to the rule count as the rule is evaluated in the agent context as well as the original.

Progress bars

The agent and rule 'progress bars' show a moving green line. This indicates the where in the agent or rule cycle execution is currently occurring. The horizontal position relates to the $OutlineOrder position of the item in question, as opposed to a proportional place in context of the number of rules or agents. If, for example, all rules are in notes in the middle of the outline order, the rule line will cycle but only through the middle part of the progress bar. The speed of progress indicates how fast or slow the cycle is running. If no defined items are active, the line for that bar will be static otherwise no movement may imply an agent or rule that has hung.

The agent progress bar list the number of agents and in parentheses the number of agents currently active. The latter is useful in fine tuning performance in complex documents. Note: the rule count is the total of notes with rules *and* all of their aliases: a note with a rule that has three aliases adds *four* rules to the overall rule count.

Each progress bar shows the title ($Name) of the currently executing item to help give an indication as to the scope of execution.

If automatic agent updates have been disabled, the agent bar shows the caption 'manual updates only'.

Pending Tasks. This shows how many pending operations are currently waiting on the agent task queue. This may be useful to know in very complex documents. Note that the exact nature of the 'tasks' is not described but the list number of tasks can indicate unseen load on the app created by the current active document.

[Grey text. This applies to the agents and rules progress bars rather than Pending Tasks.]

Tagging. From v9.5.2, this displays the name of the current note that is being tagged.

Network. The control to the right of the caption shows synching is in use and its status. This process covers AutoFetch, watched folders, DEVONThink watched groups, geocaching updates, Bookends import updates, and other periodic maintenance (Note: Simplenote synch was discontinued in v8). The processes described start to (re-)run as the progress reaches the 12:00 position (top centre). One cycle means all affected processes will have run once. The more processes, the longer the overall time to complete the cycle. Within this display it is not possible to reveal the exact processes running, e.g. to differentiate AutoFetch calls from geocaching updates, etc.

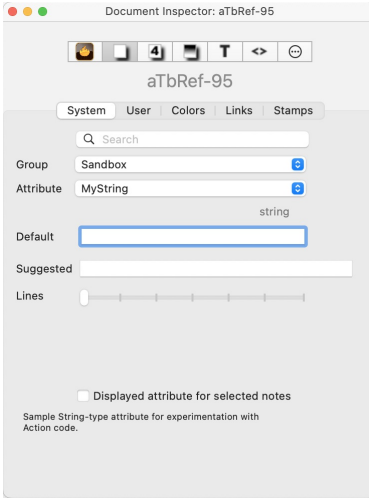
Edicts. To the right of the Network control a label indicates the number of edicts in use. If any edicts are present it shows beneath how recently they were updated.

Document Inspector

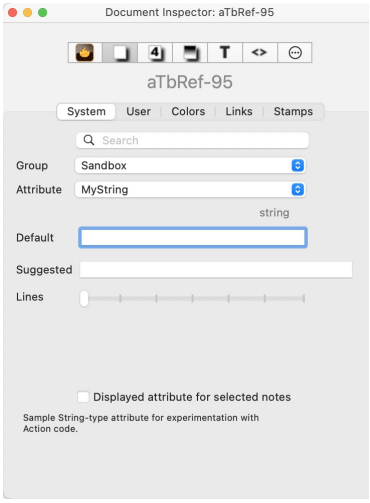
The Document Inspector data is document-scoped, and displays the file name document for the current (frontmost) document if more than one is open.

The Document Inspector has these sub-tabs:

- System tab
- User tab
- Colors tab
- Links tab
- Stamps tab



System tab



This allows the user to view the current document's pre-defined system attributes, which are added to any new TBX document (and added to automatically as new app versions introduce new system attributes). This Inspector allows the user to view and change their data type and default values.

Search box. This allows the user to start typing an attribute name that Tinderbox will attempt to auto-match, suggesting completions. If an item is matched (or clicked form the suggested match list), the tab will select the appropriate **Group** and **Attribute** names to populate the other controls.

Group. This allows the user to select a name from a list of any of the defined groups of Tinderbox pre-defined System attributes. By default, this loads the 'General' group, but within a current session will remember the last-selected group. In older releases, this was labelled 'Category'.

Attribute. This allows the user to select an individual attribute's data from the currently selected group (above). By default, this loads the 'AdornmentCount' attribute, but within a current session will remember the last-selected attribute. Any attribute shown in strike-through text is deprecated, though normally still present for legacy issues. The meaning of the styling of different listed attributes (bold, strikethrough, etc.) is explained here.

Intrinsic. If the attribute is intrinsic, a label 'intrinsic' in grey text is shown above the left end of the default value box. Otherwise this area is blank

Data type. The data type for the selected attribute is shown in grey text above and to the right of the default value box.

Default. This box shows the program default for the attribute in one of a number of forms:

- Grey text. These are inherited defaults, normally from Document settings, and can not be edited (at Document level) via the Inspector interface. Inherited defaults may be qualified with a suffix in one of two ways, to give an indication as to why they can not be edited here:
  - ' (preference)'. This attribute's default can only be set via a control in Document Preferences. An example is $TextFont. Use the appropriate Document Settings control to change the default.
  - ' (read only)'. This is a calculated total, e.g. $ChildCount. It is calculated on the fly and whilst it has the notional unset default of its data type it invariably seen as a current calculated value.
- Black text.This is a default that may be modified by the user. Changing defaults is not recommended until the user has some experience of Tinderbox. Enter a new value and press the Return key (↩) to commit the change.

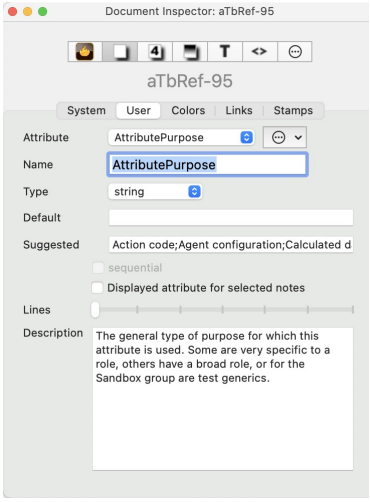
Suggested. Enter an optional list (semi-colon delimited) of suggested attribute values.

Lines (shown for String-type only). From v9.5.2, a slider control allows allows a variable number of lines to be used to display String-type attribute values in the display tables of Displayed Attributes and Get Info/attributes. The default value is 1. The slider has seven stop positions running from left (1) to right (7). From v9.6.0, multi-line attributes are also permitted for Sets and Lists as well as String type attributes.

Displayed attribute for selected notes . Ticking this box will add the attribute as a displayed attribute to all currently selected items. In general this is a less useful way of adding Displayed Attributes than using the control at the top of the te pane. As changing the value here changes the note's $DisplayedAttributes, it is not the best method to set/manage Displayed Attributes if using prototypes as the edit may break inheritance of the attribute (more on Displayed Attributes). If some selected items do not use this Displayed Attribute and some do, the box remains un-ticked. Ticking it sets this for the whole selection.

Description. This is a short description of the purpose of the selected attribute. Occasionally, this may be blank (though it implies nothing other than a missing label!).

User tab



This allows the user to view the current document's user-defined (i.e. custom) attributes. This Inspector allows the user to view and change their data type and default values.

Attribute. This pop-up menu lists all User attributes already defined (otherwise the list is empty). Selecting a list item loads it into the **Name** box below the pop-up. The list is sorted case insensitively (i.e. A–Z, before a–z, before 0–9).

Gear wheel button (right):

- **Delete user attribute**. Press this option to delete the attribute currently shown in the Name box. See more detail below.
- **New user attribute**. Press this option to add a new user attribute. See more detail below.

Name. Shows the name of the selected **Attribute**. Also used for setting the name of a new attribute. This input rejects attempts to create an attribute name that contains operators like + or punctuation like ";".

Type. The data type for the selected attribute. Open the pop-up to change the data type. The change is saved as soon as this control loses focus. The selected value (as a zero-based number) is recorded the document's XML attribute type definition for this attribute.

Default. This box shows the program's default for the attribute's selected Type (more on data types). The value may be left as the default or set to a desired initial value.

Suggested. Enter an optional list (semi-colon delimited) of suggested attribute values.

sequential (tick-box). An option only for Number-type attributes. When ticked all notes, including existing notes are allocated a sequential number. Unused numbers, e.g. from deleted notes, are not re-used making the number also a unique identifier. Notes duplicated or copied/pasted are allocated a new number. Aliases use the original's value.

Displayed attribute for selected notes . Ticking this box will add the attribute as a Displayed Attribute to all currently selected items (adding to existing Displayed Attributes if found). In general this is a less useful way of adding Displayed Attributes than using the control at the top of the text pane. As changing the value here changes the note's $DisplayedAttributes, it is not the best method to set/manage Displayed Attributes if using prototypes as the edit may break inheritance of the attribute (more on Displayed Attributes). If some selected items do not use this Displayed Attribute and some do, the box remains un-ticked. Ticking it sets this for the whole selection.

Lines (shown for String-type only). From v9.5.2, a slider control allows allows a variable number of lines to be used to display String-type attribute values in the display tables of Displayed Attributes and Get Info/attributes. The default value is 1. The slider has seven stop positions running from left (1) to right (7). From v9.6.0, multi-line attributes are also permitted for Sets and Lists as well as String type attributes.

Description. This holds an optional user's description of the intent and/or usage of the attribute. The description is only seen in the inspector when when the attribute is actually selected. The value is stored in the TBX but cannot be accessed except via this Inspector.

Creating a New User Attribute

On selecting the option to make a new attribute, the **Attribute** pop-up will change to show 'NewAttribute'. The **Name** box is not refreshed; any previously shown attribute name is retained deliberately, in case it is to be used as part of the ne attribute. Edit or delete the existing text to add the desired new value. The new attribute is created as soon as focus shifts away from the Name box. The default data type is 'string' and the default value is no value, i.e. an empty string. Optionally, a description of the attribute, e.g. its purpose, my be added. The latter description is only seen in this Inspector and when this attribute is selected.

When a new attribute is created, the default value of that attribute is set to a conventional value. For example, the default value for a new numerical attribute is 0, and the default value for a new Boolean attribute is `false`.

When a 'new attribute' is selected in a document where user attributes already exist, the existing selected attribute name is retained but all other value input boxes are reset to default.

Altering an existing attribute

Note: *there is no undo for this operation* .

Select the attribute from the **Attribute** list. **Name**, **Type**, **Default**, **Description** and tick-box settings can all be changed. The changes take effect once focus moves off the edited input box. When a user attribute is renamed, the values of the old attribute are moved to the new attribute, and Displayed Attributes referring to the new attribute are updated to use the new attribute. But, before changing data *type*, do consider the effects: it may make more sense to make a new attribute of the desired type and use an agent to copy values across with whatever additional changes are needed over and above basic type coercion. Changes to the attribute name will update Displayed Attributes but do not update attribute names in action code (e.g. roles, edicts, etc.). If using action code in a TBX, consider this aspect *before* altering an existing attribute.

When the type of an existing attribute is changed, the default value of that attribute is reset to a conventional value. For example, the default value for a new numerical attribute is 0, and the default value for a new Boolean attribute is `false`
When a 'new attribute' is selected, the existing selected attribute name is retained but all other value input boxes are reset to default.

Deleting a User Attribute

Select the attribute to be deleted sing the **Attribute** list. Once selected, chose the **Delete user attribute** option. The selected item is deleted immediately. Note: *there is no undo for this operation* .

Colors tab

This tab allows inspection and editing of the current documents defined (named) colours. By default all new Tinderbox documents have 28 defined colours. There are 10 greyscale colours named '0' though '9' (from light to dark) and 18 colours named for the actual colour ('green', bright red', etc.). The colours may be added to or deleted or the colour represented by the name may be changed. The colours represented can be seen (online only) here.

Color. The pop-up menu shows the 28 defined colours in alphabetical order (0-9 then a-z). Any new user-added colours are listed at an appropriate place in this list.

Gear wheel button (right):

- **Delete color**. Deletes the currently displayed colour.
- **New color**. Adds a new colour to the list. The default name is 'new colour' and the default colour is #999999. Both the name and colour value can be edited.
- **Save Color Scheme...** Opens a macOS file save dialog for saving the current colour scheme as a Tinderbox colour scheme (.tbc) file. The file is saved to the app's current folder. Note the 'color schemes' folder for making schemes available to all files via the app itself.

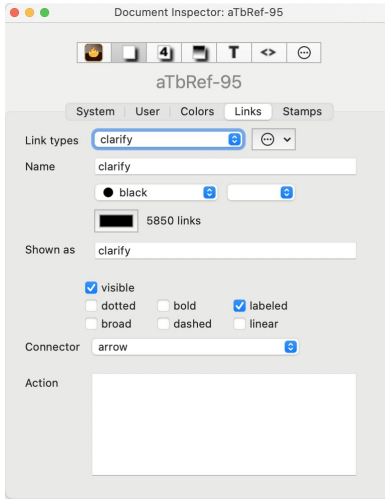
Colour controls. The colour chip shows the current colour defined. Clicking on the chip opens the OS' Colors dialog. Any picker on the dialog may be used to define a colour which is used to update the colour chip and the hexadecimal code for that colour.

Name box. This box shows the name stored for a defined colour. The name can be edited if desired. Note colours are case-sensitive - 'blue' and 'Blue' are treated as separate colours.

Hidden tick-box (default un-ticked). Unwanted system colours can be omitted from colour menus and lists. To hide a colour, select it and tick the "Hidden" box. Hidden colours can still be used in rules and actions, and their definition can be changed in this Inspector. However, the hidden colours will not be shown in menus and pop-up colour lists.

See more about Colours in Tinderbox.

Links tab



This tab allows management of the names and styling of the link types defined for the current document. Link types can be added, deleted or their settings edited.

Link types. This pop-up menu shows the 11 link types defined for all new documents, as amended by any edits/additions/deletions made by the user.

Name. This shows the name of the currently selected Link Type. The name can be changed.

(Link Color). There are 3 'standard' colour controls that set $Color:

- Defined colour list pop-up menu.
- Colour shade pop-up menu.
- Custom colour picker chip. Shows the colour used to draw this type of link line (and link labels) in map view. The default is black but can be customised per-link type.

From v9.6.0, when a new link type is created via the Links Inspector, it takes its colour from the document's current colour scheme (previously new link types always used black).

(Use count of selected type). From v9.6.0, the Links Inspector reports, for the *currently selected* link type, the overall number of links of that type in the document.

Gear wheel button (at right):

- **Delete "[link type]"**. Deletes the named link type, which is the type currently selected. Some link types are required by Tinderbox (e.g."untitled, prototype, note, note+) and cannot be deleted; if selected this menu option is greyed out.
- **Delete all unused link types**. Deletes all link types (bar required ones) which do not have a least one link of that type defined in the current document.
- **New link type**. Adds a new link type to the list. All three boxes **Link types**, **Name** and **Shown as** will show a default value of 'new link'. Edit either/both of the latter two to the desires title(s) and change to colour and tick-boxes as required. Note: the value 'text' is used internally and so cannot be used a custom link type name.

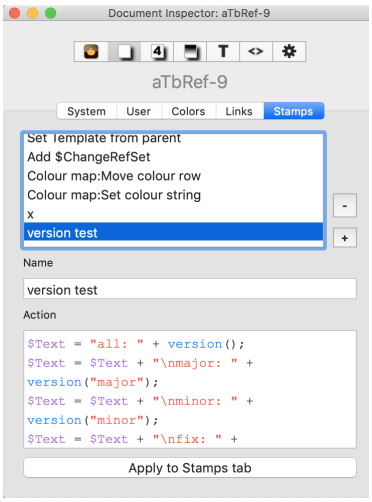
Shown as. This is an alternative screen label for the link type's **Name**. If using link types in action code, it can be useful to have short simple names for labels that would otherwise be long or use accents, etc., that might be troublesome in code context.

- **Tick boxes**:
  - **visible** tick-box (default = ticked). This indicates whether this Link Type is to be shown in Map views.
  - **bold** tick-box (default = un-ticked). This indicates whether links of this Link Type is are drawn in bold in Map views.
  - **labeled** tick-box (default = ticked). This toggles the visibility of the text label for the currently selected Link Type in Map views.
  - **dotted** tick-box (default = un-ticked). This indicates whether links of this Link Type is are drawn as dotted lines in Map views.Note that ticking both **dotted** and **dashed** boxes gives a dot-dash line type.
  - **dashed** tick-box (default = un-ticked). This indicates whether links of this Link Type is are drawn as dashed lines in Map views. Note that ticking both **dotted** and **dashed** boxes gives a dot-dash line type.
  - **linear** tick-box (default = un-ticked). This indicates whether links of this Link Type is are drawn as straight lines in Map views. This setting trumps the 'broad' setting.
  - **broad** tick box (default un-ticked). This setting indicates if bezier links are to be drawn in the broad style.
- **Connector**. Link types (and the Link Type Inspector) let you choose a connector type: **arrow** (default) or **circle**. New links of a given type adopt the type's connector, and you can override the connector type for individual links as before. Link types record the preferred connector type, and links will use the corresponding connector unless individually overridden.
- **Action**. LinkTypes can define an optional OnLink action that will be performed whenever a new link of that type is created. When running the OnLink action, the source is bound to the link's source note and destination is bound to the link's destination note. The designator this is also bound to the source note. This action has no associated attribute: the action is a document-level setting and can only be entered via the Links Inspector. From v9.1.0, the link Action field performs action syntax colouring.

Code fields do not select the entire text after when gaining focus or after pressing the **Return** key (↩) to update (save changes). This lessens the chance of accidental deletion of existing code.

Individual links show the styling inherited for their given link type, but can be further customised using a note's Browse Links dialog.

Stamps tab



The top list box holds a list of all those Stamps configured for the current document. Clicking on an item makes its values appear in the other boxes on the form. The stamps are also listed in the Stamps menu for easy use.

If the current document has the Built-in Hints folder with the Stamps container added, this tab can still be used to managed Stamps also viewable via the Stamps container. Code fields do not select the entire text after when gaining focus or after pressing the **Return** key (↩) to update (save changes). This lessens the chance of accidental deletion of existing code.

Stamp list. This lists the names of all currently defined stamp. Any list items can be dragged to re-order them. Altering the listing order (see below)also updates the Stamps menu's listing order.

- (minus) button. This deletes the currently selected stamp in the list

+ (plus) button. This adds a new stamp. The initial stamp name defaults to an unused name rather than previous "stamp name", making it easier to create short-lived, temporary stamps. If a stamp is selected when a new stamp is created, the new stamp is inserted immediately below the selected stamp and its initial name is derived from the selected stamp.

Name. This shows the name of the currently selected Stamp. It is the also the name as seen listed in the Stamps Menu. The name can be changed by editing this box. Such change only affect the listing's title for the stamp, the action code is unaffected.

Action. This holds the action expression(s) to be applied by the Stamp. Typing in the box triggers auto-complete for action codes and (if using a $ prefix) any system or user attributes. The action code can be (e-)edited; the change is applied if the Return key (↩) is pressed or the box loses focus. Any subsequent use of the stamp will use the altered code. The code panel supports action code syntax highlighting.

Apply. This button applies the currently loaded stamp to the main view selection (of one or more items). The button label shows the $Name of the selected note or if a multiple selection, the number of items in the selection. When a single note is selected, the Apply button reflects the note's $DisplayName rather than its $Name, and the name is correctly encoded.

Re-ordering the stamp list

Any list items can be dragged to re-order them. Altering the listing order (see below)also updates the Stamps menu's listing order.

Note that the listing order in the Inspector's list is not synched with the Hints' Stamps container, so the list order in the two *may* differ over time. The Stamps menu uses the Inspector's listing.

Exporting & Importing Stamps

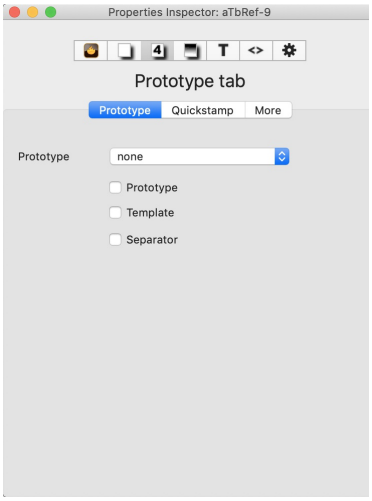
Stamps can be dragged from the Stamps to Finder windows or the desktop. To import a stamp drag/drop a stamp file (.tbxstamp) from a Finder onto the view pane of a TBX document. To copy a stamp between TBX documents, export to Finder from the source document and then drag the stamp file into the target document.

Properties Inspector

The Properties Inspector data is selection-scoped, and displays note $DisplayName for the currently selected item; if more than one item is selected it shows the number of items in the selection (e.g. 'Visual Styling' for a single note of that name, but 3 notes selected shows '3 Notes').

The Properties Inspector has these sub-tabs:

- Prototype tab
- Quickstamp tab
- More tab



Prototype tab

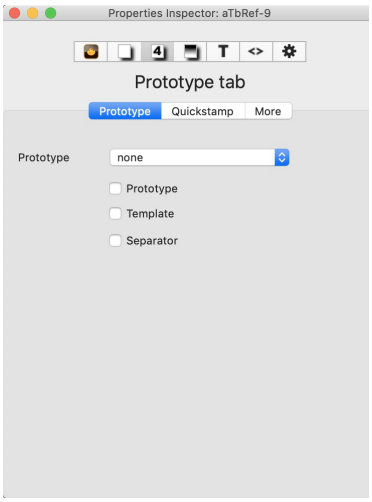
This tab defines a number of special purpose tasks for notes. This can be opened directly via the shortcut [Cmd]+3.

Prototype (list). This holds a list of prototypes defined for the current document. The current selection's prototype is pre-selected; default is 'none'. A prototype cannot use itself, but can use another prototype. So if a prototype is selected, it will be greyed out in the pop-up list.

Prototype (tick-box). Ticking this makes the selected note into a prototype. This is just one means of setting a note's prototype.

Template. Ticking this defines the note as being used to store code for an export template.

Separator. Ticking this defines the note as a separator and thus is visible *only* in Outline view.



Quickstamp tab

The Quickstamp tab allows you to set a single attribute's value on one or more currently selected notes. It remembers the last used settings during the current session (i.e. until the TBX is next closed). Instructions for using this tab are below the description of the controls.

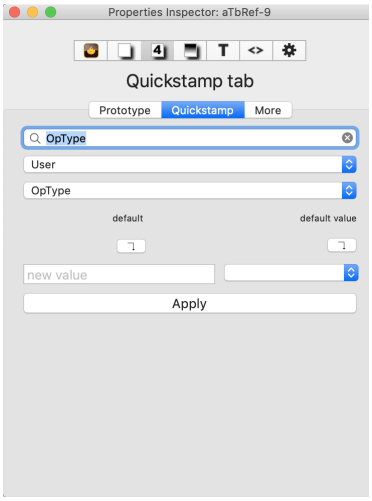
Search box. This allows the user to start typing an attribute name that Tinderbox will attempt to auto-match, suggesting completions. If an item is matched (or clicked form the suggested match list), the tab will select the appropriate **Group** and **Attribute** names to populate the other controls. This box has focus when the dialog is opened via keyboard shortcut (Cmd+2).

Group. This allows the user to select a name from a list of any of the defined groups of Tinderbox pre-defined System attributes, or the 'User' group of user-created attributes (if any).

Attribute. This allows the user to select an individual attribute's data from the currently selected group (above). Read-only attributes as shown in italics. They can be selected, so their values can be viewed, but the cannot be edited. The meaning of the styling of different listed attributes (bold, strikethrough, etc.) is explained here.

Default/value states. These are two bent-arrow (↰) buttons each with a label above it and to its left a value text:

- **Inherited value** (left button). The value shown is the value to be inherited if different from the local value. Click the button to apply this value/inheritance to the current selection, and (re-)set inheritance. Labels for this button are either of:
  - **default**. Labels the value that is (or can be) inherited from the doc preference or doc's default for that attribute.
  - **inherited**. If the current note has a prototype (or all notes in a selection use the same prototype) this labels the value that is (or can be) inherited from that prototype.
- **Value assigned to this note** (right button).The value shown is the inherited value (i.e. same as the above) or a local value particular to this note (and thus breaking inheritance). Click the button to re-apply this value to the current selection, e.g. if editing a complex experiment and making a mistake. Generally, if re-setting inheritance, use the left button. The label for this (right-hand) button is any of:
  - **default value**. This value is inherited from the doc preference or doc attribute default.
  - **inherited from** *xxxx*. This note (or selection uses the prototype named *xxxx*. The value shown is the prototype's value for that attribute.
  - **assigned to this note** . The value is set locally, breaking any inheritance of both doc defaults and prototype values. Setting an attribute value locally to the same as a prototype/default value does **not** restore inheritance. To do the latter use the left-hand assignment button above the input box.



Value edit box. An input box for reviewing/changing the current value of the selected attribute for the current note. Altering the value and using the Apply button will set a local value for the selected attribute, for the current note selection. From v9.5.0, when Quickstamp is editing a Boolean attribute, it now displays a checkbox in place of the value text field, which should make editing boolean attributes a bit easier.

Existing value pop-up list. If the selected attribute is of String, List or Set data type, this pop-up shows existing unique values for the attribute (up to a maximum of 199 values; if more only the first 199 are shown). Clicking on a value in the lis inserts it into the **Input box**.

Apply button. Applies the Value box value to the selected item(s). With very large selections this may take a few seconds. Using the Return key (↩) acts as if clicking this button.

Note: using this Inspector to set the value of $Prototype is just one way of setting a note's prototype.

How to use the Quickstamp

The Quickstamp acts on every selected item, be it one note/agent/adornment or several. The Quickstamp can be used in several ways:

- Step 1. *Setting the target attribute*: First use the top three controls to select a particular system or user attribute. If the attribute name is known, type it in the **Search** box and select the attribute from the auto-complete list. This will configure the next two pop-ups—**Group** and **Attribute**. Or, use the latter manually, in turn to set an attribute **Group** (top) and **Attribute** within the selected group (bottom). In the illustration, a User (group) attribute 'WebImage' has bee selected.
- Step 2. Editing the attribute value. There are a number of different actions than can be taken:
  - *Entering or editing a local value* . This is done using the input box on the left side immediately above the **Apply** button. If no local (i.e. per-note) value is set, the inherited value is shown. If the latter is 'no value' (i.e. an empty strin the box shows a prompt text of 'value'. Enter the desired local value here; if the note already has a local value, simply change the entry to the desired new value. When ready, click the **Apply** button to save the change.
  - *Using existing or suggested attribute values* . When editing or entering a local value, it can be useful to simply use a value used for this attribute by other note(s) in the document, or to use suggested values if these are defined. such a case, the **Existing Value** pop-up to the right of the value input box (i.e. *right* side above the **Apply** button) can be opened. Selecting a value from the pop-up list sets that as the text of the value box to its left. Any change made is not saved until the Apply button is saved.
  - *Re-setting the inherited value* . If a local value has been set, it is possible to re-enable inheritance of the document default or a prototype-set value by using the *left-hand* of the two bent-arrow (↰) buttons above the value editing box. Use the document button to save the change. The right-hand arrow button can used to reset the pre-existing local value, but only until the **Apply** button has been used, after which it functions like the left-hand button.

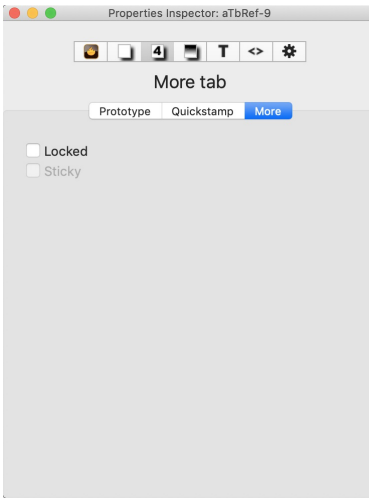
Unlike editing attributes via a note's Displayed Attributes table or Get Info's attributes tab, altering an attribute via a Quickstamp (or stamp) does not alter $Modified.

More tab

This tab holds two setting associated with Map view use.

Locked. A locked note or adornment is 'locked' to the map background. Moving the note moves the whole map. The locked state is stored in $Lock.

Sticky. A sticky note or adornment means that when another note overlaps (on top of) it the latter is moved when the underlying note is moved. Sticky adornments are useful so that if the adornment is moved all the notes on it remain in place on the adornment and move with it. The stickiness state is stored in $Sticky. This control is disabled (greyed out) when the selected note is not an adornment.

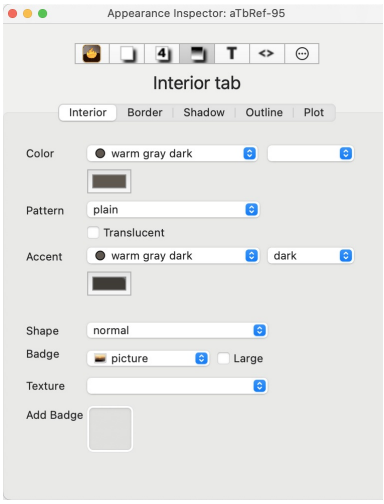


Appearance Inspector

The Appearance Inspector data is selection-scoped, and displays note $DisplayName for the currently selected item; if more than one item is selected it shows the number of items in the selection (e.g. 'Visual Styling' for a single note of that name, but 3 notes selected shows '3 Notes').

The Appearance Inspector has these sub-tabs:

- Interior tab
- Border tab
- Shadow tab
- Outline tab
- Plot tab



Interior tab

This gives easy access for setting a number of appearance-related attributes of notes. The results are applied to all selected note(s).

Color. There are 3 'standard' colour controls that set $Color:

- Defined colour list pop-up menu.
- Colour shade pop-up menu.
- Custom colour picker dialog.

Pattern. Sets $Pattern. Note that the inspector can only set static patterns and not dynamic patterns like bar()/vbar()

- **Pattern** pop-up menu.
- **translucent** option. Sets $Opacity to 50(%).

Accent. There are 3 'standard' colour controls that set $AccentColor:

- Defined colour list pop-up menu.
- Colour shade pop-up menu.
- Custom colour picker dialog.

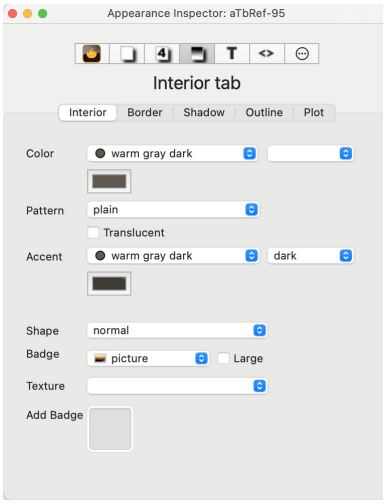
Shape. This sets $Shape to render the note's map icon in one of a number of shapes. Some shapes (such as tag or leaf) have addition attributes controlling the actual shape. These additional settings are stored in other attributes but can all be set manual via controls on the selected note when in map view. The list includes outlines of the basic shapes alongside the actual name, to aid correct selection.

Badge. This sets the $Badge value via the Badge pop-up menu. The pop-up list only shows the default set of icons, showing the badge's icon and the name. Additional user sets can only be (visually) accessed via the badge widget on the selected item in each of the main views. From v9.6.0 this control is wider so the badge's name can be read.

Large (tick box). When ticked a display box of 64x64 pixels is used instead of the default 32x32.

Texture. This sets the $Fill value via the Fills pop-up menu. The list shows both default and user added textures.

Add Badge. An image well accepts a drag-drop of a bitmap image as a custom badge.



Border tab

This gives easy access for setting a number of border appearance-related attributes of notes. The results are applied to all selected note(s).

Color. There are 3 'standard' colour controls that set $BorderColor:

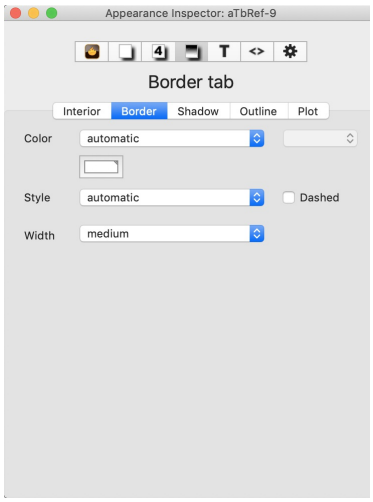
- Defined colour list pop-up menu.
- Colour shade pop-up menu.
- Custom colour picker dialog.

Style. Defines the border's style:

- Border Style pop-up menu: sets $BorderBevel.
- **Dashed** option: toggles $BorderDash between 0 if un-ticked and 5 if ticked.

Width. Sets $Border to preset widths via Border Width pop-up menu (1/2/4). Default is medium (2).

To set no border, select 'none' for the style. Note, that this is not exactly the same as setting both $Border and $BorderBevel.



Shadow tab

This Inspector pane, give easy access for setting a number of border appearance-related attributes of notes. The results are applied to all selected note(s).

Shadow. Options for setting $Shadow to show/hide a shadow.

- **Distance**. Sets $ShadowDistance.
- **Blur**. Sets $ShadowBlur.
- **Color**. There are 3 'standard' colour controls that set $ShadowColor:
  - Defined colour list pop-up menu.
  - Colour shade pop-up menu.
  - Custom colour picker dialog.

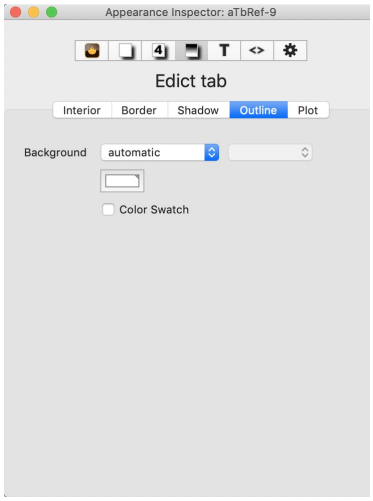
Outline tab

This tab controls aspects of the view of Outline view selected items. It as a single control:

Color. There are 3 'standard' colour controls that set $OutlineBackgroundColor:

- Defined colour list pop-up menu.
- Colour shade pop-up menu.
- Custom colour picker dialog.

Color Swatch. This checkbox shows/hides the note's outline colour swatch. Feature discontinued and removed in in v9.5.2+.



Plot tab

This tab gives controls for setting calculated patterns, such as for container plots.

Pattern. The type of container plot expression. A pop-up list shows the options available. Static patterns are listed but should be ignored as they are not pertinent here.

Expression. The container plot expression (code), i.e. the input argument used by the plot type. The input box and label are hidden if the selected pattern type does not support expression.

X axis. Only visible if xyplot() is the selected plot type. Second input argument for the xyplot() expression.

Minimum. Optional maximum value for the Y axis. Only shown if a dynamic plot type is selected.

Maximum. Optional maximum value for the Y axis. Only shown if a dynamic plot type is selected.

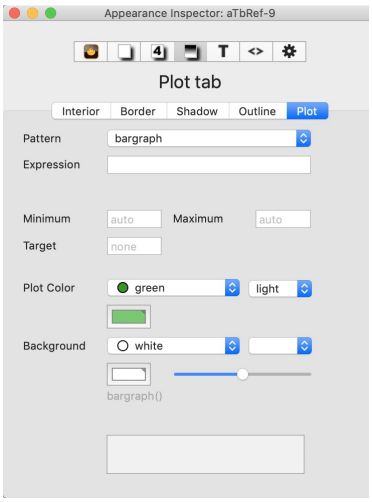
Target. Optional value at which dotted line is drawn in $PlotColor. Only shown if the plot type is bar() or vbar(). Shown for all plot types except pie().

Plot Color. Tinderbox colour controls to set the colour of the plotted data in a container plot ($PlotColor).

Background. Tinderbox colour controls to set the colour of the container plot background for this note's Map ($PlotBackgroundColor). The slider is used to set the opacity of the plot's background; if transparent, the child map shows throug ($PlotBackgroundOpacity).

Defined plot pattern. Grey text caption with the current contents of $Pattern, as defined by the settings above.

Preview window. A preview image of the above settings as a plot.

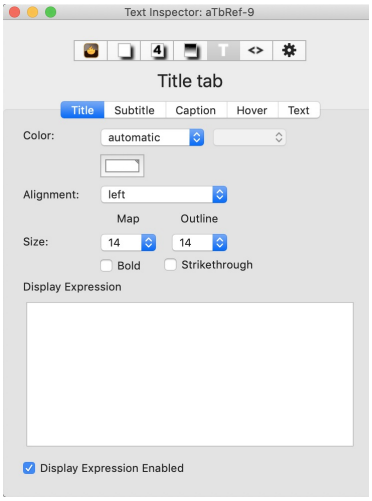


Text Inspector

The Text Inspector data is selection-scoped, and displays note $DisplayName for the currently selected item; if more than one item is selected it shows the number of items in the selection (e.g. 'Visual Styling' for a single note of that name, but 3 notes selected shows '3 Notes').

The Text Inspector has these sub-tabs:

- Title tab
- Subtitle tab
- Caption tab
- Hover tab
- Text tab



Title tab

This tab gives easy access for setting a number of main view title-related attributes of notes. The results are applied to all selected note(s).

Color. There are 3 'standard' colour controls that set $NameColor:

- Defined colour list pop-up menu.
- Colour shade pop-up menu.
- Custom colour picker dialog.

Alignment. Map icon tile alignment options; sets $NameAlignment.

Size. The pop-up shows type point size which sets an underlying attribute value (in brackets). Two similar pop-up lists set notes' title size for Map view ($MapTextSize) and separately for Outline/Chart/Timeline views ($OutlineTextSize). The default is derived from the size being set in the Map Font control Maps pane of Document Settings: 14 point by app default. The stored attribute value is a *percentage* of magnification used to draw note titles, so 14pt maps to 100(%). Thus, if a note's $MapTextSize is 200, it will be twice as large as normally-sized items (e.g. 28pt vs. 14pt). If the underlying attribute is set to any other value than those below, the title's size is scaled accordingly but the pop-up list shows a blank selected value, though the others remain available:

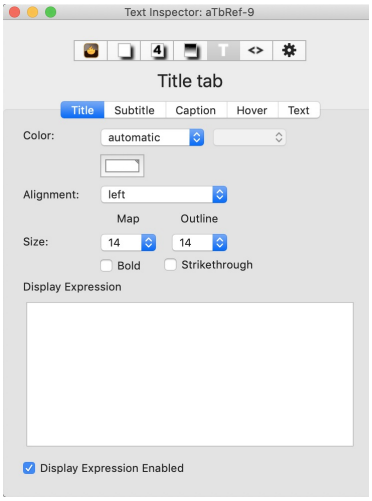
- 9 (stored value = 64, i.e. 64% of default base size)
- 10 (71)
- 12 (86)
- 14 (100) default (set via Doc Settings: see above).
- 16 (114)
- 18 (129)
- 24 (171)
- 32 (229)
- 48 (343)

Bold. This option bolds the title ($NameBold).

Strikethrough. This option set strike-through on the title toggles ($NameStrike).

DisplayExpression. Displays/sets $DisplayExpression.

Display Expression Enabled. Toggles $DisplayExpressionEnabled.



Subtitle tab

This tab controls settings for Subtitles, used in Map view only.

Color. There are 3 'standard' colour controls that set the colour of the subtitle ($SubtitleColor):

- Defined colour list pop-up menu.
- Colour shade pop-up menu.
- Custom colour picker dialog.

Opacity. The slider sets the opacity of the subtitle ($SubtitleOpacity).

Size. The slider sets the text size of the subtitle ($SubtitleSize).

Subtitle. Displays/sets the subtitle text ($Subtitle).

Caption tab

This tab controls settings for Captions, used in Map view only.

Color. There are 3 'standard' colour controls that set colour of the caption ($CaptionColor):

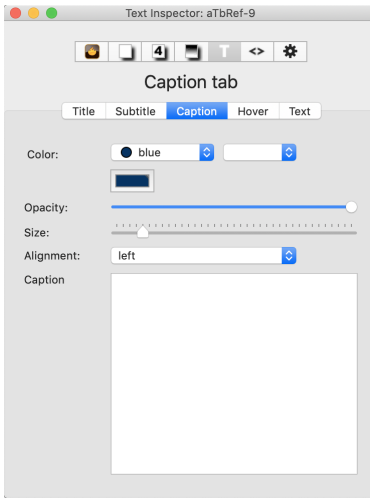
- Defined colour list pop-up menu.
- Colour shade pop-up menu.
- Custom colour picker dialog.

Opacity. The slider sets the opacity of the caption ($CaptionOpacity).

Size. The slider sets the text size of the caption ($CaptionSize) as a relative percentage of map title size ($MapTextSize).

Alignment. Sets the alignment of the caption ($CaptionAlignment).

Subtitle. Displays/sets the caption text ($Caption).



Hover tab

This tab controls settings for Hover Expressions.

Hover Expression displays/sets $HoverExpression.

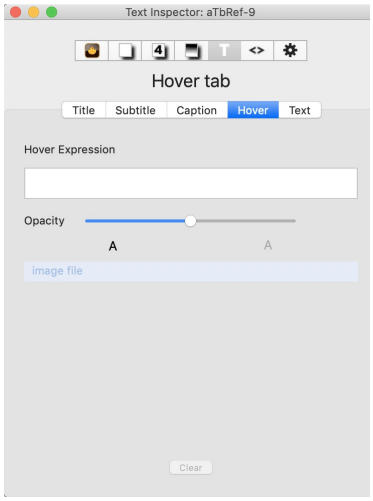
Color. (from v9.6.0) There are 3 'standard' colour controls that set $HoverBackgroundColor:

- Defined colour list pop-up menu.
- Colour shade pop-up menu.
- Custom colour picker dialog.

Opacity. Alters the note's $HoverOpacity.

[**image file**]. Drop a bitmap image here to set $HoverImage. When an image is set, the blue bar shows the path to the image asset.

Clear. Button is only enabled if a $HoverImage is set. Click to clear the existing image.



Text tab

This tab controls settings for note text ($Text). The primary role of the Text pane of the Text Inspector is to change the default text style of *new notes*. In addition, from v9.6.0, using this pane to change the text colour, line spacing, or paragra
spacing immediately changes the text ($Text) of currently-selected notes.

Color. There are 3 'standard' colour controls that set the colour of the note's default text ($TextColor):

- Defined colour list pop-up menu.
- Colour shade pop-up menu.
- Custom colour picker dialog.

If trying to set text colour for pre-existing $Text content, note that above colour controls only configure the note's *default* text colour. To colour all or a selected part of *already-existing* $Text use either the Format ▸ Style or Format ▸ Font menus.

Paragraph Spacing. Pop-up menu controls $ParagraphSpacing. Default is 8pt (set via preferences).

Line Spacing. Pop-up menu controls $LineSpacing. Default is 100%. Set larger percentage values to increase line spacing and vice versa.

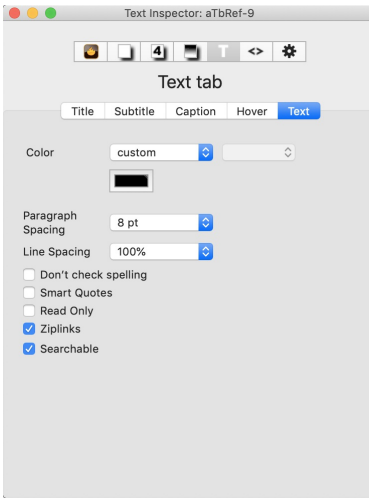
Don't check spelling. Toggles $NoSpelling for selected notes.

Smart Quotes. Overrides default substitution settings for straight-to-curly quote correction ($SmartQuotes).

Read Only. Toggles the note's read-only state (default: un-ticked, read/write), stored in $ReadOnly.

Ziplinks. Toggles the note's ability to create Text link creation via the Ziplinks method (default: ticked, on), stored in $Ziplinks.

Searchable. Toggles the note's searchable status (default: ticked, on), stored in $Searchable.

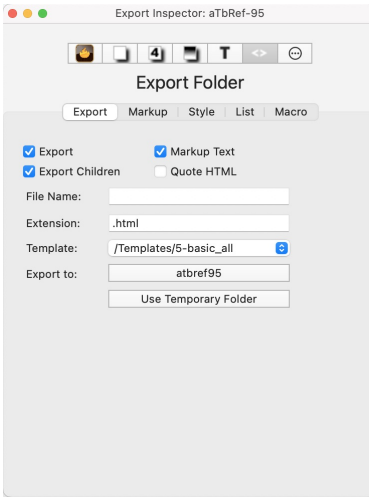


Export Inspector

The Export Inspector data is selection-scoped, and displays note $DisplayName for the currently selected item; if more than one item is selected it shows the number of items in the selection (e.g. 'Visual Styling' for a single note of that name, but 3 notes selected shows '3 Notes'). A minor exception is that the 'Export to' control on the Export sub-tab which works at document scope. In older versions, this Inspector was called the HTML Inspector.

The HTML Inspector has these sub-tabs:

- Export tab
- Markup tab
- Style tab
- List tab
- Macro tab



Export tab

The settings in the Export tab control whether notes export, their templates and the export location. All are selection-scope except 'Export to' that applies to all notes in the document.

Export. Controls whether the current note is exported when a file level 'Export as HTML…' operation is invoked. If this box is checked, this note will be exported to HTML when you export the document. Having this box ticked is equivalent to setting the $HTMLDontExport attribute to `false`.

Export Children. Controls whether the current note's children are exported when a file level 'Export as HTML…' operation is invoked. If this box is checked, this note's children will be exported to HTML when you export the document. This box being ticked is equivalent to setting $HTMLExportChildren to `true`.

Markup Text. If this box is checked (`true`), Tinderbox's export to HTML attempts to translate text styles, such as italics, boldface, and relative size, as well as elements that look like headings or lists, to similar text styles using HTML mark-up. If this box is unchecked, Tinderbox exports the text of the note without any added HTML mark-up codes. This box being ticked is equivalent to setting $HTMLMarkupText to `true`.

Quote HTML. If ticked (`true`), any HTML markup detected in the note is converted to HTML entities on export so it is seen on screen as code instead of being rendered as HTML code. This box being ticked is equivalent to setting the $HTMLQuoteHTML attribute to `true`.

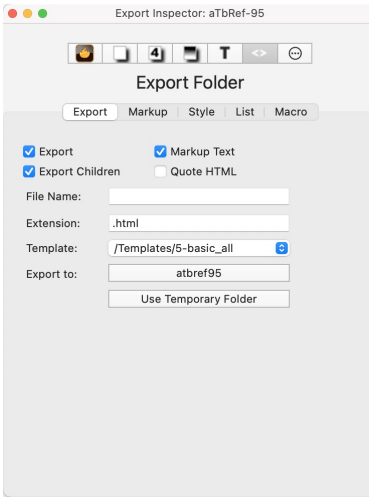
File Name. When Tinderbox exports a note to HTML, it automatically generates a name for the HTML file base on the note's title. If you want a note's file to have a specific name—so that you can know what name to link to from another web site, for instance—enter a name here. An entry here is equivalent to setting an $HTMLExportFileName attribute value; if no name is pre-set, the name is generated on the fly when exporting, based on the note name, $HTMLFileNameLowerCase, $HTMLFileNameMaxLength and $HTMLExportExtension attribute settings (if not modified on the HTML view itself).

Extension. The file extension suffix that should be added to the filename for the HTML file: the default is normally '.html'. Setting a non-default value is equivalent to setting the $HTMLExportExtension attribute. Note the value should include a period as the first character.

Template. The Template pop-up list shows the available templates within the document. The default is 'none' but as soon as a single template is added, thus becomes the default. The chosen value is stored in $HTMLExportTemplate.

Export to. This opens a folder chooser dialog and allow the export folder to be used when exporting the whole document.

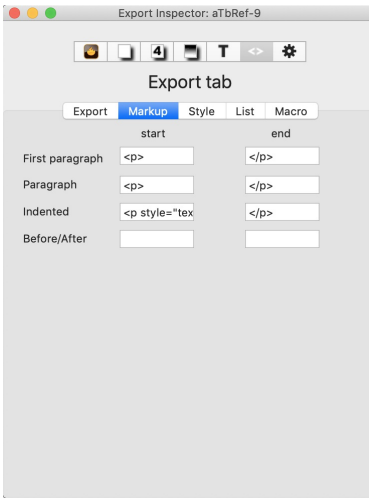
A second, new (v9.5.0) button allows (re-)selection of the hidden internal location used for zero-configuration internal preview.



Markup tab

These settings in the Paragraph tab control the markup of HTML text paragraphs:

- First paragraph. These fields specify what HTML markup should surround the first paragraph of text when the text of this note is exported to HTML. They default to `<p>` and `</p>`. Stored in $HTMLFirstParagraphStart and $HTMLFirstParagraphEnd attributes.
- Subsequent paragraphs. Defaults are as above, but for all other paragraphs of text in the text of the note. Stored in $HTMLParagraphStart and $HTMLParagraphEnd attributes.
- Indented. As above but for all paragraphs beginning with a tab character; such paragraphs will use `<blockquote>`. Stored in $HTMLIndentedParagraphStart and $HTMLIndentedParagraphEnd.
- Before and after note. Empty by default, this is what is exported immediately before a note's opening paragraph mark-up and after a note's final paragraph market, respectively. In effect, mark-up to enclose ^text^. Can be useful in connection with CSS stylesheets, such as setting before and after to `<div>` and `<div>`. Stored in $HTMLExportBefore and $HTMLExportAfter attributes.



Style tab

The settings in the Style tab control styling of HTML text.

Bold. The start and end fields hold the opening and closing tags for bold text and default to `<b>` and `</b>`. Stored in $HTMLBoldStart and $HTMLBoldEnd.

Italic. The start and end fields hold the opening and closing tags for italic text and default to `<i>` and `</i>`. Stored in $HTMLItalicStart and $HTMLItalicEnd.

Underline. The start and end fields hold the opening and closing tags for underlined text and default to `<u>` and `</u>`. Stored in $HTMLUnderlineStart and $HTMLUnderlineEnd.

Strike through. The start and end fields hold the opening and closing tags for struck-through text and default to `<strike>` and `</strike>`. (Equivalent to setting the $HTMLStrikeStart and $HTMLStrikeEnd).

Code. The start and end fields hold the opening and closing tags for $Text using the 'Code' font style. They default to `<code>` and `</code>`. Stored in $HTMLCodeStart & $HTMLCodeEnd.

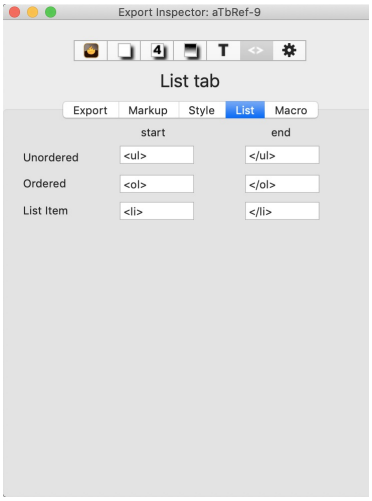
List tab

The settings in the List tab control styling of lists in HTML text.

Unordered. The before and after fields hold the opening and closing tags for unordered (bulleted) lists and default to `<ul>` and `</ul>`. Stored in $HTMLListStart and $HTMLListEnd.

Ordered. The before and after fields hold the opening and closing tags for ordered (numbered) lists and default to `<ol>` and `</ol>`. Stored in $HTMLOrderedListStart and $HTMLOrderedListEnd.

List item. The before and after fields hold the opening and closing tags for an italic passage and default to `<li>` and `</li>`. Stored in $HTMLListItemStart and $HTMLListItemEnd.



Macro tab

The settings in the Macro tab control the creation and storage of macros both for HTML export and for use within the current document for the current document.

Gear wheel button (at left):

- **Delete macro**. Deletes the currently selected macro. *There is no undo for this option* .
- **New user attribute**. Press this option to add a new macro. A new macro 'newMacro' is added to the macro list and selected ready for editing.

Macro list. A pop-up menu lists all currently defined macros. Select an item to load its details for review or edit. The name of the selected macro is shown in the **Macro name** box (below). The code inserted by the selected macro is shown i the **Macro code** box (below)

Macro name. The name of the currently selected macro.

Macro code. Holds the code to be evaluated when the Macro is invoked. See the article on macros to explain the process.

Action Inspector

The Action Inspector data is selection-scoped, and displays note $DisplayName for the currently selected item; if more than one item is selected it shows the number of items in the selection (e.g. 'Visual Styling' for a single note of that name, but 3 notes selected shows '3 Notes').

The Action Inspector has these sub-tabs:

- Query tab
- Action tab
- Rule tab
- Remove tab
- Edict tab
- Sort tab

Query tab

The Query tab is used for agents (and smart adornments) only.

The main input box is used for adding or editing a query ($AgentQuery). Pressing the **Return** key (↩) commits any changes made and runs (or updates) the query. The code box has auto-completion for action code and attribute names (based on using a $-prefix). The code box supports syntax colouring.

Code fields do not select the entire text after when gaining focus or after pressing the **Return** key (↩) to update (save changes). This lessens the chance of accidental deletion of existing code.

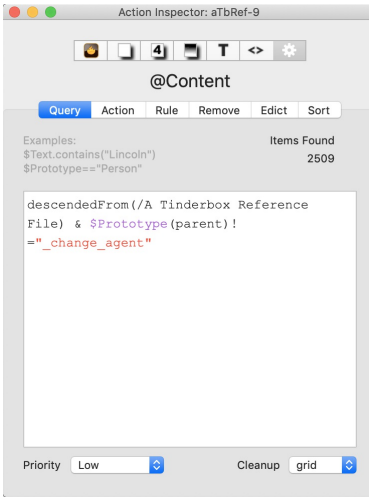
The number of items currently matching the query, i.e. the child count of the agent is show at top right.

Priority. This pop-up menu controls whether the agent is 'on' or not. Sets $AgentPriority. See more on controlling agent priority.

Cleanup. This pop-up menu sets the agent's cleanup action ($CleanupAction).

Smart adornments use neither a Priority or Cleanup actions so both these pop-ups are greyed out if a smart adornment is selected.

Note that an agent's query, action and priority can also be set from the agent tab of the Get Info pop-over.



Action tab

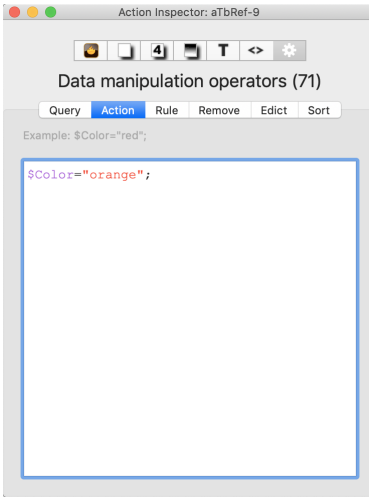
The Action tab is used to set the OnAdd ($OnAdd) action of notes and adornments or the agent action ($AgentAction) of agents.

The main input box is used for adding or editing the action code. Pressing the Return key (↩) commits any changes made and runs (or updates) the query. The code box has auto-completion for action code and attribute names (based on using a $-prefix). The code box supports syntax colouring.

Code fields do not select the entire text after when gaining focus or after pressing the **Return** key (↩) to update (save changes). This lessens the chance of accidental deletion of existing code.

Note that an agent's query, action and priority can also be set from the agent tab of the Get Info pop-over.

OnAdd also affects adornments created or pasted into containers.



Rule tab

The Rule tab is used to set the rule ($Rule) action of notes, adornments and agents.

The main input box is used for adding or editing the action code. Pressing the Return key (↩) commits any changes made and runs (or updates) the query. The code box has auto-completion for action code and attribute names (based on using a $-prefix). The code box supports syntax colouring.

Code fields do not select the entire text after when gaining focus or after pressing the **Return** key (↩) to update (save changes). This lessens the chance of accidental deletion of existing code.

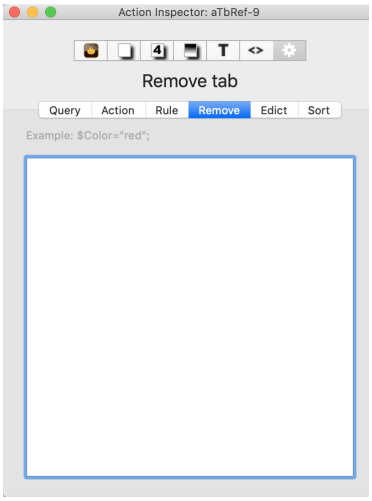
Inheritance state. A caption above and right of the code box indicates the inheritance state of the rule. Captions are:

- **default**. Uses document default.
- **inherited**. Uses an inherited (prototype) value.
- (no caption). The rule is set locally in the note.

Enabled. This toggles disabling of the rule ($RuleDisabled). When un-ticked the rule's code is not evaluated as part of the rule cycle.

Run Now button. This runs the rule immediately, once: the rule is added as a single task to the pending work queue. If the rule is disabled, this button is disabled. The button also updates the Displayed Attributes table.

Remove tab



The Action tab is used to set the On Remove ($OnRemove) action of notes and adornments. For agents, this action is performed on the *original* of the agent alias.

The main input box is used for adding or editing the action code. Pressing the Return key (↩) commits any changes made and runs (or updates) the query. The code box has auto-completion for action code and attribute names (based on using a $-prefix). The code box supports syntax colouring.

Code fields do not select the entire text after when gaining focus or after pressing the **Return** key (↩) to update (save changes). This lessens the chance of accidental deletion of existing code.

Note that for agents this action cannot be set from the 'agent' pane of the Get Info pop-over.

Immediately before a note is deleted, the OnRemove action of its parent container is performed. If the note lay on an adornment, the adornment's OnRemove action is performed. In the actions, the effect is bound to the note that will be removed.

Edict tab

The Edict tab is used to set the edict ($Edict) action of notes, adornments and agents.

The main input box is used for adding or editing the action code. Pressing the Return key (↩) commits any changes made and runs (or updates) the query. The code box has auto-completion for action code and attribute names (based on using a $-prefix). The code box supports syntax colouring.

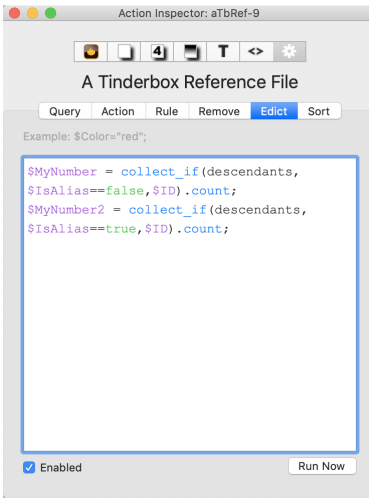
Code fields do not select the entire text after when gaining focus or after pressing the **Return** key (↩) to update (save changes). This lessens the chance of accidental deletion of existing code.

Inheritance state. A caption above and right of the code box indicates the inheritance state of the edict. Captions are:

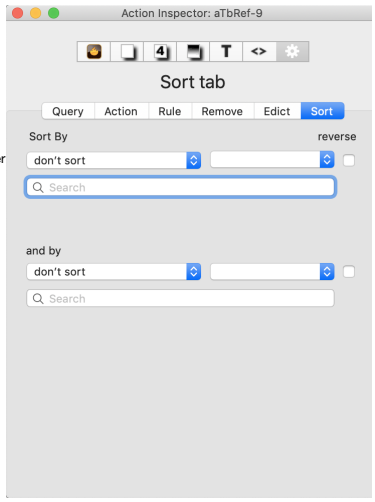
- **default**. Uses document default.
- **inherited**. Uses an inherited (prototype) value.
- (no caption). The edict is set locally in the note.

Enabled. This toggles disabling of the edict ($EdictDisabled). When un-ticked the edict's code is not evaluated as part of the rule cycle.

Run Now button. This runs the edict immediately, once: the edict is added as a single task to the pending work queue. If the edict is disabled, this button is disabled. The button also updates the Displayed Attributes table.



Sort tab



The sort tab is used to control the sorting of containers, adornments and agents.

Sort by / (transform type) / reverse . Optionally, sets a sort order on any attribute value (pop-up menu), a transform on the order (pop-up menu), with the sub-option of reversing that order (tick-box). Values are persisted in the agent's $Sort, $SortTransform and $SortBackward attributes. This may be edited.

and by / (transform type) / reverse . Optionally, sets a sort order on any attribute value (pop-up menu), a transform on the order (pop-up menu), with the sub-option of reversing that order (tick-box). Values are persisted in the agent's $SortAlso, $SortAlsoTransform and $SortBackwardAlso attributes. This may be edited.

Both sets of selector controls offer a **Search** box to assist in quickly locating the desired attribute on which to sort. Entering "none" in the search field, or clearing the search field and pressing the Return key (↩), will remove any local value for the $Sort attribute thus restoring the inherited or default value if any.

Sort order depends on the data type of the attribute being used for sort, e.g. lexical vs. numerical vs. date, etc. In languages using accented characters, relative lexical sort order of accented characters may vary by locale.

Once set, Tinderbox continues to sort new items to reflect that setting. However, it is often the case that contents are not under constant change and all that is needed is a one-off sort. To do this, select the desired sort attribute, wait a few moment for the sort to occur and then return the setting to do not sort.

Sorting agents to reflect the relative outline location of original notes: to do this set the agent's transform type to ' **original note**' and leave sort as '**don't sort**'.

When a container's sort method is **don't sort**, the **reverse** sort flag is ignored.

The meaning of the styling of different listed attributes (bold, strikethrough, etc.) is explained here.

Dialogs

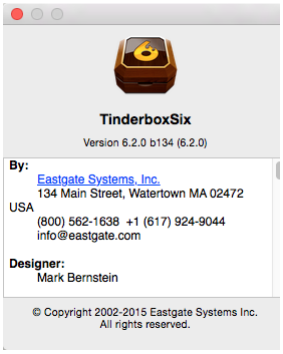
The Preference dialog panes are described here. View windows are described here. All other dialogs are described below.

To close pop-over dialogs, either click outside the dialog or use the **Escape** key (⎋ or 'Esc').

When clicking outside the dialog, do not click on the calling object. Thus, if opening a pop-over from the text pane, click in the view pane. For a pop-over in the view pane click the text pane or—for *some* pop-overs—anywhere in the view except the currently selected note.

About Tinderbox dialog

This dialog shows the application's 'About' dialog and information about the application.

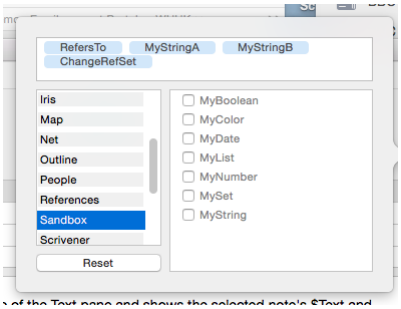


Add Displayed Attributes pop-over

This pop-over allows the Displayed Attributes table for the current note to be defined (or re-edited; attributes can be added to or deleted from the table and their order changed. The pop-over is invoked either from the head of the text pane's text tab or via the View menu.

To add an attribute, either:

- Type the desired name in the box in the upper section of the pop-over. Autocomplete is offered for all currently defined system and user attributes. The box is blank if no Displayed Attributes have yet been defined. If large numbers of attributes are added, the box will auto-expand to keep existing attribute choices on view.
- use the two bottom lists. First select the attribute group in the left pane then tick the appropriate attribute in the right pane (un-ticking will also remove an existing item).



To delete a current Displayed Attribute, click to select the item in the top list and delete it. Or, use the lower two panes to find the item and un-tick it.

To re-order Displayed Attributes, click on an item in the upper list box and drag it to the correct location in the list.

To remove all current Displayed Attributes (and reset inheritance of $DisplayedAttributes) click the **Reset** button. The Reset button removes the viewed note's immediate Displayed Attributes settings, so that the inherited or default Displayed Attributes will be used. (Formerly, it could copy those Displayed Attributes as an immediate value, temporarily interfering with inherited changes to the prototype or default.)

To close the pop-over click outside it or press Escape. On closing, the changes made (above) are effected. If the top box contains a name that does not match a currently defined attribute, the Define new Displayed Attributes pop-over will then open.

The Displayed Attribute picker ignores proposed Displayed Attribute names that cannot refer to attributes, such as "3cats" and "My sprocket" that may result from mistyping attribute names.

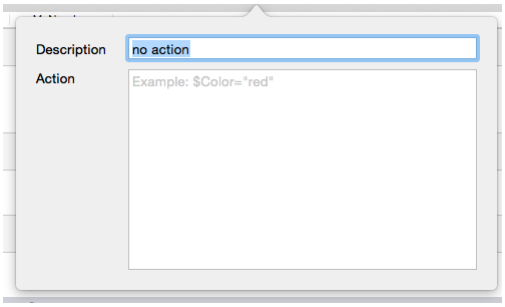
The meaning of the styling of different rows (bold, strikethrough, etc.) is explained here.

Attribute Browser Action pop-up

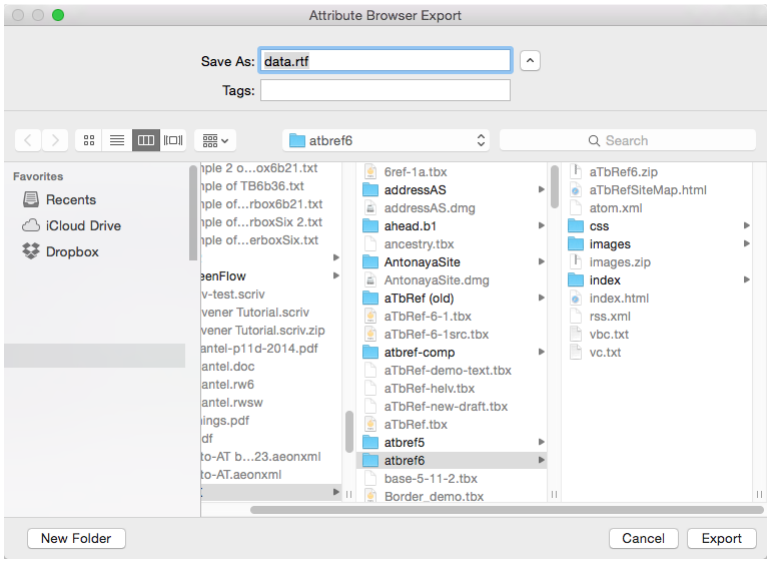
Called from the Attribute Browser controls, this pop-up is used to configure an optional query.

Description: The description entered here is used as a label on the main control panel.

Action: Enter any action code expression, as with a rule or agent action.



Attribute Browser Export



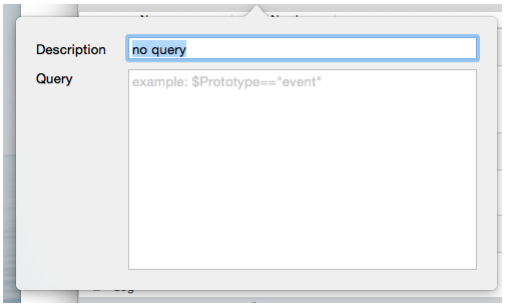
A standard OS folder dialog called when exporting an Attribute Browser view.

Attribute Browser Query pop-up

Called from the Attribute Browser controls, this pop-up is used to configure an optional query.

Description: The description entered here is used as a label on the main control panel.

Query. Enter a valid query such as for an agent.



Badge pop-up picker

Called from the badge control of main view note icons, this picker gives a visual picker for available badges. The badge selection is stored, via its string name, in $Badge.

The range of badges shown depends on the built-in sets and any added by the user. The pickers tabs are (left to right):

- User. Any custom badge artwork placed directly into the 'badges' folder within Tinderbox's application support folder.
- Classic. The primary set of badges (default tab selection)
- Avatars. Built-in set.
- Icons. Built-in set.
- Symbols. Built-in set.
- User folders… Any discrete folders of custom badges placed into the 'badges' folder within Tinderbox's application support folder. The tab name is the folder name.

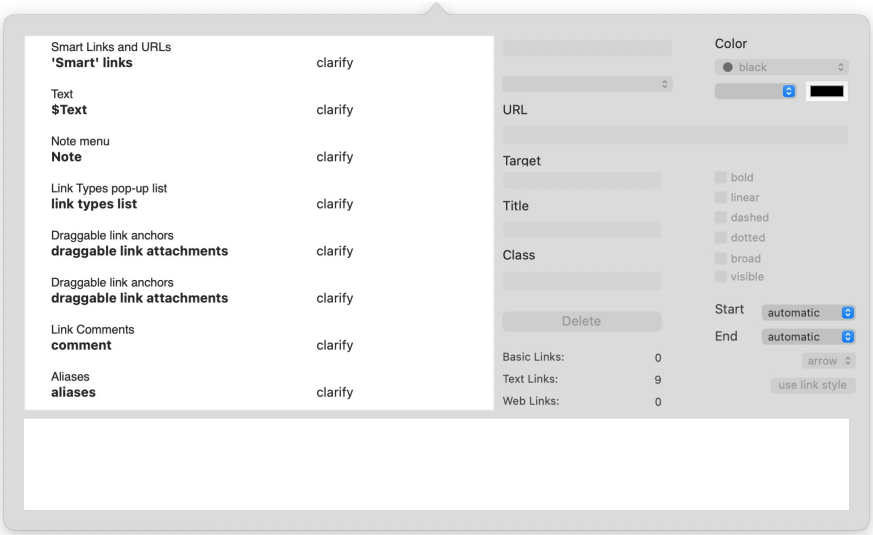
Click on any badge to set the note's $Badge. Clicking the first (blank) tile in any set resets the Badge to the default inherited state.

Place the mouse cursor over any icon, without clicking, and after a brief pause the icon's name is shown as a tool-tip. Names are case-sensitive and all built-in badges use all-lowercase.

To close the pop-over click anywhere outside it.

The Badge Picker adopts the prevailing background colour in dark mode. Monochrome icon families are drawn in white over a dark background, and in black over a light background; colour families are drawn normally.

Browse Links pop-over



The Browse Links pop-over shows all the links originating from the currently selected note (with the exception of 'Smart' links) and offers the ability to edit their details. In v9.5.0, the layout was adjusted for improved legibility. This pop-over cannot be 'torn off' so is effectively a modal dialog. For every link originating from the selected note, the left-hand listing shows:

- The display name of the destination note, or '(web link)' for web links from $Text.
- The anchor text in $Text (text links only).
- The link type (bottom right).
- Mouseover (either link destination or anchor text) shows the full path ($Path) of the destination note.

Any link selected in the list has its details shown in the lower panel.

Links may be dragged to reorder them. This is useful to set the 'first' link in the list, as used by the 'Navigate' feature in the Note menu.

All basic links list before all web links and then all text links. Text links are listed in order of text offset, i.e. where they occur in the flow of text. Thus the first text link in $Text links first. This makes finding the correct link much easier in a well-linked page's listing. Links to or from aliases are shown in italics.

Double-click any link to select its destination and dismiss the popover. If invoked from a view pane, the view will be scrolled or refocused if necessary to locate the newly-selected note. For to-text links this scrolls the destination note to the lin point and highlights the linked word.

When a text link is selected the text pane scrolls to make the link anchor visible and the link anchor is temporarily highlighted.

The middle column of controls are the same as for creating links:

- **link type list**. Pick a link type (or none) from the pop-up link types list. To create a new link type, first use the Links Inspector to add the new type to the list. This field is disabled when no link is selected, but otherwise lets a link type value be typed to select it, or to create a new link type (without having to define it first). From v9.5.0, changing the link type here now fires the link's OnLink action, if any.
- **URL**. The destination URL for the link, either for import or display in a browser. This can in fact be FTP, mailto or various other online URL types.
- **Target**. The name of the window to use. Pertinent for framed web sites, or if you wish the exported HTML to call the link in a separate window. You do not see these names but your browser does and if the window cited is already open, then that window is re-used. An easy method for naming the 'target' is to always use the value 'tbx'. As it is unlikely another application will spawn a window called 'tbx' only your Tinderbox site's external links will share a second common window.
- **Title**. This sets the link's HTML 'title' attribute which may (depending on browser) type be shown in a status bar or mouse-over of HTML.
- **Class**. The name of a CSS style sheet class to be applied to the link when in HTML form. This is useful if you would like different link types (or just arbitrary collections of links) to have different web styling.
- **Delete button**. Deletes the currently selected link.

The right column of controls allow most aspects of this particular link to be customised. Only the link label text, label visibility and link line visibility apply to all links of a given type. The controls are:

- **Color**. Standard trio of controls for colour: a pop-up list of defined colours, a colour shade pop-up and a colour chip control.
- **line style tick-boxes**. Dashed and dotted can be combined to give a dot-dash-dot style of line.
- **link terminator list**. The shape used at the destination end of the link. The current choices are 'arrow' (default) or 'circle'.
- **link start position**. Click the pop-up to set from which map icon face (automatic/top/right/bottom/left) the link is drawn. Default is automatic. See more on draggable link attachments.
- **link termination position**. Click the pop-up to set at which map icon face (automatic/top/right/bottom/left) the link is terminated. Default is automatic. See more on draggable link attachments.
- **use link style button**. Clicking this button resets the above controls to the defaults for the overall link link, i.e. removing per-link customisation.
- **visibility tick-box**. Link visibility can be controlled for individual links as well as through link types. If a link type is not visible, all links of that type are hidden. If a link type is visible, then individual links may be hidden or shown.
- **link comment box**. If the selected link has a comment, it can be viewed or edited in this box.

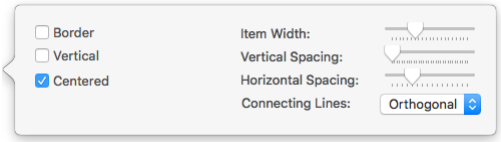
Note that aliases can support their own outbound *basic* links ($Text links are shared with the original). In such a case, the link listing of the Browse Links may differ when open from the alias or from the original.

To dismiss the pop-over, click anywhere outside the pop-over.

To view a note's links when a note no longer has focus, open its Roadmap and tear-off the pop-up as a stand-alone window (that will last for the rest of the current session).

Chart Settings pop-over

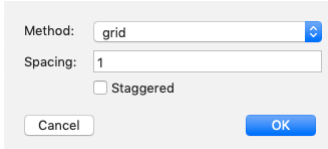
This is the configuration pop-up for the current Chart view:



- **Border**. Default: un-ticked. If ticked a thin border is drawn around each item on the chart.
- **Vertical**. Default: un-ticked. If tick-ed, the chart is drawn from top-to-bottom of the screen as opposed to the default left-to-right layout
- **Centered**. Default: ticked. If un-ticked, the item top aligns (default layout) or left aligns (vertical layout) with the line from the parent item.
- **Item Width**. Scrubber control sets the width of chart items
- **Vertical Spacing**. Scrubber control sets the amount of vertical spacing between chart items.
- **Horizontal Spacing**. Scrubber control sets the amount of horizontal spacing between chart items.
- **Connecting Lines** pop-up:
  - **Orthogonal**. (Default) Lines interconnecting items are drawn using horizontal and vertical sections.
  - **Straight**. Lines interconnecting items are drawn using straight lines.
  - **Curved**. Lines interconnecting items are drawn using bezier curves.

Cleanup view tab

A view tab called from the Arrange menu. The cleanup is effected on the map view selection, if 2 or more notes are selected, otherwise the whole map is cleaned up. Use Undo (Cmd+Z) if you make a mistake. The clean-up layout is anchored on the map position of the note with the lowest $OutlineOrder value. Options are:



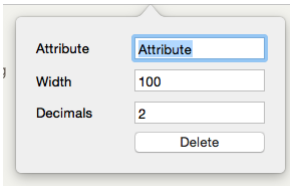
- **Method**. This opens a pop-up:
  - **To Grid**. (default) Rearranges all the notes in the map into a rectangular grid.
  - **To Row**. Arranges all icons in a single horizontal row.
  - **To Column**. Arranges all icons in a single vertical column.
  - **To Box**. Arranges all notes in an open rectangle. Best with more than 8 notes.
- **Spacing**. (default: 1) This is the spacing in map units between the nearest edges of items. Thus, in a row layout, the left edge of the second item with be 1 map unit to the right edge of the first item.
- **Staggered**. (default: not ticked) Ticking this box produces a staggered layout if 'to grid' is selected above. It has no effect with other layout methods. Tthis control is only enabled when the grid layout method is selected.

On completion of a cleanup operation the map will be scrolled, if necessary so that some notes are in view. If the whole map is being re-arranged, this will normally be the top left corner of the map.

Column view, column format pop-up

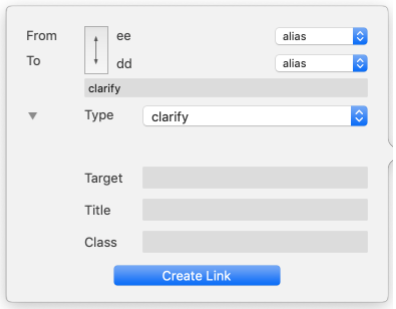
This pop-up is displayed when a column head in Column view is clicked. There are 4 controls:

- **Attribute**. Default value: 'Attribute'. Edit this to the name of any existing attribute, but do not use a $-prefix. Auto-complete is provide to aid completion. It is possible to enter a value for an attribute that does not currently exist, as with the default value, in which case no data will be displayed.
- **Width**. Default: 100. Width of the column (in pixels).
- **Decimals / Format** (if attribute is Date-type). Default: 2. The number of decimal places to use. For Date-type attributes, date format strings can be used. By default, date attributes show the user's locale's short-form date only. Booleans are shown as tick boxes.
- **Delete**. Click to delete the current column.



Create Link from alias pop-over

This is the dialog is shown after dragging a link from an aliases's link widget or from a link park. If opened by accident it can be closed by the Escape key (⎋).



For internal links there are a minimum two verification pieces of information and one setting to make.

From. The source note, set automatically. A pop-up allows the link to originate from the current **alias** or the alias' **original** note. A pop-up allows the link to originate from the current alias or the alias' original note.

To. The destination note; omitted for web links. A pop-up allows the link to terminate at an **alias** or the alias' **original** note.

Link direction button. To reverse the direction of the link as shown, click this button; the to/form labels are reversed and the link will be created in the reverse direction. The button is a toggle, a second press will restore the original direction.

Type. Either select a type from the link types pop-up list, or type a new one into the box. New types are automatically added to the list of defined link types for the document. The last used link type value is remembered for subsequent links during the document's current edit session and reset when the file in next opened.

URL. This input is only displayed for web link creation, otherwise this space on the dialog is left blank. The destination URL for the link. This can in fact be FTP, mailto or various other online URL protocols.

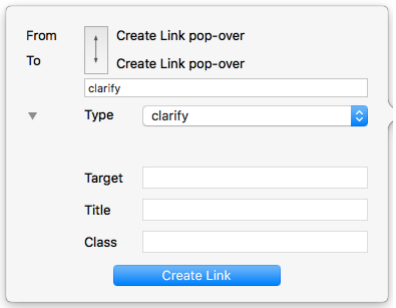
Disclosure triangle control. (Left of **Type** label). Shows/hides the following (hidden by default):

- **Target**. The name of the window to use. Pertinent for framed web sites, or if you wish the exported HTML to call the link in a separate window. The easy method for the latter is to always use the value 'tbx'. You do not see these names but your browser does and if the window cited is already open, then that window is re-used as it is unlikely another application will spawn a window called 'tbx'. Only your Tinderbox site's external links will share a second common window.
- **Title**. This sets the link's HTML 'title' attribute which may be shown in a status bar or mouse-over of HTML depending on your type of web browser.
- **Class**. The name of a CSS style sheet class to be applied to the link when in HTML form.

Create Link button. Creates the link using the details set in the dialog.

Create Link pop-over

This is the dialog is shown after dragging a link from a note's link widget or from a link park. If opened by accident it can be closed by the Escape key (⎋). The initial keyboard focus is set to the destination name field if a new destination note is being created. Otherwise, the initial keyboard focus remains on the **Create Link** button.



For internal links there are a minimum two verification pieces of information and one setting to make.

From. The source note, set automatically.

To. The destination note; omitted for web links.

Link direction button. To reverse the direction of the link as shown, click this button; the to/form labels are reversed and the link will be created in the reverse direction. The button is a toggle, a second press will restore the original direction.

Type. Either select a type from the link types pop-up list, or type a new one into the box. New types are automatically added to the list of defined link types for the document. The last used link type value is remembered for subsequent links during the document's current edit session and reset when the file in next opened.

URL. This input is only displayed for web link creation, otherwise this space on the dialog is left blank. The destination URL for the link. This can in fact be FTP, mailto or various other online URL protocols.

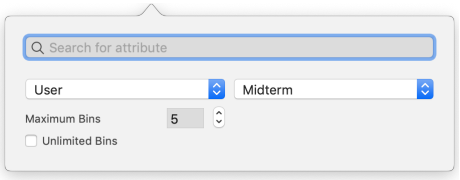
Disclosure triangle control. (Left of **Type** label). Shows/hides the following (hidden by default):

- **Target**. The name of the window to use. Pertinent for framed web sites, or if you wish the exported HTML to call the link in a separate window. The easy method for the latter is to always use the value 'tbx'. You do not see these names but your browser does and if the window cited is already open, then that window is re-used as it is unlikely another application will spawn a window called 'tbx'. Only your Tinderbox site's external links will share a second common window.
- **Title**. This sets the link's HTML 'title' attribute which may be shown in a status bar or mouse-over of HTML depending on your type of web browser.
- **Class**. The name of a CSS style sheet class to be applied to the link when in HTML form.

Create Link button. Creates the link using the details set in the dialog.

Crosstabs view axis configuration pop-over

This pop-over dialog is shown when clicking either the button to the right of either the **Rows** or **Columns** labels on the Crosstabs view toolbar. The default Row attribute is $Name, and for columns it is $Height.



Search box. Type the name of an attribute (with auto-completion) to locate the group/name of the desired attributes. Pressing Return with a valid attribute name will correctly configure the group and name pop-ups below.

Maximum Bins. Either type a number in the box (between 1 and 24) or use the up/down control to set the desired number of bins for distribution of the results in the Crosstabs matrix. This control and its label are hidden if/when the control below is ticked.

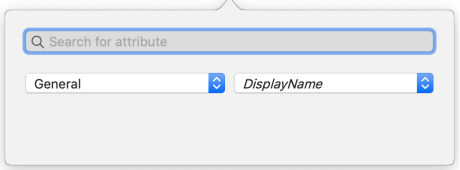
Unlimited Bins. Tick this to override the above control and to show as many bins as there are discrete values (e.g. for String-based attributes). Ticking this control temporarily hides the Maximum Bins control.

Numbers of bins. For some data types there are additional default behaviours:

- Boolean attributes always have exactly two bins.
- Set and List attributes always have unlimited bins, allowing each tag or element its own bin.
- Date attributes: if the date "never" is present, it is always placed in its own bin.

Crosstabs view secondary configuration pop-over

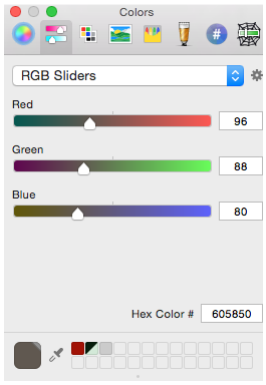
This pop-over dialog is shown when clicking either the button to the right of either the **Display** or **Bar** labels on the Crosstabs view toolbar. The default **Display** attribute is $DisplayName, and for **Bar** it is $OutlineOrder. The **Bar** attribute must always be numeric: the Bar selection offers autocompletions only for numeric attributes.



Search box. Type the name of an attribute (with auto-completion) to locate the group/name of the desired attributes. Pressing Return with a valid attribute name will correctly configure the group and name pop-ups below.

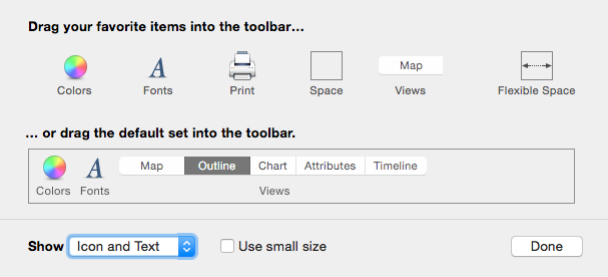
Custom Color colour picker dialog

Most colour menus provide a clickable colour swatch which both displays the chosen colour and, when clicked, lets you select any colour you wish. If no colour is set the colour chip defaults to a mid-grey.



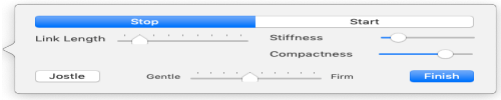
Customize Toolbar panel

A self-explanatory panel used for customising the toolbar on document windows.



Dance pop-up

Called from the View > Arrange menu. This control panel provides controls over the "dancing" force-directed layout algorithm.

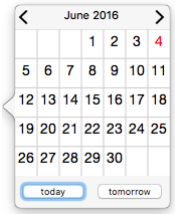


- **Stop/Start** toggle buttons. This turns the dance on/off, using the current settings as set below.
- **Link Length**. This controls the distance between linked notes.
- **Stiffness**. This controls the tension of the spring between linked notes.
- **Compactness**. This controls the distance at which unlinked notes neither attract nor repel each other. Even unlinked notes attract each other at long distances. Notes also repel each other at short distances.
- **Jostle**. This adds some random motion to each note, and continues to add decreasing amounts of random motion for the next several seconds. The magnitude of the random motion can be adjusted from *gentle* to *firm*.
- **Finish**. This button closes the dialog. It does not cause the current setting to be re-applied, i.e. it closes the dialog without further 'dance' actions.

Date picker pop-over

If an attribute of the 'Date' data type shown in Displayed Attributes, the Displayed Attributes picker and the Get Info attributes tab will show a date-picker widget that opens a date-picker pop-over.

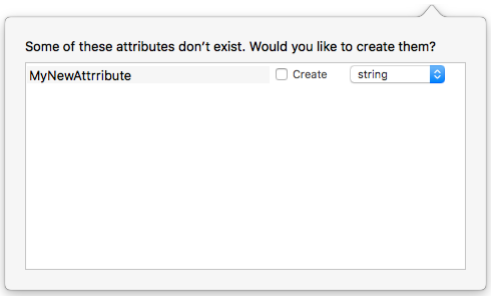
The pop-over shows the days of the current month (as per the user's OS locale), with today marked in red. Clicking any day will set that day, with current system time, for the attribute. The top control allows the dates shown to be shifted earlier (left) or later (right) in one month increments. Special buttons are provided to allow quick setting of today or tomorrow as the attribute's value.



Define new Displayed Attributes pop-over

If an unidentified attribute is specified in the Add Displayed Attributes pop-over, this pop-over is shown. For each undefined item if it is a valid attribute name it is listed with a pair of controls, or else it is ignored.

For each listed item 2 controls are shown:

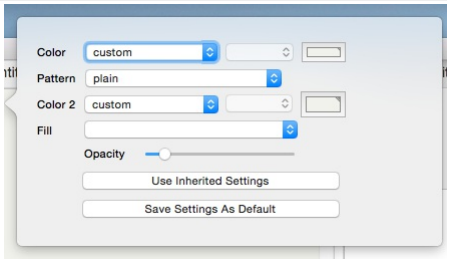


- **Create**. If ticked, a new attribute of this name is created when the pop-over is dismissed and added to the current note's Displayed Attributes. If left un-ticked (default) a new attribute is not created.
- pop-up list. This lets the user choose the data type of attribute to create. The default is to create a String data type.

Clicking outside the pop-over or pressing Escape will close the pop-over and create any ticked items. Leave all items unticked if no new attributes are desired.

Edit Background dialog

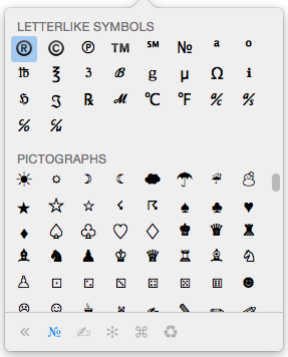
This pop-over aids customisation of main view backgrounds:



- **Color**. A standard trio of colour controls allow setting of $MapBackgroundColor. This forms the main background colour of main views.
- **Pattern**. A list of static patterns is used to set $MapBackgroundPattern.
- **Color**. A standard trio of colour controls allow setting of $MapBackgroundAccentColor. This is used as the accent colour in the background if a pattern is set.
- **Fill**. A texture list allows setting of $MapBackgroundFill, using a fill texture instead of a pattern or plain background.
  - **Opacity**. Sets the fill opacity ($MapBackgroundFillOpacity).
- **Use Inherited settings**. Restore the current background to document default settings.
- Save Settings As Default. This does the reverse of the above and uses the current background settings to set them as the document default settings.

Emoji & Symbols pop-over

This pop-over (called 'Special Characters', pre-Yosemite) gives easy access to a visual picker for emoji symbols and special non-letter characters.



Error List pop-over

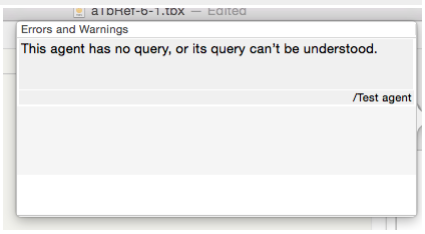
When action code errors occur that Tinderbox wants to bring to your attention, an orange marker is shown in the top left corner of the Text pane's title bar.

There is no exact listing of what triggers a listing, but primarily the feature exists to flag up un-parsable agent queries and action or similar rules and edicts.

The pop-over shows a list of error items. Each item describes and error and then path of the note to which it refers.

If an error is resolved, e.g. a broken query is fixed, it is removed from the list. If the entire error queue is emptied the orange marker is removed.

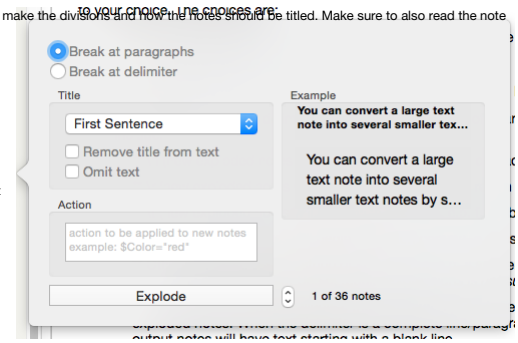
Double-clicking on an item causes it to clear from the list, which is useful if there are several entries or if the user wishes to ignore the error and not be reminded.



Explode pop-over

This dialog is called from the Note menu. It allows you to split a large text file into several smaller text notes and to specify both where Tinderbox should make the divisions and how the notes should be titled. Make sure to also read the note on Exploding Notes for more detail of the overall process and options available.

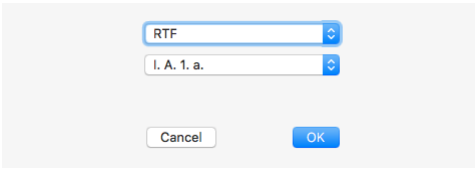
The choices for splitting content are:



- **Break at paragraphs**. Breaks each paragraph into a new note (default option).
- **Delimiter**. Break on a delimiter. A box for entering the delimiter and the delete delimiter tick-box are hidden when this option is not selected.
  - Custom delimiter string: see below.
  - Delete delimiter. Only available if the 'Delimiter' option is set (default = not ticked). Tick this to remove the specified delimiter from the new exploded notes. When the delimiter is a complete line/paragraph, ensure the string used includes the line return character at the end or some output notes will have text starting with a blank line. Tinderbox remembers the last-used custom delimiter until the end of the current session
- **Title**. This controls what text is placed into the title of each newly split note. Text is still truncated with an ellipsis if greater than c.512 characters (in older version pre-v5 it was c.64 characters). Within this overall limit, the title can based on sentences or paragraphs in the exploded note's $Text. A 'sentence' is delimited by the occurrence of a terminating period, exclamation mark or question mark, a paragraph is delimited by a line return (line break). Sentence/paragraph detection is discussed in more detail under the main article on Exploding Notes.
  - **first sentence**. Only the first sentence forms the new title.
  - **first two sentences**. Only the two sentences form the new title.
  - **first paragraph**. Only the first paragraph is used for the new title.
- [Text options]:
  - **Remove title from text**. If selected the text used for the above choice is deleted from the new note's body text.
  - **Omit text**. This results in only a $Name being set for new notes and no $Text.
- **Action**. Allows a simple action to be inserted that is applied to each newly exploded note by setting an $OnAdd for the 'exploded notes' container created by the process. It is useful for tasks like applying a prototype to all new exploded notes. See the main Exploding Notes article for use of the 'Exploded Notes' built-in prototype if using large amounts of code here.
- **Example**. Gives a preview of the $Text of the first exploded note. A control below the preview allows cycling through a preview of successive exploded notes.
- **Explode** button. Click to start the Explode process using the choices set above. The Return key (↩) will also start the Explode process.

Export as Outline panel

On selecting Outline as the export type in the Export sub-menu, a configuration panel is shown. The format choices available are:



- **RTF**
- **text**
- **doc**

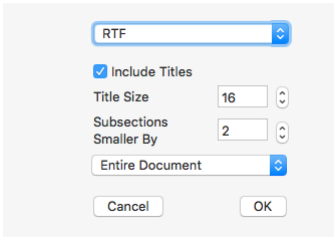
When exporting to RTF or doc formats, list styles are used. A pop-up offers a choice of list styles:

- **I. A. 1.** (Harvard).
- • (bullet)
- **1.** (numbered)

After selection, click **OK**, and the Export folder dialog is shown.

Export as Text panel

On selecting Text as the export type in the Export sub-menu, a configuration panel is shown. The default format for export is RTF, but other formats are available:

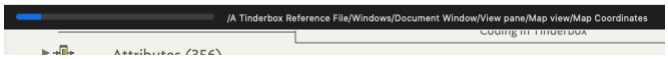


- **RTF**.
- **Word®** (pre v9.5.0 it was **doc**).
- **text**.
- **OPML**. Exports the selected scope in OPML format. If necessary, the relevant built-in templates and prototype are added to the document to facilitate the export before it occurs.
- **Scrivener**. Exports the selected scope in Scrivener (OPML) format. If necessary, the relevant built-in templates and prototype are added to the document to facilitate the export before it occurs.

The controls are:

- **list of formats**. Choices as above.
- **include titles** (RTF, doc and text formats only). This includes the note titles ($DisplayName) as well as their $Text. If this is not ticked, the following title-related controls are hidden:
  - **Title Size** (RTF and doc formats only). The point size used for titles in the exported text.
  - **Subsections Smaller By**. Subsection heads to be reduced in size for each level of indenting, providing finer control over export styling.
- Scope list. A choice of:
  - **Entire Document**.
  - **Selected Notes**.

Export HTML progress bar

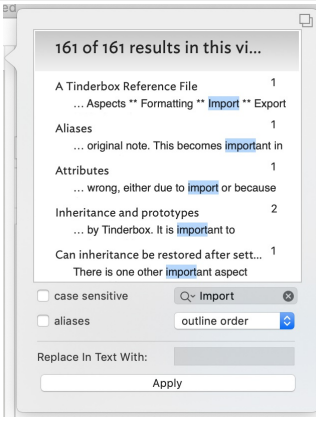


This shows the current progress of the HTML export process. Note that pressing Escape can be used to cancel the export.

The progress bar label shows the full path of the note being exported.

Find results pop-over

The Find results for main view searches are shown in a pop-over. The pop-over can be torn off as a stand-alone dialog to give a more persistent record of the search results. The torn-off Find window retains its selection when clicking on a note. Its title reflects the number of notes found in the document, and the number in the scope of the current view. Selecting a note in the torn-off find window will select that notes in the view, if the note is available.



[list of matching notes]. The list are shows notes matching the Find criteria. Results are listed by note title ($Name) sorted in the default, lexical, sort order. A lexical sort is used means that instances of the same word in different letter case do not sort together: `Ant, Bee, ant` and *not* `Ant, ant, Bee` as a non-coder might assume. To have greater control over the sorting of results or to get a count of matches, use an agent instead of Find.

For notes where the match is in $Text, the (first) match context is sown in a second line of listing with the match highlighted. Matches for note title or user attribute value use a single line listing.

Clicking on a list item makes that note the context of the front window's text pane. Listed notes are rendered in using $Color. Text for notes not in the current view scope are shown as a preview - see 'Preview windows' below.

The pop-over has a series of controls:

- search input box (from which a new search can be run). The box's pop-up menu offers an option for case-sensitive search, and an option to turn off regular expression search.
- a tick-box to alter the case-sensitivity of searches.
- a tick-box to allow inclusion of aliases (by default they are filtered out).
- sort order pop-up with the following choices:
  - outline order (default)
  - Creation date
  - last modified
  - name

The controls work live on the current search results (both in pop-over and tear-off form) without the need to re-run the Find.

Found items are grouped into two groups. The first grouping shows items within context of the current tab's view. Such items might not necessarily immediately be seen within the current view area, e.g. because they are outside the display part of a current Map or inside a collapsed container within a current Outline, etc. The second group lists all other matches that cannot be seen in the view pane without changing the context of the current view, i.e. setting its scope differently from the current settings. Clicking these out of scope list items invokes a preview window (see below).

The Find window can match outline separators and map adornments.

A contextual menu lets the right-clicked-on list item be opened in a new tear-off text window or in a new tab. Shift-double-click on a listed item will also open the result in a new tab.

Find offers up to seven suggested related words as possible autocompletions in the Find bar and in torn-off Find windows. Note: the related text feature only works on macOS 10.14 or later.

The Find results pop-over contains controls to replace all occurrences of a regular expression with a designated string. Note: *this is not undoable*. These replacements apply to $Text only; $Name or the (optionally) selected user attribute are not checked/altered. This features in not included in the torn-off version of the Find results.

Preview windows

After selecting a note in the Find window, if the note cannot be selected and previewed in the current view, Tinderbox displays a preview of the note's text.

Once displayed, either click the preview to close it, or drag it to tear it off as a stand-alone text preview window. Preview closing instructions are shown briefly, at the bottom of the preview pane, on first opening.

Closing the pop-up

Use Escape (⎋) or click outside the pop-up. However, if a preview window is also open, the preview must be closed *before* the Find Results pop-up can be closed. Either tear off the preview to a stand-alone note window, which auto-closes the results pop-up, or click the preview to close it, then use Escape or a click outside the pop-up to close.

Dragging aliases

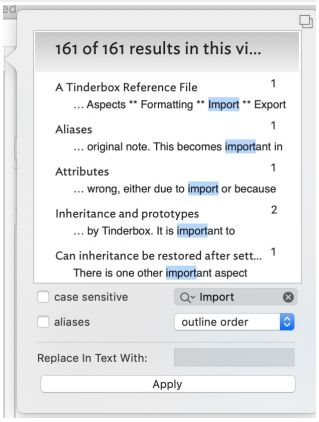
Results dragged into the view pane generate an alias of the dragged result's source note.

Dragging text links

Results from the Find results pop-over can be dragged into the text pane. The display name of the dragged item will be inserted into the text and used as the anchor of a text link to the corresponding note.

Find results stand-alone dialog

The Find results for main view searches are shown in a pop-over. The pop-over can be torn off to give a more persistent record of the search results. The torn-off Find window retains its selection when clicking on a note. Its title reflects the number of notes found in the document, and the number in the scope of the current view. Selecting a note in the torn-off find window will select that notes in the view, if the note is available.



[list of matching notes]. The list are shows notes matching the Find criteria. Results are listed by note title ($Name) sorted in the default, lexical, sort order. A lexical sort is used means that instances of the same word in different letter case do not sort together: `Ant, Bee, ant` and *not* `Ant, ant, Bee` as a non-coder might assume. To have greater control over the sorting of results or to get a count of matches, use an agent instead of Find.

For notes where the match is in $Text, the (first) match context is sown in a second line of listing with the match highlighted. Matches for note title or user attribute value use a single line listing.

Clicking on a list item makes that note the context of the front window's text pane. Listed notes are rendered in using $Color. Text for notes not in the current view scope are shown as a preview - see 'Preview windows' below.

The pop-over has a series of controls:

- search input box (from which a new search can be run). The box's pop-up menu offers an option for case-sensitive search, and an option to turn off regular expression search.
- a tick-box to alter the case-sensitivity of searches.
- a tick-box to allow inclusion of aliases (by default they are filtered out).
- sort order pop-up with the following choices:
  - outline order (default)
  - Creation date
  - last modified
  - name

The controls work live on the current search results (both in pop-over and tear-off form) without the need to re-run the Find.

Found items are grouped into two groups. The first grouping shows items within context of the current tab's view. Such items might not necessarily immediately be seen within the current view area, e.g. because they are outside the display part of a current Map or inside a collapsed container within a current Outline, etc. The second group lists all other matches that cannot be seen in the view pane without changing the context of the current view, i.e. setting its scope differently from the current settings. Clicking these out of scope list items invokes a preview window (see below).

The Find window can match outline separators and map adornments.

A contextual menu lets the right-clicked-on list item be opened in a new tear-off text window or in a new tab. Shift-double-click on a listed item will also open the result in a new tab.

Results from the Find results pop-over can be dragged into a text pane. The display name of the dragged item will be inserted into the text and linked to the corresponding note.

Results dragged into the view pane generate an alias of the dragged result's source note.

Find offers up to 7 suggested related words as possible autocompletions in the Find bar and in torn-off Find windows. Note: the related text feature only works on macOS 10.14 or later.

$Text replacement: this feature is only available in the pop-over mode of Find results.

Preview windows

After selecting a note in the Find window, if the note cannot be selected and previewed in the current view, Tinderbox displays a preview of the note's text.

Once displayed, either click the preview to close it, or drag it to tear it off as a stand-alone text preview window. Preview closing instructions are shown briefly, at the bottom of the preview pane, on first opening.

Closing the dialog

Click the dialog window's 'close' (red) button. This will close the dialog and any preview windows open.

Dragging aliases

Results dragged into the view pane generate an alias of the dragged result's source note.

Dragging text links

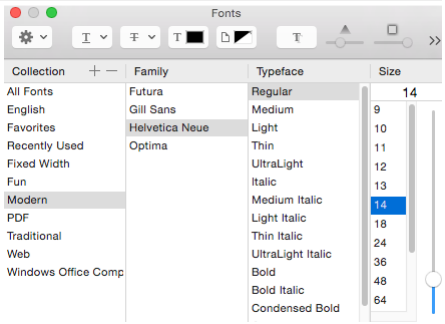
Results from the Find results pop-over can be dragged into the text pane. The display name of the dragged item will be inserted into the text and used as the anchor of a text link to the corresponding note.

Fonts dialog

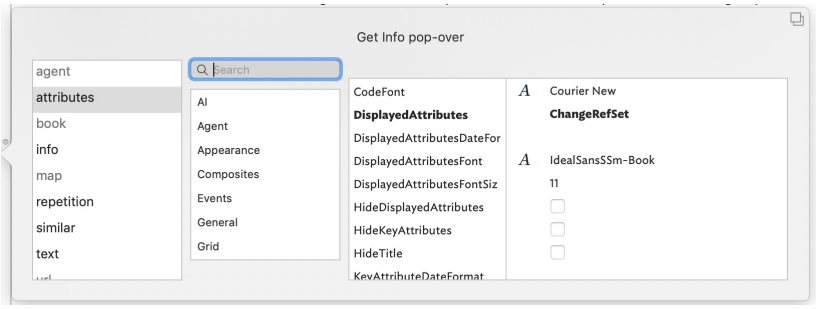
The macOS font palette can be shown either via the Note menu or a shortcut (Cmd+T).

This palette can be used to set only the typeface and size. Other controls such a text colour have no result on text within Tinderbox notes. Bold and italic text are created by using the bold and italic font variants of the regular font in use.

The palette can also be used to set Outline and Map view title font typeface and size, etc.



Get Info pop-over



The Get Info pop-over provides a series of tabs showing information about the currently selected note. Dismiss the pop-over by clicking anywhere outside it. The pop-over will position over the non-active pane of the main window, i.e. the main view or text pane.

Dragging the pop-over will result in a stand-along dialog. This allows the Get Info information to be viewed and used when the source note is no longer selected. It is possible to have multiple Get Info dialogs open at the same time.

The sub-tab selected on first opening will generally default to the last one used in the current session.

The pop-over has the following tabs:

- agent tab
- attributes tab
- book tab
- info tab
- map tab
- paths tab
- repetition tab
- similar tab
- text tab
- url tab
- words tab

agent tab

This tab controls basic agent functions. This replicates information found on the sub-subs of the Action Inspector. Edits made on the this dialog are live as soon as the Return key (↩) is pressed or focus shifts from the current input box on the pop-over.

This popover is shown by default as soon as a new agent is defined.

Agent title (only shown if the current object is an agent). The $DisplayName for the agent is displayed above the input boxes.

Query. The agent's query (stored in $AgentQuery). Auto-complete is offered for action code terms and system/user defined attributes (if starting with a '$' prefix, as is best practice). Editing this code is the same as if editing the Action Inspector's Query tab.

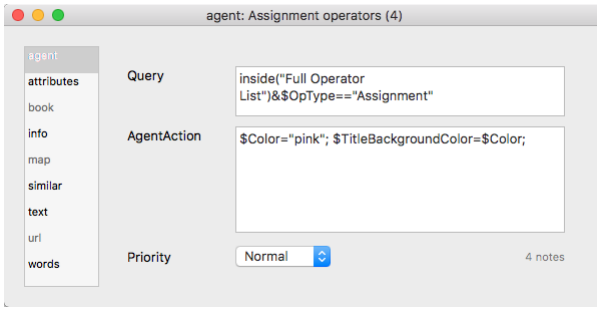
Action. The action to be applied to each child alias (stored in $AgentAction). Auto-complete is offered for some input, as described above. Editing this code is the same as if editing the Action Inspector's Action tab.

Priority. This opens the Agent Priority pop-up list, allowing the agent to be turn on, off or to a non-default working state (e.g. on, but at lower priority). Setting is stored in $AgentPriority. Using this is the same as using the Action Inspector's Query tab's **Priority** pop-up.

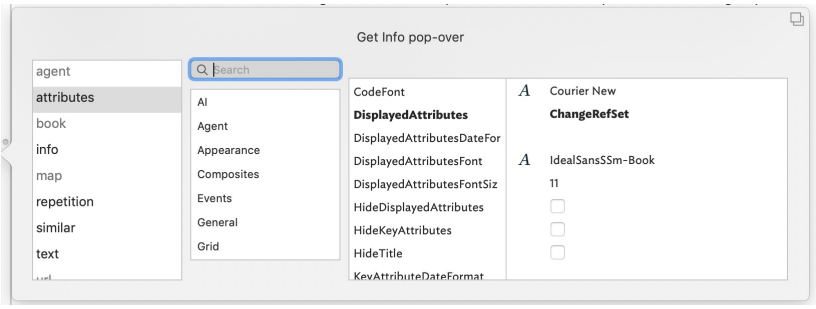
Result count (only shown if the current object is an agent). The count of matches to the current query is shown in grey text bottom right of the pop-over.

Dragging the pop-over will result in a tear-off window that will persist until the end of the current session (i.e. the document and/or app are closed). A torn-off window may be closed during the current session if no longer needed.

From v9.5.0, the agents query and action code boxes receive action code syntax colouring.



attributes tab



This tab shows attribute data and is the most direct successor to the Get Info dialog in previous versions of Tinderbox. The tab has 3 sections.

Search. This works as per the Document Inspector, System tab, in seeking to auto-match any input to a currently defined attribute (here for system or user). If multiple matches are found, these are displayed in a pop-up listing allowing the user to select the preferred match. If a match is found, the correct attribute group is selected in the listing and its contents loaded. From this the matched attribute is also selected. Here, $BorderColor has been searched for and matched.

Attribute Group listing. A list of all the defined groups of System attributes and any defined User attributes.

Attribute data table. A list of attributes within the currently selected group. The left column shows the attribute name. The right column shows the current local or inherited value. Display and editing of attribute values is as described for the Displayed Attributes table. String, Number, Set and List type attribute will display a pop-up menu showing values currently applied to that attribute elsewhere in the document.

Row height in the table. From v9.5.2, String-type (only) attributes can have a bigger line height. The default row hight for String-type attributes remains one line, but this can be extended to a maximum of seven lines. Within that space line breaks in the String's value are honoured but any overflow is clipped (i.e. additional content cannot be scrolled). Line height is set per-attribute using The Document Inspector's System and User tabs. From v9.6.0, multi-line attributes are also permitted for Sets and Lists as well as String type attributes.

Some attribute data types show an icon between the attribute name and value cells. The function of these is described under the notes on the Text pane's Displayed Attributes table.

The meaning of the styling of different rows (bold, strikethrough, etc.) is explained here.

Dragging the pop-over will result in a tear-off window that will persist until the end of the current session (i.e. the document and/or app are closed). A torn-off window may be closed during the current session if no longer needed.

Editing the $Prototype value in this dialog is just one way of setting a note's prototype.

The attribute $ReadOnly is exempted from being read-only, even if set to `true`. This allows you to turn off ReadOnly from the Displayed Attributes table or from Get Info ▸ Attributes.

Changes to Displayed Attributes affect $Modified

Changing an attribute value in the displayed attributes table or in Get Info's attributes tab updates $Modified. Changing an attribute value in a stamp (including Quickstamp) or an action does not update $Modified.

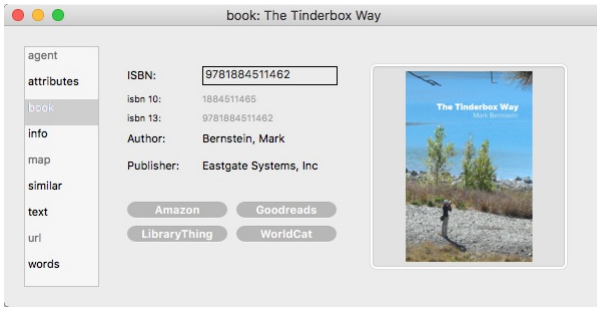
book tab

The book tab shows book-related data. For notes with an $ISBN value, Tinderbox will assume the note is about some from of book and the book tab will:

- look up author and title, if not already known, from ISBNDB.com
- look up the cover image from amazon.com
- provide useful links to Amazon, Goodreads, LibraryThing, and WorldCat for this book.

The **ISBN** box shows $ISBN data or allows it to be set directly by editing in the box.

Dragging the pop-over will result in a tear-off window that will persist until the end of the current session (i.e. the document and/or app are closed). A torn-off window may be closed during the current session if no longer needed.

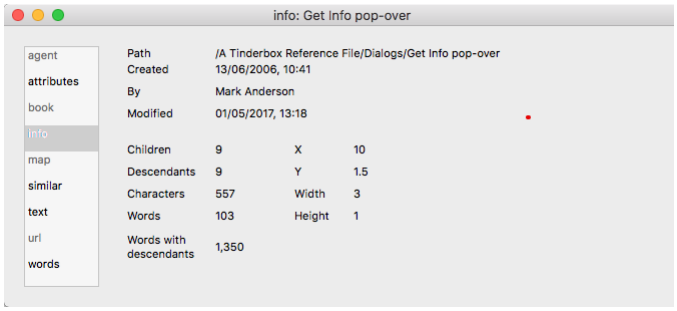


info tab

The info tab holds basic metrics about the current note, mostly relating to attributes:

- **Path**. $Path.
- **Created**. $Created.
- **By**. $Creator.
- **Children**. $ChildCount.
- **Descendants**. $DescendantCount.
- **Characters**. $TextLength.
- **Words**. $WordCount.
- **Words with descendants**. $WordCount plus the sum of descendants' $WordCount.

Dragging the pop-over will result in a tear-off window that will persist until the end of the current session (i.e. the document and/or app are closed). A torn-off window may be closed during the current session if no longer needed.



map tab

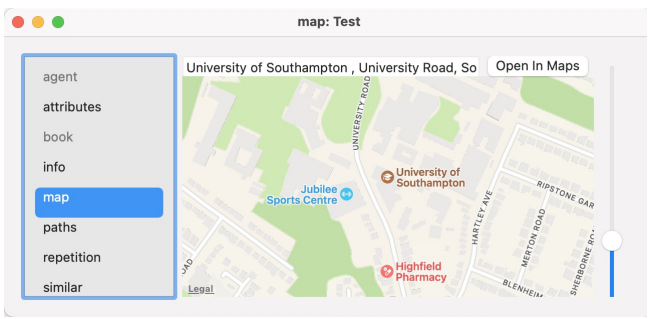
The map tab checks to see if $Address is populated and if it is will attempt to resolve the address into Google Maps, populating $Latitude, $Longitude and $GeocodedAddress.

The top input box shows data in $Address, or adding data to the box will set there attribute.

If a map location is found, the map can be zoomed via the slider control to the right of the map.

Dragging the pop-over will result in a tear-off window that will persist until the end of the current session (i.e. the document and/or app are closed). A torn-off window may be closed during the current session if no longer needed.

From v9.6.0, the map tab has an **Open In Maps** button that opens the (macOS Apple) Maps application.



paths tab

The paths panel displays the number of distinct link types in use within the document. These are listed in a pop-up control.

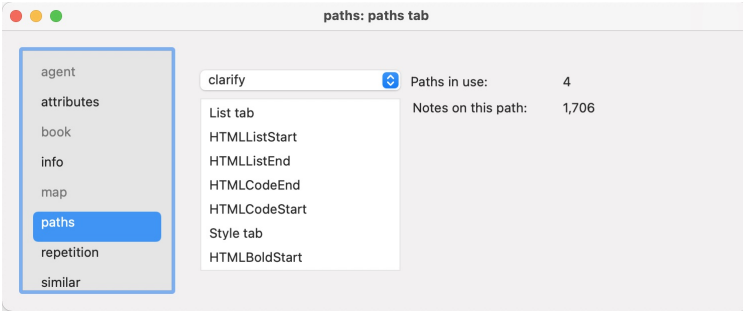
If you select a link type from the pop-up, the list panel will tell you which notes use that link type, and how many notes are on this path. With a selected path (link) type, the list box below it shows the titles ($Name) of all notes with inbound or outbound link(s) of that type.

As that list includes both the source and destination of such links, the list will thus show more notes than the discrete number of links.

If using information from this dialog to triage link type allocation, remember that link configuration (e.g. link type) can only be done in the context of the link's *source* note.

There are two totals:

- **Paths**. The number of discrete, currently defined, links that are actually used. Often there may be link types defined in the document that are not actually used, i.e. larger than the figure shown here.
- **Notes on path**. The number of unique notes that are either the source or target of a link of the selected type.

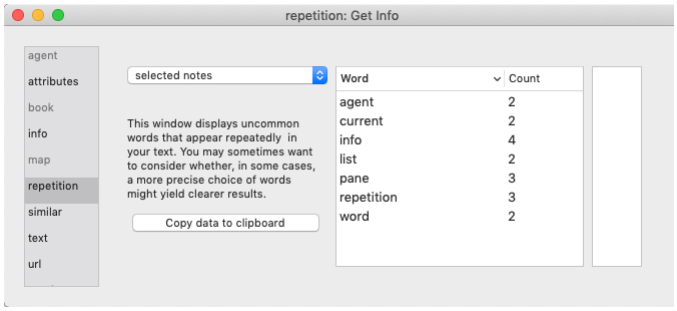


repetition tab

The left list-box shows the repetition count of each repeated word. The right listbox shows the context of occurrence of the currently selected repetition (i.e. left list-box selection) with the text excerpt highlighting the repeated word. Clicking on an item in the right list-box opens that note in the text pane of the front document window. The data in the list-boxes can not be copied.

The **Scope** of inspection is variable, selected by a pop-up menu with the following choices (the last option is the default):

- **document** (whole current/front document). Use this option with caution in a very large document, it may take a little while to complete scanning the existing text.
- **parent & descendants**
- **note & descendants**
- **selected notes** (default)



The **Copy data as clipboard** button copies the word/count list data to the clipboard as a tab-delim list.

A table of results shows the discrete words and the count of said words. The occurrence list can be sorted on any of the columns (default is 'word' in alphabetical order).

The repetition tab thus offers insight into words that are used repeatedly in the selected notes, sections, or in the entire document. Consistent usage may be needful or desirable, of course, but noting repetition can call attention to opportunities to adopt more precise language.

The pane lists words that occur two or more times except:

- words with fewer than four characters
- words that appear in the built-in 'stoplist' of 100 common English words
- words that appear in an option customised stoplist.txt in Tinderbox's Application Settings.
- words that appear in a note in the current document named 'stoplist', if one exists. The note title is case sensitive, but the note can be placed anywhere in the document (most likely away from the primary content).

The repetition indexing process tries to treat words derived from a common stem as repetitions, so plurals and verb conjugations are often handled intelligently.

similar tab

The similar tab uses Tinderbox's 'similar to' function to list (up to) 10 items it adjudges most similar to the current note. It replaces the stand-alone minor view in older Tinderbox versions. Similarity is based on several factors, including:

- the text ($Text) of the note
- the note title ($Name)
- any text contained in user attributes (i.e. attributes of String type or which are string-based)

In addition, weighting is applied for:

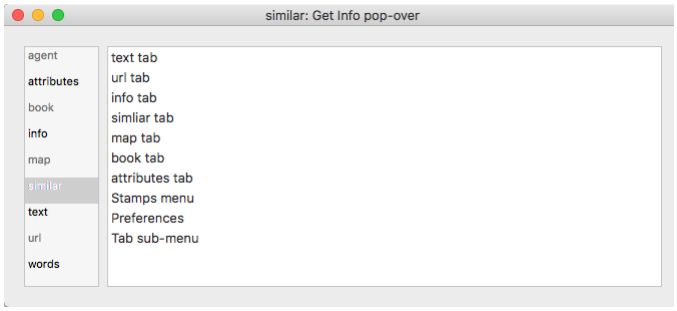
- notes having the same prototype
- notes having roughly similar amounts of text

Similar notes are listed in order from top to bottom, starting with the most similar note at the top.

Double-clicking any list item sets the clicked note as the focus of the document window's text pane and dismisses the pop-over.

Dragging the pop-over will result in a tear-off window that will persist until the end of the current session (i.e. the document and/or app are closed). A torn-off window may be closed during the current session if no longer needed.

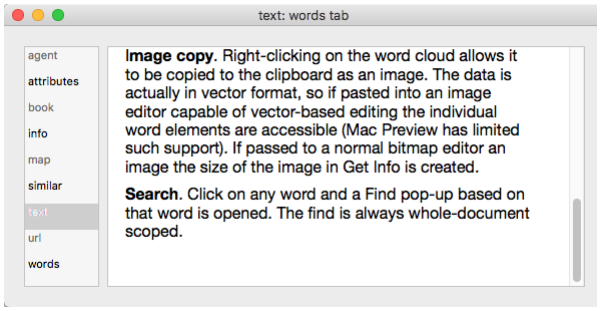
This information can also be accessed as an action code similarTo() and an export code ^similarTo()^.



text tab

The text tab shows the $Text space of the current note. Dragging the pop-over will result in a tear-off window that will persist until the end of the current session (i.e. the document and/or app are closed). A torn-off window may be closed during the current session if no longer needed.

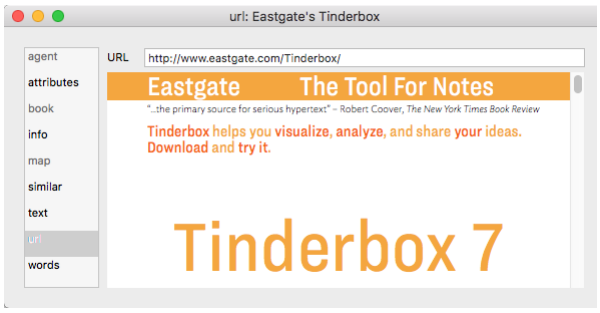
This feature is offered in a richer form by opening a stand-alone text window for a note. Links in $Text are non-functional in this display.



url tab

If the note's $URL is populated, the url tab will preview the linked webpage. The web content in the preview is live and links can be followed (but it is not intended for general web browsing).

Dragging the pop-over will result in a tear-off window that will persist until the end of the current session (i.e. the document and/or app are closed). A torn-off window may be closed during the current session if no longer needed.



words tab

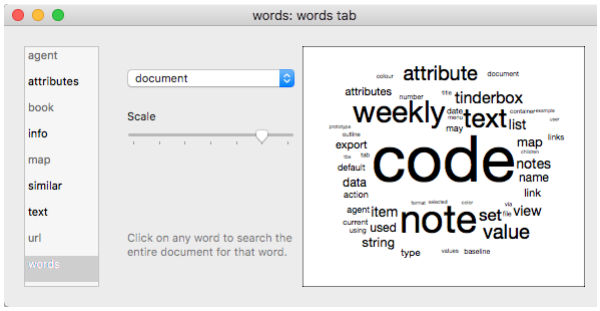
This tab visualises the most common words in the $Text and $Name of in-scope notes. The word cloud acknowledges both the built-in stoplist and document's own stoplist.txt list, if present.

Word clouds and aliases: If viewing the word cloud of a note and its descendants, and if the selected note is an *agent*, the word cloud indexes each alias found by the agent.

The tabs controls are:

'scope' pop-up. The scope of examination within the (current) document can be set to one of:

- **document**: whole document (default)
- **parent & descendants**: parent note of the current note and all the parent's descendants
- **note & descendants**: current note and all its descendants
- **selected note**: currently selected note



The scope shown in the pane is equivalent to the export output of ^documentCloud^.

'Scale' slider. The view can be magnified or reduced by dragging the dragging the slider.

Image. A word cloud of the most common words found in the current scope. The words are drawing in $NameFont. Re-sizing the pane causes the layout of the contents to be re-arranged.

Image copy. Right-clicking on the word cloud allows it to be copied to the clipboard as an image. The data is actually in vector format, so if pasted into an image editor capable of vector-based editing the individual word elements are accessible (Mac Preview has limited such support). If passed to a normal bitmap editor an image the size of the image in Get Info is created.

Search. Click on any word and a Find pop-up based on that word is opened. The find is always whole-document scoped.

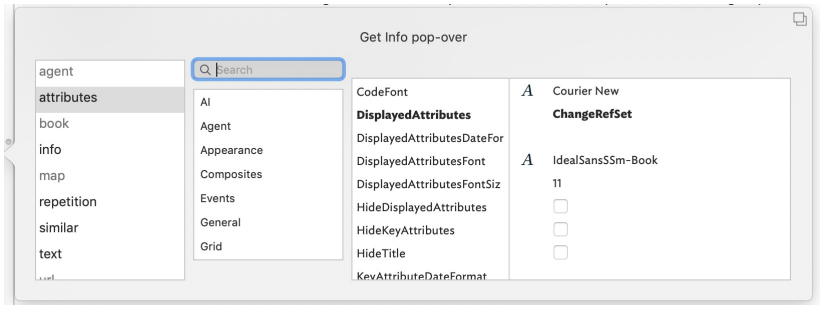
For note and section, each distinct occurrence of a word is counted. For the document level view, Tinderbox counts the number of notes that contain a word 1 or more times. The display contains up to 100 words - less if there are fewer valid words in scope.

Unlike the 'similar' notes tab, only $Text and $Name are interrogated. Other sources of words such as textual user attributes are thus ignored.

If the dialog is torn-off as a discrete window, subsequent changes to in-scope $Text are not reflected; re-select the window sub-tab to refresh the view. Similarly, the 'selected only' scope only maps the note selected at the time the view was opened, and does not track changes in selection while the tab is active.

This tab replaces the 'Common Words' view in very early versions of Tinderbox.

Get Info stand-alone dialog



The Get Info pop-over provides a series of tabs showing information about the currently selected note. Dismiss the pop-over by clicking anywhere outside it. The pop-over will position over the non-active pane of the main window, i.e. the main view or text pane.

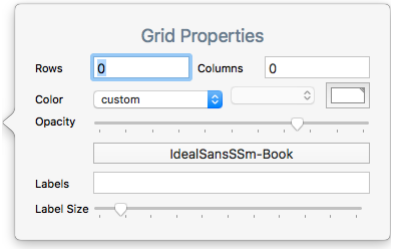
Dragging the pop-over will result in a stand-along dialog. This allows the Get Info information to be viewed and used when the source note is no longer selected. It is possible to have multiple Get Info dialogs open at the same time.

The sub-tab selected on first opening will generally default to the last one used in the current session.

For detail about the Get Info sub-tabs, see here.

Grid Properties pop-over

This pop-over is used to set up an optional grid layout on map adornments. It is opened by clicking the 2x3 grid icon on a selected adornment.



- Rows. The desired number of grid rows. Stored in $GridRows.
- Columns. The desired number of grid columns. Stored in $GridColumns.
- colour controls. Sets the colour used to draw the grid and its optional labels. Stored in $GridColor.
- Opacity. Controls the opacity used to drawn the grid. Stored in $GridOpacity.
- font control. Sets the font used to draw grid labels. Stored in $GridLabelFont.
- Labels. A list-type list of row and column labels. Stored in $GridLabels.
- Label size. Sets the (point) size at which labels are drawn. Stored in $GridLabelSize.

Labels are drawn, centred, at the bottom of each grid cell. Labels are read by row, then column. Thus in a 2-row 3-column grid, the first 3 labels in the list are the labels for all 3 columns of row #1, and so on. By entering an empty list item using 2 consecutive semi-colons in the list, i.e. ';;', specific cells may be left un-labelled.

If only rows or columns are set and the other value is left at zero a single row or column is drawn, i.e. a value of 1 is assumed and drawn though not set in the relevant attribute (which remains at '0').

HTML Export folder dialog

On calling an HTML Export or Outline Export or Export of Selected Note (via the File menu) a OS-style folder selection dialog is shown. Select an existing folder or create a new one. The selected folder is the location to which export will occu
This location is remembered in the TBX so it should not normally be required to reset it after first use unless deliberately choosing a new location.

Depending on the size and complexity of your TBX, there may appear to be some delay before the dialog opens. This is because before doing a full HTML export, all agents are updated to ensure output is correct, especially where automatic updating of agents is turned off.

Once HTML export starts a progress bar appears above the main view showing the progress of the export process.

Outline export. Outline export results in a single plain text file with the title of each note in the document indented one tab stop per outline depth level. Thus root level notes are not indented, their children are indented one tab, etc. See more on Outline Export.

Selected Note(s). The selected items are exported to the same folder, regardless of hierarchy location. Do not use this option for re-exporting a single note to its location within a whole document export context.

Link parking space (empty) click pop-over

If the tab bar's link parking space is clicked when empty, a pop-over is shown but cannot be used as there is no stored link available to complete. This error is further signposted by the link parking space briefly showing a red fill when the mouse click occurs.

Link parking space click pop-over

This pop-over is shown when the link parking space is clicked. It allows rapid completion of a link, especially to a destination not currently visible in the view pane.

To set the destination note, type in the input box. As the user types Tinderbox offers autocomplete solutions to speed the process. Once the destination is entered, click the 'Link' button to create the link. The Create Link dialog is then shown to configure the link. If choice is wrong, click the Escape key (⎋) to cancel the whole process.

Take care if linking deliberately to aliases or notes whose title is a duplicate of other note(s) in the document. In such cases it is probably better to use the drag/drop method of link creation to ensure the correct destination item is selected.

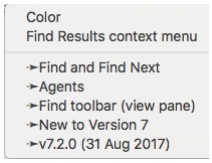
The parking space popover accepts either the name of a destination node or the destination URL of a web link.

The quick link destination popup understands ziplinks backlink mark-ups, so '[[

Link widget context pop-up

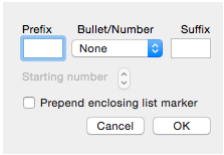
Right-clicking the link widget of a selected note in Outline, Map, Chart and Timeline views shows a pop-up summary of links.

The pop-up lists all the ingoing and outgoing internal links of the selected note; web links are not included. Incoming links are listed at the top, separated by a ruler from outgoing links. Outgoing links are further marked by being preceded with a right-pointing arrow. No differentiation is made between basic links and text links.



List panel

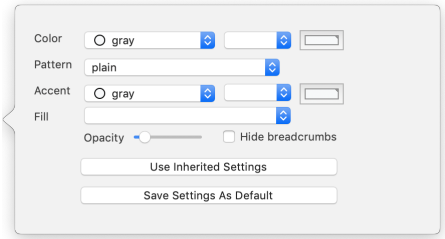
This panel uses standard List configuration controls to inset a list into $Text.



Map Settings pop-over

This is the configuration pop-up for the current Map view:

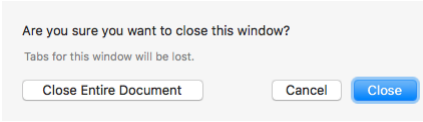
- **Color**. Colour controls to set the parent object's $MapBackgroundColor.
- **Pattern**. Uses Pattern pop-up to set the parent object's $MapBackgroundPattern.
- **Accent**. Colour controls to set the parent object's $MapBackgroundAccentColor.
- **Fill**. Uses Fills pop-up menu to set the parent object's $MapBackgroundFill.
- **Opacity**. Slider control to set the parent object's $MapBackgroundFillOpacity.
- **Hide Breadcrumbs**. Toggles view's breadcrumb bar (default: ticked).
- **Use Inherited Settings**. A button to reset all the above attributes to current document's default values.
- **Save Settings As Default**. A button to set all the above attributes as the current document's default values.



Multi-window close confirmation pane

A sheet is displayed when closing one window in a document with multiple main windows open, asking for confirmation of intent to close. This warning is shown because closing a window (other than the last one open) causes the current tab choices to be lost. The options are:

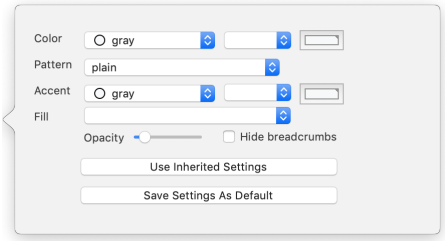
- **Close Entire Document**. This closes and retains all (persitable) windows.
- **Cancel**. Cancels the close action and closes the dialog.
- **Close**. (default selection) Closes only the selected window. This windows tab arrangement is not saved so cannot be re-opened.



Outline Settings pop-over

This is the configuration pop-up for the current Outline view:

- **Color**. The parent object's $MapBackgroundColor value.
- **Pattern**.The parent object's $MapBackgroundPattern value.
- **Color 2**. The parent object's $MapBackgroundAccentColor value.
- **Fill**. The parent object's $MapBackgroundFill value.
- **Opacity**. The parent object's $MapBackgroundFillOpacity value.
- **Hide Breadcrumbs**. Toggles views breadcrumb bar (default: ticked).
- **Use Inherited Settings** button: click to reset the document defaults for Outlines.
- **Save Settings As Default** button: click to apply the current view's settings as the document default.



Roadmap pop-over

The Roadmap pop-over is called from the View menu or from the note's main view icon's pop-up menu. The Roadmap lets you see all of the internal links leading to or from a selected space—the local area of a complex hypertext (web links out of the document are not included. Aliases have their own Roadmap, not that of their original. The Roadmap not only gives and overview of inbound and outbound links but also allows exploration of the local hypertext (i.e. the TBX's content). Links out of the current document, e.g. web links, are not visualised in the Roadmap. RTF-based 'Smart' links are also omitted from the listing.

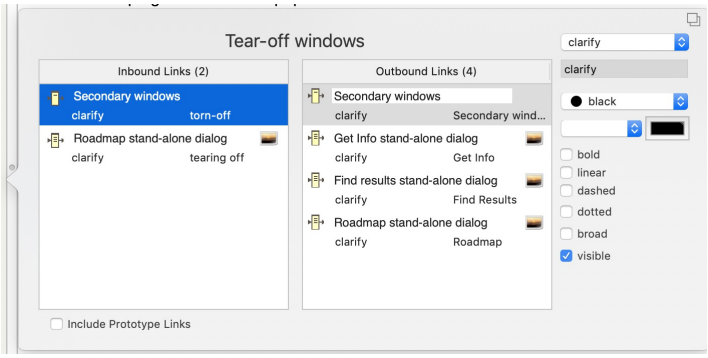
The left column lists all notes that link into the current note note; the column header lists a count of the inbound links. The right column lists all the note's outbound links, again with a count in the column header. To change the focus of a Roadmap, double-click any item in either column. The Roadmap will change to focus on the selected note, with links leading in and out of it in the two columns. The current window's main view will update to the selected note, *if it is in scope*; note that if the pop-up is torn off the text pane stops following focus of the view.

Roadmap opens with the initial focus on the outbound links list if there are any outbound links. Otherwise, the inbound links list gains the initial focus.

Roadmap allows editing of the properties of the selected link. At the top of the Roadmap, Tinderbox additionally shows the *display name* of the note for which inbound and outbound links are currently being shown. The tooltip of this label is the full path of that note (which will include the *name* of the note.

Pressing the **Delete** key (⌫) will delete the selected link.

Tab (↹) cycles from first listed inbound link → first listed outbound link → link type → first listed inbound link, etc. Use **spacebar** to follow a link (or **Return** (↩) from v9.1.0) and set window focus on the currently selected list item. Use *blind type* to select a source or destination.



Note: if the main view is a map, selection/focus can move to other items on the current map but not to notes on other maps. In the latter case, only the Roadmap updates.

The Roadmap title shows the $DisplayName of the notes whose data is being displayed. Each listed link item shows:

- Line one. The $DisplayName of the linked note (in $Color) and at right its $Badge (if one is set).
- Line two. The link type of the link and, for text links only, that link's anchor text.
- Roll-over tooltip for either line: the full path of the linked note.

If the Outline Document Settings for 'Darker colors' is ticked, the item is rendered in a darker tint of their $Color.

Include Prototype Links. A tick-box (ticked by default) at bottom left allows prototype-type links to be filtered out. Prototype assignments are stored, in the TBX, as links but are not normally used in maps and excluded from link count attributes. However, leaving such links in the last can help tracing prototype-based relationships in a document.

Button **focus of [title of calling note] pop-over**. Located at middle-bottom of the pop-over, this refocuses the pop-over on the note from which the pop-over was opened.

If double-clicking to follow a link and if the destination is not visible in the current view, Tinderbox will refocus the view so as to make it visible. This makes it easier to follow link-based trails around a document.

The Roadmap pop-up can be torn off into a stand-alone window. Such windows last until the end of the current session but cannot be saved to last across sessions. Tear-off Roadmaps can be useful triaging links: when torn-off, the Roadn dialog has a button to allow you to refocus the roadmap on the selected note and the ability to see link comments.

Roadmap stand-alone dialog

The Roadmap dialog is created by ' tearing off' a Roadmap pop-over. The Roadmap lets you see all of the internal links leading to or from a selected space—the local area of a complex hypertext (web links out of the document are not included. Aliases have their own Roadmap, not that of their original. The Roadmap not only gives and overview of inbound and outbound links but also allows exploration of the local hypertext (i.e. the TBX's content). Links out of the current document, e.g. web links, are not visualised in the Roadmap. RTF-based 'Smart' links are also omitted from the listing.

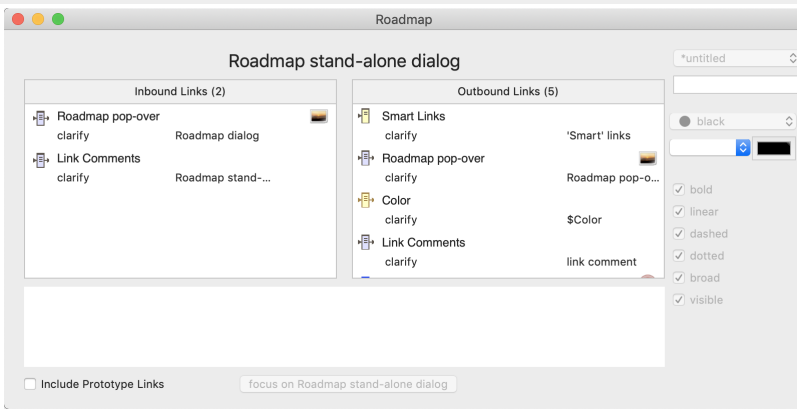
The left column lists all notes that link into the current note note; the column header lists a count of the inbound links. The right column lists all the note's outbound links, again with a count in the column header. To change the focus of a Roadmap, double-click any item in either column. The Roadmap will change to focus on the selected note, with links leading in and out of it in the two columns. The current window's main view will update to the selected note, *if it is in scope*; note that if the pop-up is torn off the text pane stops following focus of the view.

From v9.1.0, Roadmap opens with the initial focus on the outbound links list if there are any outbound links. Otherwise, the inbound links list gains the initial focus.

Roadmap allows editing of the properties of the selected link. At the top of the Roadmap, Tinderbox additionally shows the *display name* of the note for which inbound and outbound links are currently being shown. The tooltip of this label is the full path of that note (which will include the *name* of the note.

Pressing the **Delete** key (⌫) will delete the selected link.

Tab (↹) cycles from first listed inbound link → first listed outbound link → link type → first listed inbound link, etc. Use **spacebar** to follow a link (or **Return** (↩) from v9.1.0) and set window focus on the currently selected list item. Use *blind type* to select a source or destination.



Note: if the main view is a map, selection/focus can move to other items on the current map but not to notes on other maps. In the latter case, only the Roadmap updates.

The Roadmap title shows the $DisplayName of the notes whose data is being displayed. Each listed link item shows:

- Line one. The $DisplayName of the linked note (in $Color) and at right its $Badge (if one is set).
- Line two. The link type of the link and, for text links only, that link's anchor text.
- Roll-over tooltip for either line: the full path of the linked note.

If the Outline Document Settings for 'Darker colors' is ticked, the item is rendered in a darker tint of their $Color.

Link Comment box. Below the in-/out-bound columns a box shows the link comment (if any) of the selected link; the comment can be edited. This box is not shown when the dialog is a pop-over,

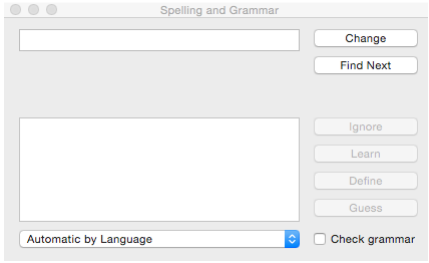
Include Prototype Links. A tick-box (ticked by default) at bottom left allows prototype-type links to be filtered out. Prototype assignments are stored as links, in the TBX's data, but are not normally used in maps and excluded from link cour attributes. However, leaving such links in the last can help tracing prototype-based relationships in a document.

Button **focus on [title of calling note]** . Located at middle-bottom of the dialog, this refocuses the dialog back onto the note from which the dialog was first opened. Or, if the user has shifted the selection of the current tab's view, the dialog updates to show the links for the currently selected note.

As stand-alone dialogs the Roadmap dialogs last only until the end of the current session; they cannot be saved to last across sessions.

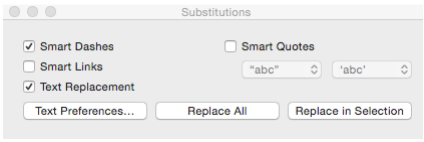
Spelling and Grammar dialog

This is an OS standard dialog allowing review and correction of presumed spelling errors.



Substitutions dialog

This is an OS standard dialog allowing setting of various smart (on-the-fly) substitutions of input text.



Summary Display Properties pop-over

Used only in Map view, this pop-over is opened via icon displayed to the left of container notes (including agents) when selected. This approach makes it easier to correctly set the two attributes controlling container summary display tables.

The properties table has two columns, one for the $TableHeading component and one for $TableExpression. Each *row* in the properties table represents a *column* in the actual table display. Add as many rows as the table needs columns, using the +/- buttons to add/remove rows.

Use Headings. If un-ticked, the Headings column is suppressed and only $TableExpression components are defined.

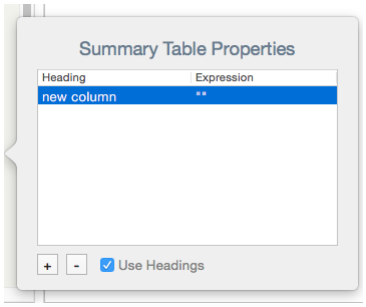
$TableHeading is set to the aggregate of the rows in the first column, and $TableExpression uses those from the second.

To alter the order of display columns, click-hold-drag the relevant row up or down in the Properties table into the correct new location.

Entering edit mode for cells in the table

This can be done via keystroke:

- using the **Return** key or **Tab** key.
  - The **Tab** key moves the active edit one cell right. The Tab loops through all cell in the row, then out of edit mode, then back to edit mode of column one of the selected row.
- Cursor right-click on a cell. Note the click must be on *existing text in the cell*. Clicking elsewhere in the cell has no effect.

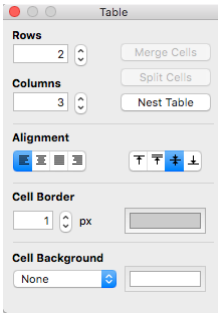


Selecting a row in the table

Either click on a row with the mouse cursor, or use the up and down arrow keys (↑↓). If no row is selected, pressing down-arrow auto-selects the first row and pressing the up-arrow auto-selects the last row.

Table dialog

This dialog contains standard controls for defining tables inserted into $Text. These tables cannot be directly exported to HTML and are intended for RTF use.

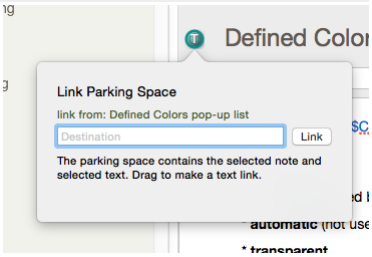


Text pane with $Text selection, Link parking space click pop-over

This pop-over is shown when the link parking space is clicked on a text pane that has some $Text selected. It allows rapid completion of a link, especially to a destination not currently visible in the view pane.

To set the destination note, type in the input box. As the user types Tinderbox offers autocomplete solutions to speed the process. Once the destination is entered, click the 'Link' button to create the link. The Create Link dialog is then shown to configure the link. If choice is wrong, click the Escape key (⎋) to cancel the whole process.

Take care if linking deliberately to aliases or notes whose title is a duplicate of other note(s) in the document. In such cases it is probably better to use the drag/drop method of link creation to ensure the correct destination item is selected.

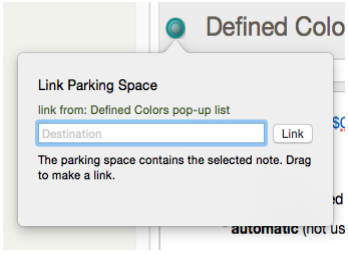


Text pane, Link parking space click pop-over

This pop-over is shown when the text pane's link parking space is clicked. It allows rapid completion of a link, especially to a destination not currently visible in the view pane. Note the 'T' in the link park indicating a Text-type link will be created.

To set the destination note, type in the input box. As the user types Tinderbox offers autocomplete solutions to speed the process. Once the destination is entered, click the 'Link' button to create the link. The Create Link dialog is then shown to configure the link. If choice is wrong, click the Escape key (⎋) to cancel the whole process.

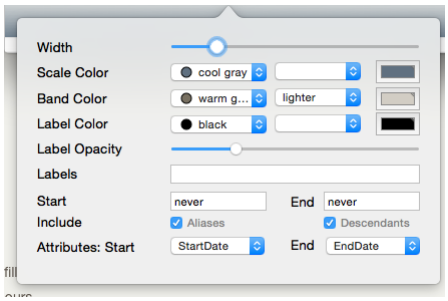
Take care if linking deliberately to aliases or notes whose title is a duplicate of other note(s) in the document. In such cases it is probably better to use the drag/drop method of link creation to ensure the correct destination item is selected.



Timeline Settings pop-over

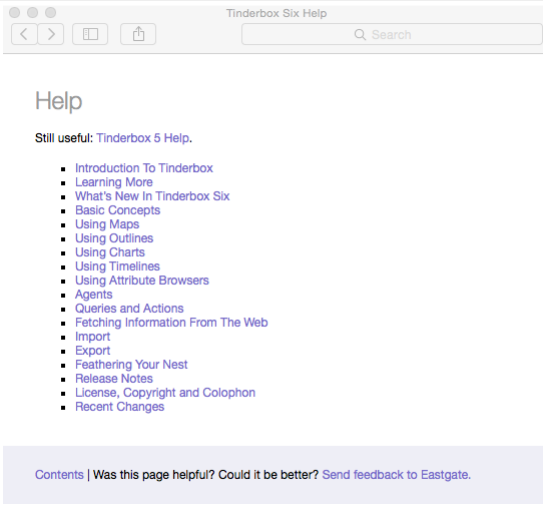
This pop-over gives a visual method for setting a number of timeline-related settings for the current Timeline view:

- **Width**. This adjusts the scale of the bottom scale, i.e. how much horizontal distance there is between days or months, etc.
- **Scale Color**. A trio of standard colour controls set the main colour used in the scale bar at the bottom of the timeline. $TimelineScaleColor.
- **Band Color**. A trio of standard colour controls set the colour a tint of which is used to draw alternate timeline bands ($TimelineColor). The actual colour set is a 'lighter' shade of the colour set in the controls.
- **Label Color**. A trio of standard colour controls set the colour used to draw timeline band labels ($TimelineBandLabelColor).
- **Label Opacity**. Sets the opacity with which timeline band labels are drawn ($TimelineBandLabelOpacity).
- **Labels**. Sets the labels used for timeline bands ($TimelineBandLabels).
- **Start**. Sets $TimelineStart.
- **End**. Sets $TimelineEnd.
- **Include**:
  - **Aliases**. Sets $TimelineAliases.
  - **Descendants**. Sets $TimelineDescendants.
- **Attributes**:
  - **Start**. Choose and alternative Date-type attribute for plotting event start instead of $StartDate.
  - **End**. Choose and alternative Date-type attribute for plotting event end instead of $EndDate.



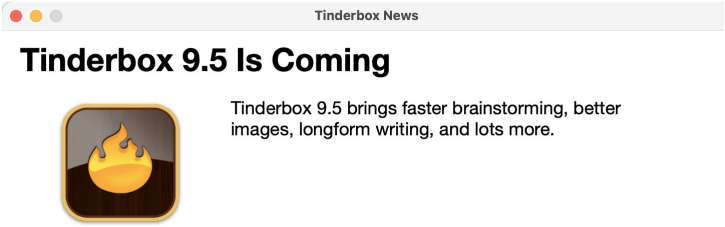
Tinderbox Help dialog

This is a standard Apple Help type of window. This contains the main Tinderbox manual.



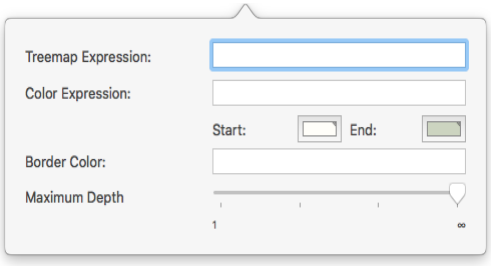
Tinderbox News dialog

A new dialog in v9.5.0. At startup, Tinderbox checks with an Eastgate server to see if there is a news item the user has not yet seen. If so, the news is displayed in a separate window at app start. News will only be auto-shown once, but the most recent news item can always reviewed by choosing Tinderbox News from the Tinderbox menu.



Treemap Settings pop-over

This is the configuration pop-up for the current Treemap view. Note that none of these setting are saved as attributes but the are saved as part of the Tab's data.

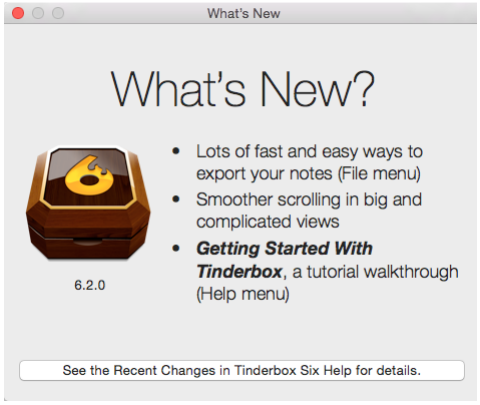


- **Treemap Expression**. Enter an attribute name or action code expression. Used to derive a number used to set each item's area. Items with a value of zero are not drawn at all. Non-Number attributes return a value of 1 if they have a set/inherited value: i.e. string-based are not empty, booleans are `true`. The default is no no expression code.
- **Color Expression**. An action code expression use to colour each note. The colour used is a shade between the start/end colours set below. The minimum value uses the start colour, the maximum the end colour with other values using a shade between the two. The default is no no expression code.
- **Start**. The starting colour used of shading treemap items. The colour chip opens a standard OS colour palette. The default is an off-white #FFFEF9.
- **End**. The ending colour used of shading treemap items. The colour chip opens a standard OS colour palette. The default is a muted light green #CCD4C0.
- **Border Color**. Enter the Tinderbox colour (named or hex value) used to draw the border of each item. Or use an expression resolving to a named or hex colour value. If not specified, items draw with a border of $Color. The default is no value. (i.e. it uses $Color).
- **Maximum Depth**. Allows display of only the **N** upper levels of the treemap. This can sometimes improve clarity of presentation.

What's New dialog

This dialog opens with each new Tinderbox session. It indicates the current version, and a few headline changes for that version.

Clicking the button at the bottom opens the Apple Help for Tinderbox.


What's New
What's New?
6.2.0
Lots of fast and easy ways to export your notes (File menu)
Smoother scrolling in big and complicated views
Getting Started With Tinderbox, a tutorial walkthrough (Help menu)
See the Recent Changes in Tinderbox Six Help for details.

Menus

- Main menu bar
- Menus & sub-menus
- Pop-up menus and lists

Main menu bar

The main menu bar has the following menus:

- Tinderbox 9
- File
- Edit
- Format
- Note
- View
- Stamps
- Window
- Help

Menus & sub-menus

These are the menus available from Tinderbox's main menu.

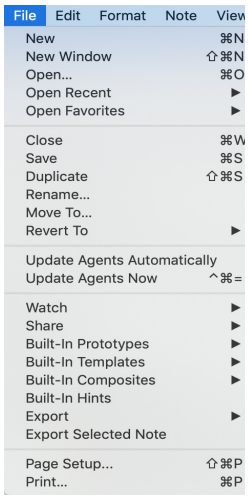
- Tinderbox 9 menu
- File menu
- Edit menu
- Format menu
- Note menu
- View menu
- Stamps menu
- Window menu
- Help menu

Tinderbox 9 menu

The standard macOS application menu. Menu items:

- **About Tinderbox.** Opens the About Tinderbox dialog.
- **What's New…** Opens the What's New dialog.
- **Tinderbox News…** Opens the Tinderbox News dialog
- **Preferences…** (⌘,) Opens Tinderbox Preferences.
- **Register Tinderbox…** Opens the Register tab of Tinderbox Preferences.
- **Services.** Standard Mac Services sub-menu. Contents will depend on the state and selection of the current window.
- **Hide Tinderbox.** (⌘**H**) Toggle: hides/unhides all Tinderbox windows.
- **Hide Others.** (⌥⌘**H**) Hides all non-Tinderbox windows.
- **Show All.** Reveals all windows with Tinderbox windows in front.
- **Quit Tinderbox.** (⌘**Q**) Close the application.
  - Hold the Option key (⌥) to see alternate menu item: **Quit and Keep Windows**.

File menu



The File menu contains the following items:

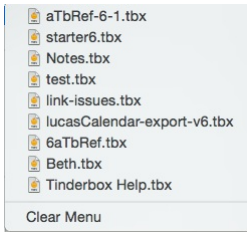
- **New**. (⌘**N**) Creates a new Tinderbox document file. The new file's document preferences will be based on Tinderbox's stored defaults.
- **New Window**. (⇧⌘**N**) Opens an additional window for the current document.
- **Open…** (⌘**O**) Opens an existing Tinderbox document file. A file open dialog is shown so that you can locate the file. The default folder offered will be the last-used folder.
- **Open Recent**. A sub-menu listing recently used Tinderbox files. The list holds up to a (default) 10 filenames; the maximum list size can be changed via the config.xml file.
- **Open Favorites**. A sub-menu listing Tinderbox files stored in Tinderbox's application support's ' favorites' folder.
- **Close**. (⌘**W**) Closes the current Tinderbox document window. Closing the last window in the file (or current file if more files are open) will prompt a save before closing dialog.
  - Hold Shift key (⇧) to see alternate menu item: **Close File "filename"**. Where "filename" is the name of the current open TBX documents, or the name of the TBX file owning the current window if more than one file is open. Clicking closes that file. There is no confirmation, any outstanding changes are saved automatically.
  - Hold Option key to see alternate menu item: **Close All**. Close all TBX documents currently open.
- **Save**. (⌘**S**) Invokes the file save routine (current filename and location) for the file, or current file if more than one. The default folder offered will be the last-used folder.
- **Duplicate**. (⇧⌘**S**) Duplicates the existing file in situ, using the existing filename with ' coy' appended to the name, and then opens it in Tinderbox in front of the source document window.
  - Hold Option (⌥) key to see alternate menu item: **Save As…** (⌥⇧⌘**S**) Opens a file save dialog to allow selection of the save location and/or new file name. The default folder offered will be the last-used folder.
- **Rename…** Allows the current TBX documents OS filename to be edited whilst the file is open, via the document window's caption bar.
- **Move To…** Allows the shored location of the current document to be changed without closing the file, via a drop-down pane from the document window's caption bar.
- **Revert To**. A sub-menu offering previous version to which to revert the current document.
- **Update Agents Automatically**. On/off toggle. If 'on' a tick is shown in the menu left margin for this item. When 'on' Tinderbox automatically runs agents on a regular cycle. Note there is no Preference or Attribute to control this setting, though this equates to a doc-level preference. When 'off' ($AgentPriority is 0), the Agents and Rules Inspector indicates agents are on manual update only.
- **Update now**. (⌃⌘=) Invokes an immediate running of all agents (and all edicts are run once). Works regardless of the above setting and regardless of $AgentPriority, unless the agent is turned off (when $AgentPriority is 0). It can be useful if:
  - an agent appears to not be functioning and you wish to be sure it has run its code.
  - there are chains of interacting agents. In this case it may require invoking several updates to pass data along the chain.
  - some agents are on low priority and thus done fire every update cycle.
  - you need an on-demand single cycle of agents when automatic updates are off. Note though, that if you have agents interrogating other agents one agent cycle may not refresh all agents to the degree assumed.
- **Watch**. Opens a sub-menu of watched application/folder options.
- **Share.** Opens a sub-menu of Sharing options. If some note(s) are selected, those notes will be shared with compatible programs. If no notes are selected, the Tinderbox document itself will be shared. The options listed in the sub-menu will depend on the range of apps/services installed on the Mac's current user account.
- **Built-In Prototypes**. Opens a sub-menu listing a choice of built-in prototypes that can be added to the current document.
- **Built-In Templates**. Opens a sub-menu listing a choice of built-in (export) templates that can be added to the current document.
- **Built-In Composites**. Opens a sub-menu listing a choice of built-in Composite sets of notes that can be added to the current document.
- **Built-In Hints**. Inserts the built-in Hints container and set of sub-items.
- **Export**. Opens the Export sub-menu.
- **Export Selected Note(s)**. Opens a dialog for exporting individual notes (defaults to $HTMLExportFileName). See more on this form of export.
- **Page Setup…** (⇧⌘**P**) Calls the OS page setup dialog.
- **Print…** (⌘**P**) Invokes OS printing dialogs.

Sub-menus:

- Recently used files sub-menu
- Favorites sub-menu
- Revert To sub-menu
- Watch sub-menu
- Share sub-menu
- Built-in Prototypes sub-menu
- Built-in Templates sub-menu
- Built-in Composites sub-menu
- Export sub-menu

Recently used files sub-menu

This sub-menu of the File menu, contains a list of recently used files (most recent first). At the bottom there is an option to clear the existing menu, after which the list will begin to re-populate again.

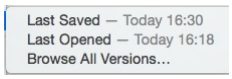


Favorites sub-menu

This sub-menu of the File menu holds a list of any TBXs stored in the user's ' favorites' folder.

Revert To sub-menu

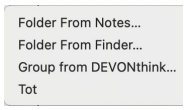
This sub-menu of the File menu. Not all listed options are necessarily shown depending on the state of the session and/or use of Time Machine:



- **Last Saved**. Date/time of file. The last explicitly saved version during the current session, i.e. deliberately saved via the user.
- **Last Opened**. Date/time of file. The last stored version prior to the current session.
- **Browse All Versions…** Opens TimeMachine (if available).

Watch sub-menu

This sub-menu of the File menu allows the se-up of various type of watched folders:

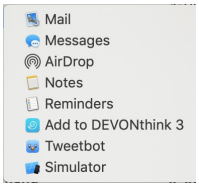


- **Folder From Notes…** This will create a top-level container that holds notes imported from a designated folder in the Notes application.
- **Folder from Finder…** This will create a top-level container that holds notes imported from a designated folder.
- **Group from DEVONthink…** This will create a top-level container that holds notes representing the contents of the watched DEVONthink group. Do read further notes on this feature as consideration should be given to the types of file in the watched group.
- **Tot**. This will create a top-level container that watches (by default) all 7 Tot notes. The macOS Tot app needs to be installed.

Any/all of these containers will watch the corresponding application periodically and will automatically add new notes and update notes which have changed. Changes made in Tinderbox are **not** forwarded back to the external application. This menu is greyed out if focus is not in the view pane.

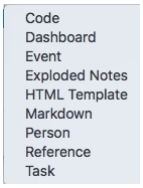
Share sub-menu

This is a sub-menu of the File menu. The options listed in the sub-menu will depend on the range of apps/services installed on the Mac's current user account.
If some note(s) are selected, those notes will be shared with compatible programs. If no notes are selected, the Tinderbox document itself will be shared.



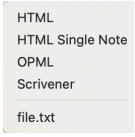
Built-in Prototypes sub-menu

This sub-menu of the File menu lists a number of predefined prototypes notes built-into the Tinderbox application for easy addition of new prototypes. Any shared prototypes are listed under standard built-in items.See more on built-in prototypes.



Built-in Templates sub-menu

This sub-menu for the File menu lists a number of predefined export template note built-into the Tinderbox application for easy addition templates for various export tasks. See more on built-in templates.
Any items seen listed below a ruler are templates stored in the application support ' templates' folder.

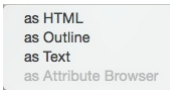


Built-in Composites sub-menu

A sub-menu of the File menu, this lists all built-in Composites available for use in the current document.

Export sub-menu

This sub-menu of the File menu offers a series of format choices to export the whole of the current document:

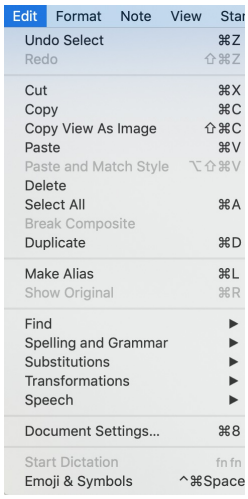


- **HTML**. Opens the HTML Export folder dialog. Exports the current document according to current HTML export configurations (templates, attributes, etc.).
- **Outline**. Opens the Export as Outline panel. Exports the current document in Outline form. Note that for Scrivener or OPML use the 'Text' option, below.
- **Text**. Opens the Export as Text panel. This offers further choices as to export format and scope of note(s) exported.
- **Attribute Browser**. Opens the Attribute Browser Export dialog. More on Attribute Browser export. The first line of an Attribute Browser export contains the names of the attributes exported in each column.

These options are available when the HTML or Preview text sub-panes are selected.

Edit menu

all the agents you can use use theThe Edit menu contains the following items:



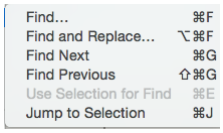
- **Undo *(last action)***. (**⌘Z**) Undo the *last action*.
- **Redo *(last action)***. (**⇧⌘Z**) If no *last action* or *last action* cannot be buffered, this item reads "Can not redo".
- **Cut**. (**⌘X**) Removes the selection and copies it to the clipboard.
- **Copy**. (**⌘C**) Copies the current selection to the clipboard.
- **Copy View As Image**. (**⇧⌘Z**) Copies current main view contents (except for Attribute Browsers) to clipboard as an image in both PDF and PNG format; the saved image includes the whole view (e.g. non-visible parts of a map). No save dialog is shown, simply open an image editor and create a new document from the clipboard or paste into a new empty image document. The exported data is in vector form, and can be edited as such if pasted into a vector-capable image editor. The image data will be rasterised by the image editor if it is pasted into a raster image image document, the PNG version of the copied data will be used. Preview will use the PDF data if creating a new document from the clipboard. For Outline and Chart views the image shows the tree in its current state of expansion; collapsed branches remain so in the image.
- **Paste**. (**⌘V**) Pastes clipboard contents to current cursor position or replaces existing selection.
- **Paste and Match Style**. (**⌥⇧⌘V**) This will cause the font typeface and size of the pasted text to pick up the settings in the note immediately preceding the paste, whether note default or further altered before the paste. Font colour and bold/italics are unaffected.
- **Delete**. Deletes current selection.
- **Select All**. (**⌘A**) Select all text in text windows or all notes in current view in view windows.
  - Hold Control+Option keys (**^⌥**)down as well to see alternate menu item: **Deselect All**. As it says, everything in the current view pane is deselected.
- **Break Composite**. Greyed out unless a composite is selected (or one or more items that are part of a composite). Breaks the composite into stand-alone notes.
- **Duplicate**. Duplicates the current note(s). The new note is called "[original name] copy". The new note is inserted as the next sibling in views except the the Map view where it is inserted down and right slightly overlapping the source note. Any sequential attributes are incremented, otherwise all attributes are those of the source note. Multiple selections can be duplicated.
- **Make Alias**. (**⌘L**) Creates an alias of the currently selected note (only available with a single note is selected). The new alias is inserted as the next sibling in views except the the Map view where it is inserted down and right slightly overlapping the source note. Multiple individual aliases can be made from multiple selections
- **Show Original**. (**⌘R**) Available when an alias is selected. Locates the position of the source note for the alias.
  - Hold Shift key (⇧) to see alternate menu item: **Show Original In New Tab**. The original is shown, but a new tab is opened and selected.
- **Find**. Opens the Find sub-menu.
- **Spelling and Grammar**. Opens the Spelling and Grammar sub-menu.
- **Substitutions**. Opens the Substitutions sub-menu.
- **Transformations**. Opens the Transformations sub-menu.
- **Speech**. Opens the Speech sub-menu.
- **Document Settings...** (**⌘8**) Opens Document Settings.
- **Start Dictation**. (**fnfn** - i.e. 'fn' twice) Commence Dictation using the macOS' built-in dictation facilities.
- **Emoji & Symbols...** (**^⌘Space**) Open the Emoji & Symbols pop-over.

Sub-menus:

- Find sub-menu
- Spelling and Grammar sub-menu
- Substitutions sub-menu
- Transformations sub-menu
- Speech sub-menu

Find sub-menu

This sub-menu of the Edit menu, contains the following items:

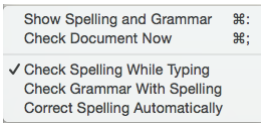


- **Find...** (**⌘F**) Opens the Find controls, in either main view or text pane.
- **Find and Replace...** (**⌥⌘F**) (Text pane only). Exposes the Find and Replace toolbar in the Text pane.
- **Find Next**. (**⌘G**) Moves selection to next item (main view) or string (text pane) matching the find value.
- **Find Previous**. (**⇧⌘G**) Moves selection to next item (main view) or string (text pane) matching the find value.
- **Use Selection for Find**. (**⌘E**) Opens the Find using the current selection. If focus is in the text pane this loads the find string for both the text pane search bar and the view pane search bar.
- **Jump to Selection**. (**⌘J**) Scroll to the selection in the pane.

Many of these options are listed mainly for their equivalent shortcuts enabling searches without needing mouse or trackpad.

Spelling and Grammar sub-menu

This sub-menu of the Edit menu, contains the following items (all items except first require text pane focus):



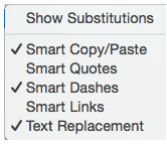
- **Show/Hide Spelling and Grammar**. (**⌘:**) Toggles the OS Spelling and Grammar dialog.
- **Check Document Now**. (**⌘;**) This causes any items not matching the host OS' spelling list to be underlined in red.
- **Check Spelling While Typing**. Clicking this causes "Check spelling" to permanently turned on. The item acts as an on/off toggle (default = off). When turned 'on' a tick is shown in the left margin of the menu. This is set (per file?) on this menu. There is no preference setting. This can be set in the General tab of Document Settings. Individual notes can be excluded from setting their $NoSpelling attribute to `true`. The text pane respects the Document settings' checkbox "check spelling at you type": if a note is selected, and if the note uses the default value of $NoSpelling, changing this setting immediately updates spell checking for that note.
- **Check Grammar with Spelling**. Turns on grammar error detection alongside spelling prompts.
- **Correct Spelling Automatically**. Tinderbox will automatically correct any spelling mistakes it detects.

If you need to choose/alter the language used for spell-checking, open OS System Preferences, Language & Text preferences, Text tab. The latter's "Automatic by Language" option for Spelling, seems to work contextually at around sentence scope.

See more on the scope of spell checking.

Substitutions sub-menu

This sub-menu of the Edit menu, contains the following items (all items require text pane focus) that work with OS-level settings. All settings are on (ticked) by default and if disabled only remain so while the note has focus. On next selection of the note all items will be re-enabled. Setting are:

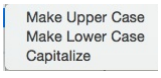


- **Show Substitutions**. Shows the OS Substitutions dialog.
- **Smart Copy/Paste**. Akin to Tinderbox's Paste and Match Style, this attempts to match the styling of pasted data to match existing text.
- **Smart Quotes**. Straight single and double quotes are automatically replaced by 'curly' typographic versions, i.e. ' smart quotes'. The toggle also sets/unsets the note's $SmartQuotes. Tinderbox offers an override for OS quote settings at document level via the Text tab Document Settings and at note level via $SmartQuotes or this menu option. $SmartQuotes also controls smart dashes.
- **Smart Dashes**. This automatically converts two or more successive hyphens (or minus signs) into a single 'em' dash, i.e. —, this ' smart dashes'. This is unsuitable for code notes and templates but can only be disabled here. $SmartQuotes also controls smart dashes.
- **Smart Links**. If enabled this will auto-detect URLs in text and make rich-text URLs, i.e. ' smart links'; sets $SmartLinks.
- **Text Replacement**. Automatically applies an OS level text replacements, e.g. for common typos.

OS substitution settings tend to vary from OS release to release. If relying heavily on these features it is a good idea to research the settings for your OS (Apple does not seem to have official documentation of the features).

Transformations sub-menu

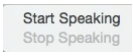
This sub-menu of the Edit menu, contains the following items (all items require text pane focus):



- **Make UpperCase**. Make the current text selection all uppercase.
- **Make Lower Case**. Make the current text selection all lowercase.
- **Capitalize**. Make the first character of every word in the current text selection uppercase.

Speech sub-menu

This sub-menu of the Edit menu, contains the following items (all items require text pane focus):

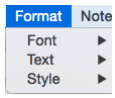


- **Start Speaking**. Start reading the current note $Text aloud using the macOS text-speech-feature.
- **Stop Speaking**. Cease speaking the current note.

Format menu

The Format menu contains the following items:

- **Font**. Sub-menu.
- **Text**. Sub-menu.
- **Style**. Sub-menu.

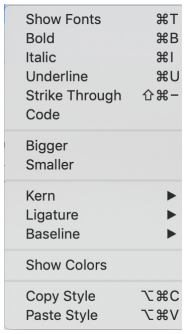


Sub-menus:

- Font sub-menu
- Text sub-menu
- Style sub-menu

Font sub-menu

This sub-menu of the Format menu contains these items:



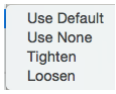
- **Show/Hide Fonts**. (**⌘T**) Toggle the OS Font dialog.
- **Bold**. (**⌘B**) Toggles the selected text to **Bold**. Note that the selection's current font must have a Bold variant.
- **Italic**. (**⌘I**) Toggles the selected text to *Italic*. Note that the selection's current font must have a Italic variant.
- **Underline**. (**⌘U**) Toggles the selected text to Underline. Greyed out unless Text pane has focus.
- **Strike Through**. (⇧⌘-) Toggles the selected text to ~~Strikethrough~~.
- **Code**. Sets the selected text to a `monospace 'Code' font` (font choice set via $CodeFont). Greyed out unless Text pane has focus. From v9.6.0, the option acts as a toggle.
- **Bigger**. Increases selected text size by 1 point (e.g. 14pt to 15 pt).
- **Smaller**. Decreases selected text size by 1 point (e.g. 14pt to 13 pt).
- **Kern**. Opens the Kern sub-menu.
- **Ligature**. Opens the Ligature sub-menu.
- **Baseline**. Opens the Baseline sub-menu.
- **Show/Hide Colors**. Toggles the OS 'Colors' colour picker palette which can be used to colour selected $Text (which is $TextColor by default).
- **Copy Style**. (**⌥⌘C**) Copies the text style of the current selection (including the paragraph ruler) to the clipboard.
- **Paste Style**. (**⌥⌘V**) Reformats the selection (including the paragraph ruler) using the text style on the clipboard, if one is found.

Sub-menus:

- Kern menu
- Ligature menu
- Baseline menu

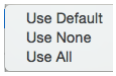
Kern menu

This sub-menu of the Font menu offers standard kerning choices.



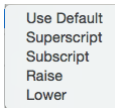
Ligature menu

This sub-menu of the Font menu offers standard Ligature choices.



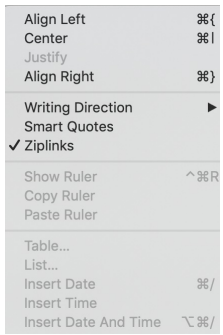
Baseline menu

This sub-menu of the Font menu offers standard Baseline choices.
The subscript and superscript commands in Format ▸ Font ▸ Baseline reduce the font size of the selected text by 25%.



Text sub-menu

This is a sub-menu of the Format menu. In Map view the left/center/right alignment setting is applied to the *titles* of selected notes (and so changing $NameAlignment). The menu contains these items:



- **Left Align**. (**⌘{**) Align the current paragraph to the left.
- **Center**. (**⌘|**) Centre align the current paragraph.
- **Justify**. Justify align the current paragraph.
- **Right Align**. (**⌘}**) Align the current paragraph to the right.
- **Writing Direction**. Opens the Writing Direction sub-menu.
- **Smart Quotes**. Toggles replacement of straight single and double quotes with typographic equivalents (set $SmartQuotes). This setting affects $SmartQuotes (and SmartDashes) but does not alter SmartLinks (as these have a discrete $SmartLinks control).
- **Ziplinks**. Toggles whether the selected note(s) allow zip method linking (i.e. the $Ziplinks setting for the note).
- **Show/Hide Ruler**. (**^⌘R**) Toggles the text pane ruler.
- **Copy Ruler**. Copy the current paragraph's ruler settings.
- **Paste Ruler**. Paste clipboard ruler settings on the current selected paragraph.
- **Table...** Inserts a table into $Text and opens the Table dialog.
- **List...** Opens the List panel.
- **Insert Date**. (**⌘/**) This causes the current date, in short format (e.g. 8/3/06), to be added at the insertion point in a current note's text. Short date day/month order depends on the user's OS settings.
- **Insert Time**. This causes the current time, in short format (e.g. 22:16), to be added at the insertion point in a current note's text.
- **Insert Date And Time**. (**⌥⌘/**) This causes the current date and time, date in short format (e.g. 8/3/06 22:16), to be added at the insertion point in a current note's text.

More on Date formats.
Sub-menus:

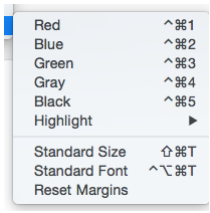
- Writing Direction menu

Writing Direction menu

This sub-menu of the Text menu offers controls for the direction of writing used by either the paragraph (containing the selection) or just the selection:

Style sub-menu

This sub-menu of the Format menu contains these items:



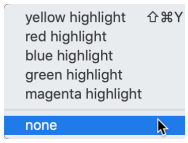
- **Red**. (^⌘**1**) Colour the selected text red.
- **Blue**. (^⌘**2**) Colour the selected text blue.
- **Green**. (^⌘**3**) Colour the selected text green.
- **Gray**. (^⌘**4**) Colour the selected text grey.
- **Black**. (^⌘**5**) Colour the selected text black. If the default text colour ($TextColor) is not black the default colour is applied.
- **Highlight**. Opens the Highlight sub-menu.
- **Standard Size**. (⇧⌘**T**) Sets the selection to the default font size ($TextFontSize). If selected from View pane, the styling is applied to the whole text of the current note. A text selection is not required so a note with no $Text can be reset.
- **Standard Font**. (^⌥⌘**T**) Sets the selection to the default font ($TextFont). Note that this destroys any bolding or italics as these are set using variant font faces. If selected from View pane the styling is applied to the whole text of the current note. Standard Font is more aggressive in removing indentation, background colours, text colours, embedded tables and list formatting. This is often what one wants when pasting from formatted sources such as Web pages. The process tries to respect passages that are bold or italic; it changes the font family to the note's default font family, using the note's text size. A text selection is not required so a note with no $Text can be reset.
- **Reset Margins**. This resets paragraphs in the selected range to use the standard margins and line spacing.

Sub-menus:

- Highlight sub-menu

Highlight sub-menu

The Highlight sub-menu of the Style menu contains the following items:

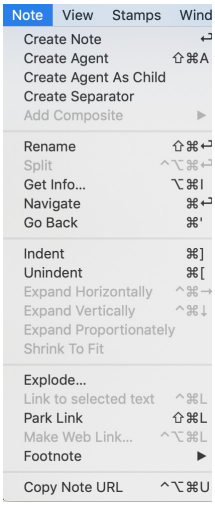


- **yellow highlight** (⇧⌘**Y**)
- **red highlight**
- **blue highlight**
- **green highlight**
- **magenta highlight**
- **none**

Highlight commands are toggles for the currently selected text.

Note menu

The note menu contains the following items (most are greyed out if focus is not in the main view):



- **Create Note** (↩) Create a new note, as next sibling to the current selection. If no selection, placement varies by view type.
  - Hold Shift key (⇧) to see alternate menu item: **Create Child Note**. The note added is created as a child of the current note and the last sibling child if children already exist.
- **Create Agent** (⇧⌘**A**) Create a new Link type honouring operators, as next sibling to the current selection. If no selection, placement varies by view type.
- **Create Agent As Child**. Create a new agent, but as a child of the current note.
- **Create Separator** (Outline only) **/ Create Adornment** (Map only). View dependent:
  - Outline view: adds a new note as next sibling with the 'separator' option pre-ticked.
  - Map View: adds an adornment.
  - Other views: shows separator option greyed out.
- **Add Composite** (Map only, greyed out otherwise). If composites are present, links to the Composites sub-menu.
- **Rename…** (⇧⌘↩) Places selected item's title in Edit-in-Place mode.
- **Split**. (^⌥⌘↩) This will split a note in two based on the selection in the text pane (cursor focus must be in the text pane). If the selection is empty, the note is split at the insertion point, otherwise the selection is extracted to a new note. The title of the new note is taken from the first sentence of its text.
- **Get Info…** (⌥⌘**I**) Opens the Get Info pop-over for the current note.
- **Navigate**. (⌘↩) Navigate will follow the first basic link from the current note, selecting its destination. For to-text links this scrolls the destination note to the link point and highlights the linked word.
- **Go Back**. (⌘') Go Back will return to the note most recently selected. Disabled when the history is empty.
- **Indent**. (⌘**]** or →|) Indents the selection one outline level, making them children of the items preceding the selection. Greyed out in map view. Using this from the text pane will indent the current paragraph.
- **Unindent**. (⌘**[** or ⇧→|) Moves the selection up one outline level, making them siblings of the parent item of the selection. Greyed out in map view. Using this from the text pane will un-indent the current paragraph.
- **Expand Horizontally** (^⌘→) (Map view only—otherwise greyed out). Tells Tinderbox to attempt to widen the note's map icon to display the whole note Name (title). The selected note's icon is expanded at the right side, maintaining existing height and X/Y origin. May be used in conjunction with Expand Vertically (below). The revised Map note width is retained for the test of the session and persisted if the TBX file is saved. If multiple notes are selected all notes are changed accordingly. This feature equates to the action of the 'Expand horizontally' option in Document Settings.
- **Expand Vertically** (^⌘↓) (Map view only—otherwise greyed out). Tells Tinderbox to attempt to increase the depth (height) of the note's map icon to display the whole note Name (title). The selected note's icon is expanded downward, maintaining existing width and X/Y origin. May be used in conjunction with Expand Horizontally (above). The revised Map note height is retained for the test of the session and persisted if the TBX file is saved. If multiple notes are selected all notes are changed accordingly. This feature equates to the action of the 'Expand vertically' option in Document Settings.
- **Expand Proportionally**. (Map view only—otherwise greyed out). Tells Tinderbox to expand the note(s) proportionately, i.e. using the current height:width ratio, if the note name is too long to fit in the current map icon. If multiple notes are selected all notes are changed accordingly. This feature equates to the action of the 'Expand proportionally' option in Document Settings.
- **Shrink To Fit**. (Map view only—otherwise greyed out). Shrinks the size of the note's *title* text ($MapTextSize) so as to fit the title into the current map icon. This option is available when multiple notes are selected. If multiple notes are selected all notes are changed accordingly. This feature equates to the action of the 'Use smaller type' option in Document Settings.
- **Explode…** (⇧⌘**E**) opens the Explode pop-over.
- **Link to selected text** / **Link to "[note name]"**. (^⌘**L**) Only active if (a) the focus is in the $Text area, (b) there is a $Text selection and (c) there is an existing note whose $Name (**note name**) is a case-sensitive match for the selected text. If so, the destination note name is shown in the menu caption. When clicked, this creates a link, anchored on the current text selection, to the matching note. If the match is not unique, i.e. where are several same-named notes, the link is created to the first match when sorted by $OutlineOrder.
- **Park link**. (⇧⌘**L**) Places a basic link, originating from the current note, in the main view parking link (any existing parked link is replaced).
- **Make Web Link…** (^⌥⌘**L**) Invokes the Create (Web) Link dialog for the current note.
- **Footnote…** Opens the Footnote sub-menu.
- **Copy Note URL**. (^⌥⌘**U**) Copies to the clipboard the Tinderbox URL schema link to address the current note form outside Tinderbox ($NoteURL).

Sub-menus:

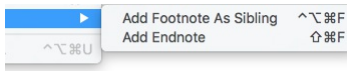
- Composites sub-menu
- Footnote sub-menu

Composites sub-menu

A sub-menu of the Note menu. This lists all (named) Composites defined for the current doc (i.e. within the '/Composites') container and with a composite name.

Footnote sub-menu

This is a sub-menu of the Note menu. These options allow the creation of footnotes from the current:

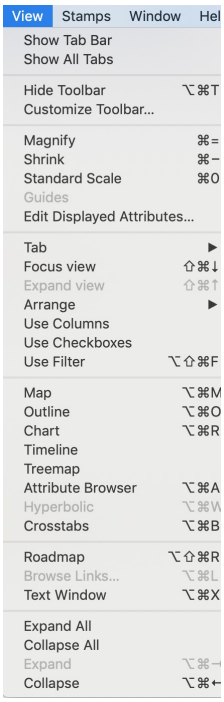


- **Add Footnote As Sibling**. (^⌥⌘**Z**) The new note is created as the next sibling to the current note.
- **Add Footnote As Endnote**. (⇧⌘**Z**) The new note is created as a child of a 'Notes' container created as a sibling of the current note. Footnotes are described in more detail in Creating Footnote notes.

Note: the menu items are greyed out if the focus is not in the current note **and** there is no $Text selection.

View menu

The View menu contains the following items:



- **Show/Hide Tab Bar**. Toggles visibility of the document toolbar. If enabled this shows a tab per TBX document.
- **Show All Tabs/Exit Tab Overview**. Toggles an overview grid showing thumbnails of all tabs in the current document.
- **Show/Hide Toolbar**. (⌥⌘**T**) Toggles visibility of the window toolbar.
- **Customize Toolbar…** Opens the Customize Toolbar panel to customise the window toolbar.
- **Magnify**. (⌘=) If focus is in the:
  - View pane, this increases the view zoom level by a factor of 1.
  - Text pane, this changes the font size ($TextFontSize) of the selected text, i.e. with no selection nothing happens. Note: this effect changes the text itself rather than zooming the pane's contents as a whole.
- **Shrink**. (⌘-) If focus is in the:
  - View pane, this decreases the main view zoom level by a factor of 1.
  - Text pane, this changes the font size ($TextFontSize) of the selected text, i.e. with no selection nothing happens. Note: this effect changes the text itself rather than zooming the pane's contents as a whole.
- **Standard Scale**. (⌘**0**) Returns the main view to default zoom level.
- **Guides**. (Map view only) Toggles use of view alignment guides, including snap-to-grid.
- **Edit Displayed Attributes…** Opens the current note's Define new Displayed Attributes pop-over. This allows the adding, deletion or re-ordering of the note's Displayed Attributes table.
- **Tab**. Tab sub-menu.
- **Focus view**. (⇧⌘↓) 'Hoist' the view pane focus. Map view: drill down to show child map of current container. Outline/Chart views: set view so this item is the root of the current view (i.e. one level down the outline). A breadcrumb bar is shown if the view focus is not at root.
- **Expand view**. (⇧⌘↑) Un-hoist the view pane focus. Map view: go up to show parent map of current map. Outline/Chart views: set view root at the parent container of the current view root container (i.e. one level up the outline). A breadcrumb bar is shown if the view focus is not at root.
- **Arrange**. Arrange sub-menu.
- **Use Columns**. Toggle on/off column data display, *Outline view only*.
- **Use Checkboxes**. Toggle on/off checkboxes, *Outline view only*.
- **Use Filter**. (⌥⇧⌘**F**) Enable a view pane filter, *Outline view only*.
- **Map**. (⌥⌘**M**) Sets the current tab's view pane to Map view.
- **Outline**. (⌥⌘**O**) Sets the current tab's view pane to Outline view.
- **Chart**. (⌥⌘**R**) Sets the current tab's view pane to Chart view.
- **Timeline**. Sets the current tab's view pane to Timeline view.
- **Treemap**. Sets the current tab's view pane to Treemap view.
- **Attribute Browser**. (⌥⌘**A**) Sets the current tab's view pane to Attribute Browser view.
- **Hyperbolic**. (⌥⌘**W**) Sets the current tab's view pane to Hyperbolic view.
- **Crosstabs**. (⌥⌘**B**) Sets the current tab's view pane to Crosstabs view.
- **Roadmap**. (⌥⇧⌘**R**) Opens the Roadmap view pop-over for the current note.
- **Browse Links…** (⌥⌘**L**) Opens the Browse Links pop-over for the current note. This is only enabled if the note has one or more outbound links.
- **Text window**. (⌥⌘**X**) Opens a stand-alone text window for the current note.
- **Expand All**. In Outline or Chart views, this expands (opens) the entire outline (i.e. all items are displayed.
- **Collapse All**. In Outline or Chart views, this collapses (opens) the entire outline (i.e. only root-level items are shown).
- **Expand**. (⌥⌘→) In Outline or Chart views, this expands (opens) the view of the current note (i.e. shows its children).
- **Collapse**. (⌥⌘←) In Outline or Chart views, this collapses (closes) the view of the current note (i.e. hides its children).

Sub-menus

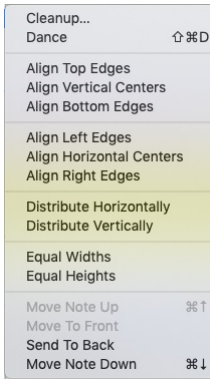
- Tab sub-menu
- Arrange sub-menu

Tab sub-menu

A sub-menu of the View menu. This menu has the following items:

- **Show Tabs/Hide Tabs**. Hides or reveals the document window's Tab bar.
- **Gallery…** (⌥⌘**G**) Opens the document window's Gallery of saved tabs.
- **New Tab**. Creates a new tab, to the right of all existing tabs, and then selects it. If an alias is selected, this will open a new tab on the original note of that alias.
- **Close Tab**. Closes the current tab. From v9.5.0 this is enabled from the Attribute Browser view and from the Text Pane
- **Previous**. (⌥⌘**[**) Selects the previous tab (i.e. to the left of the current tab).
- **Next**. (⌥⌘**]**) Selects the next tab (i.e. to the right of the current tab).

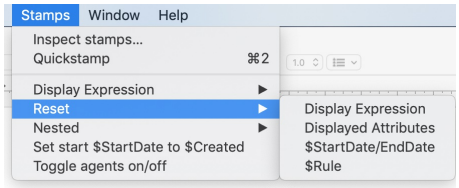
Arrange sub-menu



A sub-menu of the View menu. In previous versions this was menu called 'Align'. For quick manual alignment, do not overlook the map's grids and guides. This menu has items for controlling the layout of Map (only) items:

- **Cleanup…** Cleanup tab in the view pane.
- **Dance**. (⇧⌘**D**) Starts/stops a force-directed layout of a Map view. Calling this opens the Dance pop-over.
- **Align Top Edges**. Align the top edges of all selected items.
- **Align Vertical Centers**. Align the vertical centres of all selected items.
- **Align Bottom Edges**. Align the bottom edges of all selected items.
- **Align Left Edges**. Align the left edges of all selected items.
- **Align Horizontal Centers**. Align the horizontal centres of all selected items.
- **Align Right Edges**. Align the right edges of all selected items.
- **Distribute Horizontally**. Distribute the selected items equally in the horizontal plane.
- **Distribute Vertically**. Distribute the selected items equally in the vertical plane.
- **Equal Widths**. Set selected items to the same width (largest in selection).
- **Equal Heights**. Set selected items to the same height (largest in selection).
- **Move Note Up**. (⌘↑) In Outline views this moves the current note up one place (at sibling level).
- **Move To Front**. In Map views this moves the current note in front of all other notes (i.e. first by outline order).
- **Sent To Back**. In Map views this moves the current note behind all other notes (i.e. last by outline order).
- **Move Note Down**. (⌘↓) In Outline views this moves the current note down one place (at sibling level).

Stamps menu



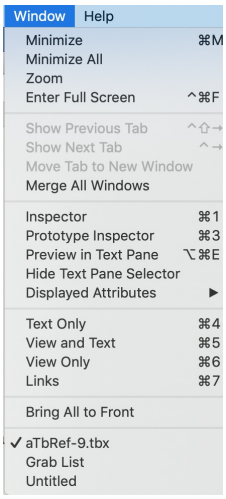
The Stamps menu contains the following items:

- **Inspect Stamps…** Opens the Stamps tab of the Document Inspector.
- **Quickstamp**. (⌘**2**) Opens the Quickstamp tab of the Properties Inspector.
- **[list of the current file's user-defined stamps]**.

Clicking a listed stamp applies that stamp to the currently selected main view items.
The stamp listing order is based on that in the Stamps Inspector, can be modified by drag-reordering the listing in Stamps Inspector.
Stamp groups are collected into discrete sub-menu(s).

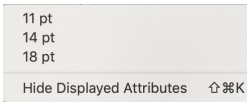
Window menu

The Window menu contains the following items:



- **Minimize**. (⌘**M**) Minimise the current items to the Dock.
- **Minimize All**. Minimise all Tinderbox windows to the Dock.
- **Zoom**. Toggle zoom state (normal/full width) of the current window.
  - Hold **Option** key to see alternate menu item: **Zoom All**. Toggle zoom state (normal/full width) of all Tinderbox windows.
- **Tile Window to Left of Screen** . (macOS 10.15+) Moves and tiles current app window to left half/side of the current screen and show a Mission control view in the right half of the screen; current screen only if more than one.
  - Hold **Option** key to see alternate menu item: **Move Window to Left Side of Screen** . Moves and tiles current app window to left half/side of the current screen; current screen only if more than one. Other apps are unaffected. For standalone windows this replaces the 'Tile Window…' as the default and only option.
- **Tile Window to Right of Screen** . (macOS 10.15+) Moves and tiles current app window to right half/side of the current screen and show a Mission control view in the right half of the screen; current screen only if more than one.
  - Hold **Option** key to see alternate menu item: **Move Window to Right Side of Screen** . Moves and tiles current app window to right half/side of the current screen; current screen only if more than one. Other apps are unaffected. For standalone windows this replaces the 'Tile Window…' as the default and only option.
- **Move to [different display name]** . (macOS 10.15+) click to move the current document window to the indicated display. If more than one display is used, each additional display will be listed as discrete option.
- **Enter/Exit Full Screen**. (^⌘**F**) Toggles the (current) screen into/out of full screen mode.
- **Show Previous Tab**. (^⇧⌘→) Selects the previous (to left) document tab; greyed out when only single document open.
- **Show Next Tab**. (^⌘→) Selects the next (to right) document tab; greyed out when only single document open.
- **Move Tab to New Window** . Moves the current document tab into a new single-document window; greyed out when only single document open.
- **Merge All Windows** . Merges all open Tinderbox documents into a single window (i.e. single document tab bar); greyed out when only single document open.
- **Inspector**. (⌘**1**) Opens the Inspector window at the last used tab (default: Appearance Inspector, Interior sub-tab).
- **Prototype Inspector**. (⌘**3**) Opens the Properties Inspector window, with the Prototype sub-tab selected.
- **Preview in Text Pane/Export Pane/Edit in Text Pane** . (⌥⌘**E**) Switch text tab focus to the next tab (Text → Preview → Export → Text, etc.). The tab context is switched even if the tab pane selectors are not visible).
- **Show/Hide Text Pane Selector**. Toggles visibility of the Text pane's sub-tabs (default: panes are hidden).
- **Displayed Attributes**. Displayed Attributes sub-menu.
- **Text Only**. (⌘**4**) Move window vertical divider so main view is hidden and the text pane fills the window. Focus automatically switches to the text pane.
- **View and Text**. (⌘**5**) Move window vertical divider so the main view and text pane are allotted equal space in the window.
- **View Only**. (⌘**6**) Move window vertical divider so text pane is hidden and the main view fills the window. Focus automatically switches to the view pane.
- **Links**. (⌘**7**) Display or hide the Links panel in the bottom of the text pane.
- **Bring All to Front** . Brings all Tinderbox windows to the front, i.e. on top of any other application's windows.
  - Hold Option key to see alternate menu item: **Arrange in Front** . Brings all Tinderbox windows to the front, i.e. on top of any other application's windows, in a cascade of overlapping windows.
- [List of all open Tinderbox windows]. Brings the Tinderbox window whose title is clicked to the front of the Tinderbox windows.

Displayed Attributes sub-menu



This sub-menu of the Window menu contains a list of suggested point sizes for rendering the Displayed Attributes table of the current note, even if not yet currently shown or populated. The menu offers a series of size values for Displayed Attributes and and a toggle for display of the Displayed Attributes table:

- 11pt (default)
- 14pt
- 18pt

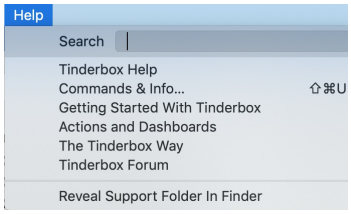
(ruler)

- **Hide/Show Displayed Attributes/No Displayed Attributes**. Toggles visibility of the selection's Displayed Attributes (assuming some are defined). The state is stored in and inherited via $HideDisplayedAttribute. If the note currently ha no Displayed Attribute defined, 'No Displayed Attributes' is shown, greyed out. This applies to all selected notes.

The underlying attribute holding the size of Displayed Attributes is $DisplayedAttributesFontSize.

Displayed Attributes replace now-deprecated Key Attributes. Apart from the name of menus, attributes, etc., the feature is the same, just more explicitly described.

Help menu



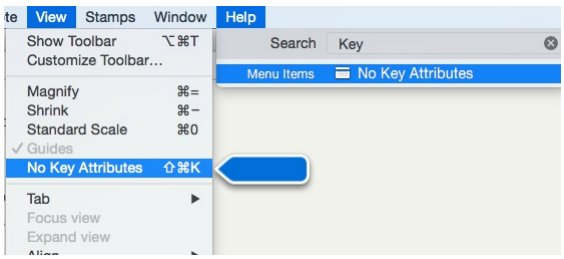
The Help menu contains the following items:

- **Search**. Search box gives access to simple Apple Help match listing. When a value is typed here the matches are shown beneath replacing existing menu items. Clear the search box to re-show default menu items.
- **Tinderbox Help**. This opens the Tinderbox Help window. Release notes are included within Help.
- **Commands & Info…** (⇧⌘**U**) This opens the document window's Command Bar.
- **Getting Started WithTinderbox**. Opens a PDF (stored inside the Tinderbox program) which explains Tinderbox to new users.
- **Actions and Dashboards**. Opens a PDF (stored inside the Tinderbox program) which explains use Tinderbox actions and construction of dashboards. There is an associated TBX files available here.
- **The Tinderbox Way**. A web link to a page describing a book about using Tinderbox by its chief architect and coder, Mark Bernstein.
- **Tinderbox Forum**. A web link to the online user-to-user Tinderbox Forum. Note the forum is user-to-user discussion & help and is not formal Eastgate support. Urgent/vital support issues or crash reports should always be emailed to Eastgate support.
- **Reveal Support Folder in Finder**. If clicked, a Finder window opens on the application support folder for Tinderbox (at `~/Library/Application Support/Tinderbox`), giving access to various user configuration features.

Apple Help match listing:

- Apple Help match listing

Apple Help match listing



This displays a list of Tinderbox menu matching to the typed term. When a value is typed in the search box the matches are shown beneath it, replacing existing Help menu items. Clear the search box to re-show default menu items.

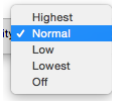
Hover the mouse over any of the listed matches and Tinderbox will display that (menu) item; a floating blue arrow points to the specific menu item.

Click a listed item and, if the menu item is active, that action is applied.

Pop-up menus and lists

Agent Priority pop-up list

Used on the Action Inspector Query tab and in the Get Info agent tab, this list controls the agent's $AgentPriority value and thus helps influence agent update cycle time. The available values are (actual $AgentPriority values are in brackets):

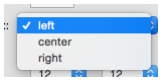


- Highest (0)
- Normal (1 default)
- Lower (4)
- Low (10)
- Off (-1)

If the selection is not an agent this list is normally disabled.

Alignment pop-up list

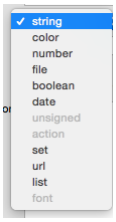
This controls the placement of the icon title and caption in map view icons ($NameAlignment & $CaptionAlignment):



- **left** (default)
- **center**
- **right**

Attribute Type pop-up list

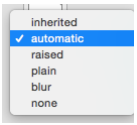
The available data types of attributes. The list may contain greyed out types; these are types for internal app use only and are not available to users. User-creatable options are:



- **string**.
- **color**.
- **number**.
- **file**.
- **boolean**.
- **date**.
- *unsigned*. For system use only
- *action*. For system use only
- **set**.
- **url**.
- **list**.

Border Style pop-up list

This controls the style used to draw a map icon border ($BorderBevel):

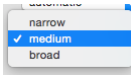


- **inherited**
- **automatic (default)**
- **blur**
- **raised**
- **plain**
- **none**

The 'inherited' option functions as 'none' in other lists, and (re-)sets the inherited document default value.

Border Width pop-up list

This sets the width of a map icon border ($Border):



- **narrow** (sets '1')
- **medium** (default, sets '2')
- **broad** (sets '4')

Cleanup Action pop-up list

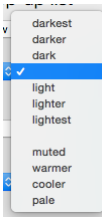
This menu is shown in the Action Inspector, Query tab, bottom right corner.
The setting (stored in $CleanupAction) controls automatic arrangement of aliases inside agents (see there for more detail).

Available values are:

- **grid** (default). Aliases are arranged in a grid layout, left-right/top-bottom in sibling order.
- **row**. Aliases are arranged in a single row, left-right in sibling order.
- **column**. Aliases are arranged in a single column, top-bottom in sibling order.
- **box**. Aliases are arranged in a an open box (square) arrangement, clock-wise in sibling order from top-left.
- **none**. Auto-layout is suspended, the user is free to arrange the items within the agent containers as they would any other map.

Color Shade pop-up list

This is used in combination with the Color pop-up menu and in the Interior/Border/Shadow tabs of the Inspector, to set shades of Tinderbox pre-defined colours. The menu is usually greyed out unless a colour has been selected in the main Color pop-up. Choices:

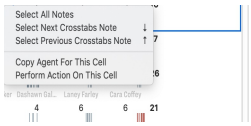


- **lightest**
- **lighter**
- **light**
- [no shade] (the default).
- **dark**
- **darker**
- **darkest**
- **muted**
- **warmer**
- **cooler**

More on note colours. When storing a shade of a named colour in a Color-type attribute, the shade is written in shade/colour order although the menu are normal drawn on screen in colour/shade order. So for 'red' and a shade of 'darker', this would be stored as a single string, 'darker red'.

Crosstabs view cell context menu

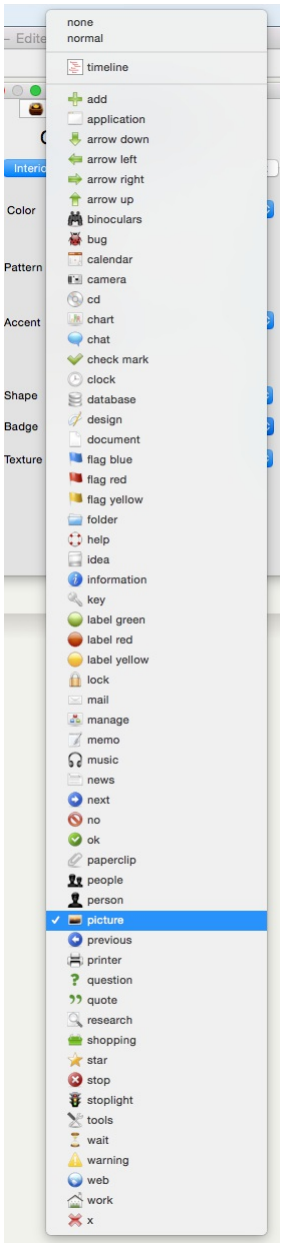
This menu is active if the context menu is activated when a cell in a crosstabs view is selected. If has the following options:



- **Select All Notes**. This selects all notes referenced within the cell. The text pane shows the normal aggregated title/text for a multiple selection.
- **Select Next Crosstabs Note**. Selects the next note in the cell into the text pane. If the current note is the last note in the cell, the first note is selected.
- **Select Previous Crosstabs Note**. Selects the previous note in the cell into the text pane. If the current note is the first note in the cell, the last note is selected.
- **Copy Agent For This Cell**. This will place on the clipboard an agent that will gather the notes in the current cell; choose a map or outline view in another tab and paste the agent where desired. The pasted agent is named "Crosstabs N,N", where the N,N are column-number and row-number from upper left of the grid, numbered from (1,1). So an agent for column 3 row 2 would be named "Crosstabs 3,2".
- **Perform Action On This Cell**. Opens an action code input box. Type action code expression(s) or the name of a Stamp. Clicking the **Return** key (↩) will apply the code to the notes in the selected cell. Use **Escape** key to cancel the code input box without applying any action

Default Badge list

Sub-menu of the Note menu. This allows a Badge to be set. The name is used as the value of the Badge attribute when an icon is selected. The full list of default badge names, as used when setting $Badge via actions and rules is below:



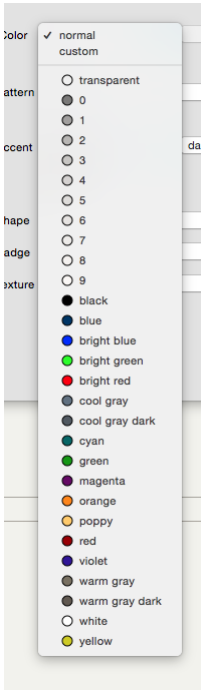
- none
- normal
- [divider rule]
- [any custom badges]
- [divider rule if any custom badges]
- add
- application
- arrow down
- arrow left
- arrow right
- arrow up
- binoculars
- bug
- calendar
- camera
- cd
- chart
- chat
- check mark
- clock
- database
- design
- document
- flag blue
- flag red
- flag yellow
- folder
- help
- idea
- information
- key
- lock
- mail
- manage
- memo
- music
- news
- next
- no
- ok
- paperclip
- people
- person
- picture
- previous
- printer
- question
- quote
- research
- shopping
- star
- stop
- stoplight
- tools
- wait
- warning
- web
- work
- x

The selected choice is applied to *all* selected notes, if more than one.

The menu will also list any custom badge artwork.

Defined Colors pop-up list

This is used for setting $Color, $AccentColor and other Color-type system attributes for selected note(s) to use Tinderbox defined colours:



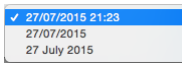
- **normal** (default)
- **custom** (replaced by a hex value, e.g. "#ddb359", if a custom colour is set.)
- **automatic** (not used in all menus.)
- **transparent**
- **greyscale series 0–9** (a series of 10 colours)
- **black**
- **blue**
- **bright blue**
- **bright green**
- **bright red**
- **cool gray**
- **cool gray dark**
- **cyan**
- **green**
- **magenta**
- **orange**
- **poppy**
- **red**
- **warm gray**
- **warm gray dark**
- **yellow**
- **white**

This is based on the main Colors menu, with the addition of the custom option.

'Custom'. If no custom colour is currently set and 'custom' is clicked, the Custom Colour colour picker dialog is opened. If a custom colour is already set, to set a new custom colour, either click the colour chip on the host dialog or select and set a defined colour then re-select 'custom'.

The colours available here may be further modified by selecting a shade from the Colour Shade pop-up menu (if the latter is co-located). The ordering of built-in/custom colours is alphabetical.

Displayed Attributes date format pop-up list

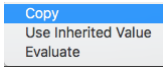


A pop-up list is used in the Text tab of Document Settings to set the default format used for Date-type attributes in note Displayed Attributes. There are 3 values.

- Locale 'long' date with time.
- Locale 'long' date only.
- Locale 'short' date only.

The values shown are for a UK locale. The strings actually seen will vary according to the user's locale.

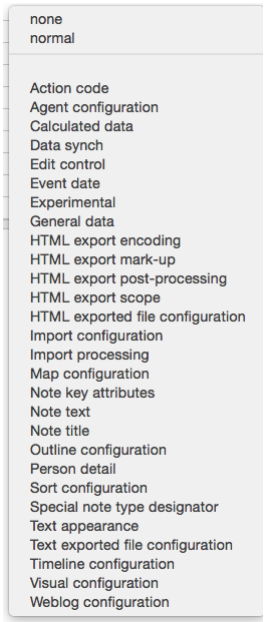
Displayed Attributes table context pop-up



For notes showing a Displayed Attributes table, there is a pop-up context menu:

- **Copy**. Copies the value of the selected table attribute, including read-only values.
- **Use Inherited Value**. Resets the attribute value to re-enabl e the inherited value.
- **Evaluate**. Evaluates the current attribute (action code) value and replaces it with the evaluation result; this presumes the existing value represents an action code expression.

Displayed Attributes, Value pop-up list



This menu for String, Number, List and Set displayed attributes allows you to pick any already-used string [sic] attribute value from a list when that attribute is displayed as a Displayed Attribute in a text window. The list is created dynamically and is only shown for String, Set and List data type attributes (unless they are (system) read-only fields).

This list is seen in the text pane Displayed Attributes table, Get Info attributes tab attribute tables (which work like a Displayed Attribute table), and in the Quickstamp Inspector.

What is in the list?

For a the selected attributes, the list shows all the values in use in the document, i.e. a de-duplicated list of all the unique (case-sensitive) values used for the attribute across the whole document. This means used values of "ant", "Ant", and "ANT" will all have discrete entries in the list.

For List and Set type attributes, the list shows all discrete values not just the lists of values. Thus if a note has a value "ant;dog" the value list will show "ant" and 'Dog" as separate list items.

For all types, the sort order is case-sensitive lexical sort (locale dependent). The list is sorted so as to assist rapidly finding the correct item in the list.

Note that in the case of List-type only the pop-up is list is sorted and this does not affect actual attribute values which retain their es-entered sort order and allow duplicate values

Selecting from the list

For a String, the clicked value replaces the existing value. For Sets and Lists only, clicking a listed value *toggles* it. The value is added unless the attribute note's already has that value, in which case it is deleted.

The 'normal' value listed separately at the top of the menu resets the default value for that attribute (or set).

Ticks in lists for Set-type attributes

Set-type attributes, and only Set-type, show a tick against all values in the list that occur in that attribute's value for the current note.

Using the list to find wrong-cased or misspelt values

As attribute values are stored case-sensitively, use an agent to help hunt down notes with incorrect spelling/case for values. Use the == equality operator for the query as this is inherently case-sensitive. Using .contains() you need to explicitly set case sensitivity although it is 'on' by default, or use .icontains().

Length of values in the list

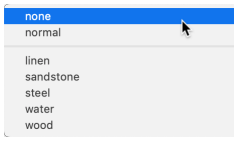
Individual listed values are truncated at 31 characters. Values longer than that must be entered manually. Value listings can be as long as 255 characters ($Text is an exception)

Size of the list

There is a limit to the number of values for which a list will be created; the current default is 999. Where this value is exceeded, the pop-up shows the first 999 values (assumed to computed from reading notes in $OutlineOrder).

In the rare context, such as very large projects, constrained by the default list size limit, it is possible to set a higher limit by using a custom config.xml with a new value. If doing this and setting very high limits do watch for adverse performance affects and be prepared to reduce the custom limit if so.

Fills pop-up menu



This menu allows the setting of image based fills in maps for notes ($Fill) in the Appearance/Interior Inspector or the view's background ($MapBackgroundFill) in the Map Settings pop-over:

- none (default). When this is set the closed menu shows no value. This option also (re-)sets the inherited default value if applied to an item with an existing value.
- normal. This (re-)sets an inherited value - the same as 'none' if there is no other inheritance, e.g. via a prototype.

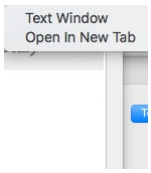
[rule]

- [if any] Custom fills in alphabetical order

[rule - if custom fills]

- plain (default)
- linen
- sandstone
- steel
- water
- wood

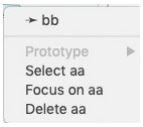
Find Results context menu



Find Results offers a contextual menu:

- **Text Window**. Open the item under the click point as a stand-alone text window
- **Open in New Tab**. Open the item under the click point in a new tab (using the existing view pane type)

Hyperbolic view context menu.

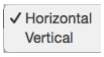


The Hyperbolic view shows a context menu when a view item is right-clicked.

The menu shows:

- **Outbound link(s)**. Each in-/out-bound link is listed, if any. The listing included an arrow indicating the in/out direction of the link. Clicking on a link selects the linked note. If links are listed they are separated from other menu items by a ruler.
- **Prototype**. This shows a sub-menu of available prototypes, allowing this note's prototype to be set or changed.
- **Select [note]**. Selects the note from which the context menu was opened.
- **Focus on [note]**. Selects the note from which the context menu was opened *and* set it as the view focus note.
- **Delete [note]**. Deletes the note from which the context menu was opened.

Layout Orientation sub-menu

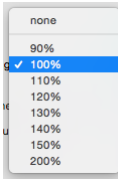


This sub-menu is called from the Text pane context menu and controls the orientation of text within the text pane.

Line Spacing pop-up list

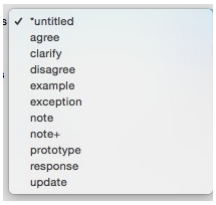
This controls the spacing between lines in the $Text pane of a note text window ($LineSpacing). The spacing is set in percent terms. The default is 100(5), smaller values give tighter spacing, larger values give wider spacing, such as allowing the printing of line-spaced text for paper-based proof-reading. The presets are:

- **90**
- **100** (default)
- **110**
- **120**
- **130**
- **140**
- **150**
- **200** (equates to line-spaced text)



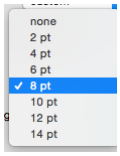
Link Types pop-up list

This pop-up shows a list of available link types in the current file. The exact listing is controlled via the Link Types dialog.



Paragraph Spacing pop-up list

This controls the spacing between paragraphs in the $Text pane of a note text window ($ParagraphSpacing). The default value, set via the text tab of Document Settings, is 8pt. The presets are:

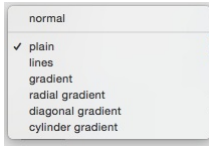


- **none**
- **2 pt**
- **4 pt**
- **6 pt**
- **8 pt** (default)
- **10 pt**
- **12 pt**
- **14 pt**

Setting a value of none is pertinent for notes like templates which hold HTML or other mark-up source code.

Pattern pop-up list

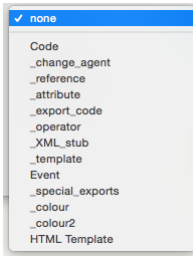
This allows the setting of non-dynamic patterns (in $Pattern):



- **normal**. This (re-)sets the inherited default value.
- **plain** (default)
- **lines**
- **gradient**
- **radial gradient**
- **diagonal gradient**
- **cylinder gradient**

Prototype pop-up list

Do any of the following to set the $Prototype of the current selected item(s):



- Outline/Chart view. Right click icon of selected item(s).
- Map view. Right click prototype tab of selected item(s).
- Properties Inspector, Prototype tab.

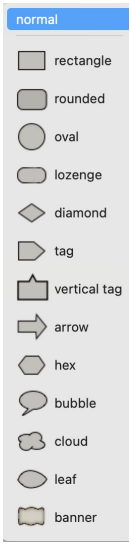
It will pop up a menu of prototypes available in the current TBX. Selecting a list item will assign that prototype to the current note.
If the current TBX has no prototypes the list simply contains the item 'none'.
The selected choice is applied to all selected notes, if more than one.
This is just one method of setting a note's prototype.

List sort order

The items in the list are sorted in order of $OutlineOrder value. The list includes both prototypes in the default '/Prototypes' container and any created elsewhere in the document.
To re-order the list simply arrange the prototypes' $OutlineOrder accordingly.
If all prototypes are in the /Prototypes folder, you can set the Sort order for that container to automatically sort the list to your desired list order.

Shape pop-up list

This sub-menu of the main view note pop-up menu, allows a note to have a non-regular (rectangular) shape. The chosen shape is stored in the $Shape attribute. The menu uses pictorial rather than text values (as in the screen grab), equating to these attribute values:

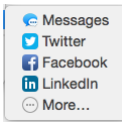


- **normal**
- **rectangle**
- **rounded** (v9.5.0)
- **oval**
- **lozenge**
- **diamond**
- **tag**
- **vertical tag**
- **arrow**
- **hex**
- **bubble**
- **cloud**
- **leaf**
- **banner**

The attribute default of [no value] or 'normal' equates to 'rectangle' in terms of the shape displayed in Map view.

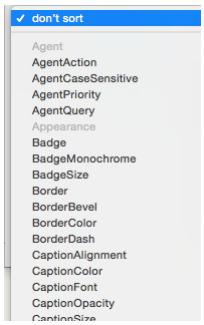
Share sub-menu

This sub-menu is called from the Text pane context menu and allows sharing of content via various forms of social media.



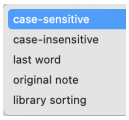
Sort pop-up list

This menu is used for the 'sort by' and 'also by' pop-ups on the Action Inspector's Sort sub-tab for Notes, Agents and Adornments. The first sort value is stored in $Sort and the second in $SortAlso.
User attributes are listed first, then the various system groups. The Internal and HTML system attributes are omitted for brevity as they are unlikely to be used for sorting. However, attributes from the latter sets can still be set via a stamp or the Info view.



Sort Transform pop-up list

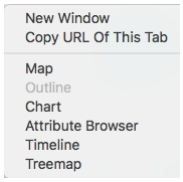
Containers and agents may apply a transform before sorting string attributes. The transform is selected from a pop-up menu in the container or the Action Inspector's Sort sub-tab.
The resulting setting is stored in the attributes $SortTransform and $SortAlsoTransform which may also be set directly to control the sort transform.
The available transforms (further described here) include:



- **case-sensitive**.
- **case-insensitive**.
- **last word**. This is useful to sort note titles that are names, i.e. 'John Doe' sorts as for 'Doe' not 'John...'.
- **original note**. Sorts as per the $Outline order of the original of the alias.
- **library sorting**. From v9.6.0, library sorting ignores initial words "a", "an", and "the". The defaults can be configured via the document's Hints section— see more.

Tab pop-up menu

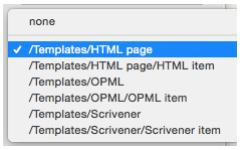
This menu is shown when right-clicking a Tab:



- **New Window**. Opens a new document window using the current context for its default two tabs.
- **Copy URL Of This Tab**. Saves the Tinderbox pseudo-protocol URL of this tab. See more on the Tinderbox protocol.
- **Gallery**. Opens the saved tabs Gallery pane.
- (ruler)
- **Main view types**. The currently selected view pane view type is greyed out:
- **Map**. Switch this tab's view pane view to Map view.
- **Outline**. Switch this tab's view pane view to Outline view.
- **Chart**. Switch this tab's view pane view to Chart view.
- **Attribute Browser**. Switch this tab's view pane view to Attribute Browser view.
- **Timeline**. Switch this tab's view pane view to Timeline view.
- **Treemap**. Switch this tab's view pane view to Treemap view.

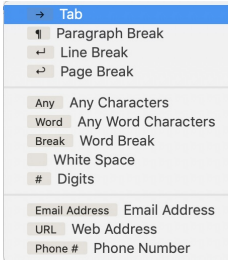
Template pop-up menu

This list shows any note templates defined for the current document, *except* any templates that are also prototypes.
This menu is used by the HTML Inspector's Export sub-tab. It is also used in the HTML tab of the text pane of a note, though only if the document currently has no export templates present and set for export use.



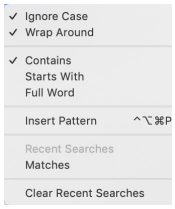
Text pane, Find input Pattern pop-up

Opened from the Text pane's Find input pop-up menu, this allows a number of pre-defined and previously-used regular expression patterns to be added to the search input.
These tokens cannot be further edited, either to add to the list or once inserted into the search string.



Text pane, Find input pop-up

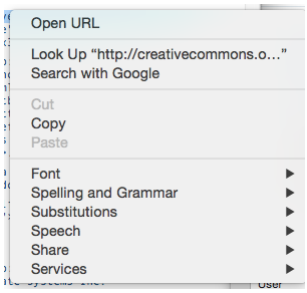
This menu is called via the chevron at the left of the search term input box in the Text pane's find toolbar. It lists the following options:



- **Ignore Case** (default: ticked). Case-insensitive if ticked.
- **Wrap Around** (default: ticked). If ticked the search wraps to the beginning of $Text if the end of the text is reached.
- **Contains/Starts With/Full Word** (default: Contains). 3-way toggle, tick indicated active choice. The selection indicates the type of search undertaken.
- **Insert Pattern**. Opens a further Pattern selection menu for regular expressions.
- **Recent Searches**. A list of recently used search terms (beneath a grey-ed header).
- **Clear Recent Searches**. Clears the above list.

Text pane, HTML tab, pop-up menu

This is the menu seen when right-clicking in the main window text pane's HTML tab:

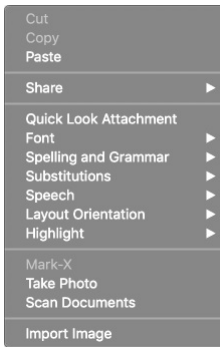


- **Open URL**. Only shown if cursor is in a URL. Open that URL in a web browser.
- **Look Up "[selection]"**. Only shown if cursor is over text. Look up the current selection in Dictionary.
- **Search with Google**. Only shown if cursor is over text. Look up the current selection online in Google.
- **Cut**. Cut the current selection to the clipboard.
- **Copy**. Copy the current selection to the clipboard.
- **Paste**. Paste the clipboard contents at the current insertion point.
- **Font**. Opens the Font sub-menu.
- **Spelling and Grammar**. Opens the Spelling and Grammar sub-menu.
- **Substitutions**. Opens the Substitutions sub-menu.
- **Transformations**. Opens the Transformations sub-menu.
- **Speech**. Opens the Speech sub-menu.

Text pane, no text selection, pop-up menu

This is the menu seen when right-clicking in the main window text pane when there is a current selection of $Text:

- **Cut**. Cut the current selection to the clipboard.
- **Copy**. Copy the current selection to the clipboard.
- **Paste**. Paste the clipboard contents at the current insertion point.
- **Find**. Opens the Find sub-menu.
- **Spelling and Grammar**. Opens the Spelling and Grammar sub-menu.
- **Substitutions**. Opens the Substitutions sub-menu.
- **Transformations**. Opens the Transformations sub-menu.
- **Speech**. Opens the Speech sub-menu.
- **Layout Orientation**. Opens the Layout Orientation sub-menu for text orientation.
- **Highlight**. Opens the text Highlight sub-menu.
- **Share**. Opens the Share sub-menu for sharing via social media.
- **Import Image**. Opens a dialog to aid importing an image from attached media devices such as a phone or tablet.
- **Capture Selection from Screen** . Opens the Mac Grab app to take a screenshot.



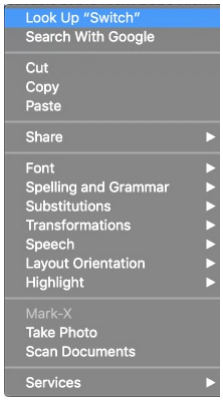
Text pane, Preview tab, pop-up menu

This is the menu seen when right-clicking in the main window text pane's Preview tab:

- **Reload**. Reload the preview.
- **Print**. Print the current preview.

Text pane, text selected, pop-up menu

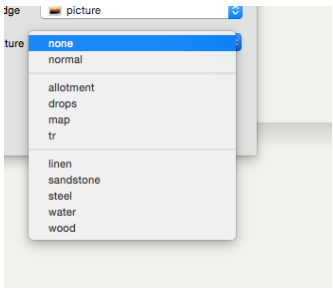
This is the menu seen when right-clicking in the main window text pane when there is a current selection of $Text:



- **Look Up** "[selected-text]". Look up the selected text in the Dictionary app.
- **Search with Google**. Open a search got the current text selection via Google in the user's default web browser.
- **Cut**. Cut the current selection to the clipboard/
- **Copy**. Copy the current selection to the clipboard.
- **Paste**. Paste the clipboard contents at the current insertion point.
- **Find**. Opens the Find sub-menu.
- **Spelling and Grammar**. Opens the Spelling and Grammar sub-menu.
- **Substitutions**. Opens the Substitutions sub-menu.
- **Transformations**. Opens the Transformations sub-menu.
- **Speech**. Opens the Speech sub-menu.
- **Layout Orientation**. Opens the Layout Orientation sub-menu for text orientation.
- **Highlight**. Opens the text Highlight sub-menu.
- **Share**. Opens the Share sub-menu for sharing via social media.
- **Services**. Opens a sub-menu listing OS Services pertinent to the current context.

Texture pop-up list

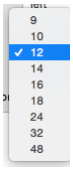
This sets the map icon fill, or texture ($Fill):



- **none**. No value (but not inherited)
- **normal**. (Re-)sets inherited value.
- [any custom fill(s)]
- **linen**
- **sandstone**
- **steel**
- **water**
- **wood**

Title size pop-up list

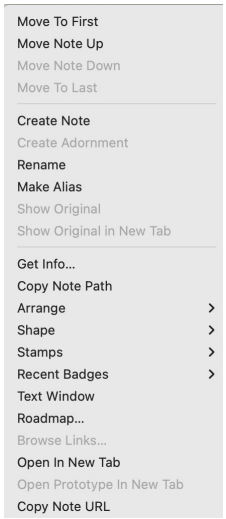
This list is used for both the Map and Outline title size pop-up list in the Text Title Inspector. The values shown are point sizes although they are stored as a percentage based on a default of 14pt being 100. The number values are the value stored in the associated attributes. Values used are:



- 9 (64)
- 10 (71)
- 12 (86)
- 14 (100) default
- 16 (114)
- 18 (129)
- 24 171)
- 32 (229)
- 48 (343)

View pane (note selected), pop-up menu

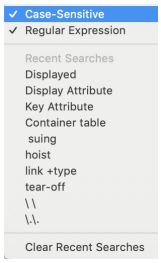
This menu is shown when right-clicking in the view pane of Map, Outline, Chart or Treemap views with at least one item selected; Attribute Browser and Treemap views have differing pop-up menus. Items may be greyed out according to context and some items vary by view type:



- **Move To First**. In Map views this moves the current note in front of all other notes (i.e. first by outline order).
- **Move Note Up**. In Outline views this moves the current note up one place (at sibling level).
- **Move Note Down**. In Outline views this moves the current note down one place (at sibling level).
- **Move To Last**. In Map views this moves the current note behind all other notes (i.e. last by outline order).
- **Rename**. Places selected item's title in Edit-in-Place mode.
- **Create Note**. Create a new note, as next sibling to the current selection.
- **Create Agent**. Create a new Link type honouring operators, as next sibling to the current selection.
- **Create Separator**. (Outline only) / **Create Adornment** (Map only). View dependent:
  - Outline view: adds a new note as next sibling with the 'separator' option pre-ticked. If more than one note is selected, the separator is placed after the first item in the selection (by outline order).
  - Map View: adds an adornment. The adornment is created so as to enclose the currently selected note. If more than one note is selected, the adornment surrounds the top left note in the selection.
  - All other views: shows 'Create Separator' option greyed out.
- **Make Alias**. Make an alias of the current selection.
- **Show Original**. Available when an alias is selected. Locates the position of the source note for the alias.
- **Show Original In New Tab**. The original is shown, but a new tab is opened and selected.
- **Get Info...** Open the Get Info pop-over for the selected item.
- **Copy Note Path**. (v9.5.2) This places the $Path value of the selected note on the clipboard (i.e. in in-app path). Note that if the selected item uses a Display Expression, the last part of the path may differ from the title seen on screen.
- **Arrange**. Open the Arrange submenu.
- **Stamps**. A sub-menu of stamps defined in the current document. Clicking an item applies the stamp to the current selection.
- **Recent Badges**. A sub-menu of that contains a list of badges that have recently been selected using the Badge Picker or the Appearance Inspector. The most recently-used badge is listed first. Badges that are set by actions, displaye attributes, or the attributes pane of Get Info are not taken into account by this menu. Using this to select a badge places the selected badge at the top of the recent badges menu; also, this menu item is applied to all selected notes.
- **Text Window**. Open the current note as a stand-alone text window.
- **Roadmap...** Open the Roadmap pop-over for the selected item.
- **Browse Links...** Opens the Browse Links pop-over.
- **Open in a New Tab**. Open the current context as a new tab. This open a new tab and shifts outline depth focus in one move.
- **Open Prototype in New Tab**. Opens the *prototype* of the selected note. Remains greyed out if the selected note has no prototype assigned.
- **Copy Note URL**. Copies a Tinderbox protocol URL to the clipboard (to re-open the document with the current view and selection).

View pane Find, pop-up menu

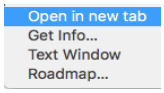
Clicking the down-chevron in the View pane's Find toolbar's input box opens this menu.
The menu includes the following options:



- **Case-sensitive** (default: unticked). Indicates whether the input search term is checked case-sensitively or not. This setting equates to the 'case sensitive' tick-box on the stand-alone Find results dialog.
- **Regular Expression** (default; ticked). Indicates whether the input terms should be parsed as if a regular expressions.This should be left in the default setting. This option is *not* repeated on the Find results dialog, unlike the setting above.
- A list or recent search terms.
- **Clear Recent Searches**. Clicking this clears the above list of past search terms.

View pane, Attribute Browser (note selected), pop-up menu

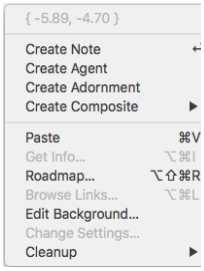
This menu is shown when right-clicking in a view of an Attribute Browser view, when a note is selected. If no note is selected no pop-up is shown. Menu items are:



- **Open in new tab**. Opens a new Map view with the parent container selected. The view can then be altered to a (hoisted) view of any other type via the tab context menu.
- **Get Info...** Opens the item's Get Info dialog as a pop-over.
- **Text Window**. Opens a stand-alone text window for the current item.
- **Roadmap...** Opens the item's Roadmap dialog as a pop-over.

View pane, pop-up menu

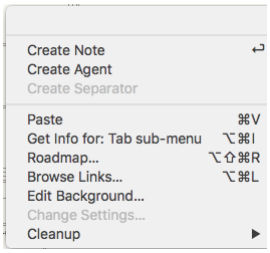
This menu is shown when right-clicking in a main view with no item selected. Items may be greyed out according to context and some items vary by view type:



- **Map co-ordinates of cursor** (Map view only). This shows the $Xpos/$Ypos map co-ordinates of the map cursor when right-clicked. In other views this entry is a blank line.
- **Create Note**. Create a new note, placement varies by view type.
- **Create Agent**. Create a new agent placement varies by view type.
- **Create Adornment**/**Create Separator**. View dependent:
  - Map view only: **Create Adornment**. Creates a new adornment, with top left corner in the context-clicked position.
  - Outline view only: **Create Separator**. Creates a new separator at the top of the current outline; for hoisted containers it is created as the first item within the root container of the view.
  - All other views. Greyed out, labelled as 'Create Separator'.
- **Create Composite**. Map view only. Greyed out in all other views. Shows a sub-menu of currently defined composites.
- **Paste**. Paste clipboard contents to the view.
- **Get Info...** Always greyed out, as there is no selection.
- **Roadmap...** Opens Roadmap pop-over.
- **Edit Background...** (Map, Outline and chart only). Opens the Edit Background pop-over.
- **Change Settings...** (Timeline only). Opens the Timeline Settings pop-over.
- **Cleanup**. Opens Clean-up sub-menu. Greyed out in views other than Map view.

View pane, Treemap (note selected): pop-up menu

This menu is shown when right-clicking in the view pane of a treemap view with an item selected. Items may be greyed out according to context and some items vary by view type:



- **Create Note**. Create a new note, placement varies by view type.
- **Create Agent**. Create a new agent placement varies by view type.
- **Create Adornment**. Greyed out in views other than Map view. Create a new adornment. For maps the item is created in the click position.
- **Paste**. Paste clipboard contents to the view.
- **Get Info...** Always greyed out, as there is no selection.
- **Roadmap...** Opens Roadmap pop-over.
- **Edit Background...** Opens the Edit Background pop-over.
- **Change Settings...** Greyed out (opens the Timeline Settings pop-over).
- **Cleanup**. Opens Clean-up sub-menu. Greyed out in views other than Map view.

Visual Styling

- Text Markups
- Colouring $Text
- Higlighting $Text
- Shapes, borders, patterns and fills
- Tinderbox built-in fonts
- Chart of Tinderbox's defined colours
- Chart of example shapes, borders and patterns
- Note Colours

Text Markups

This is **bold text**.

This is *italic text*.

This is underlined text.

This is ~~strikethrough text~~.

This is subscript text. (This effect does not export to HTML).

This is superscript text. (This effect does not export to HTML).

This is highlight text. (This does not export to HTML).

This is a bullet:

- bullet text

Tinderbox supports simple HTML unordered list generation by placing an asterisk at the start of a line. Two asterisks is a second level list item, etc. Thus:

* This is a list item

** This is a sub-item of the first list item

*** This is a sub-item of the above sub-item and so on

* This is another list item

** This is its sub-item

* Another list item…

Use hash (#) symbols in the above scenario to create numbered lists:

This is a list item

This is a sub-item of the first list item

This is a sub-item of the above sub-item and so on

This is another list item

This is its sub-item

Another list item…

Colouring $Text

A series of four Color-type attributes $TextColorRed, $TextColorBlue. $TextColorGreen, and $TextColorGray let you control the colour applied via Format ▸ Style ▸ Red and related commands. Changing these attributes does not change text colours previously applied, but affects future applications of these styles.

Higlighting $Text

A series of five Color-type attributes $TextHighlightRed, $TextHighlightBlue, $TextHighlightGreen, $TextHighlightMagenta, and $TextHighlightYellow let you control the colour applied by Format ▸ Style ▸ Highlight. Changing these attributes does not change text highlights previously applied, but affects future applications of these styles.

Shapes, borders, patterns and fills

Look at the content of this note in Map view for a better display of the various effects or view the map as an image. In either form, layout is a useful visual reference to the descriptions below. These are examples of a number of the ways the look and feel of note icons can be altered.

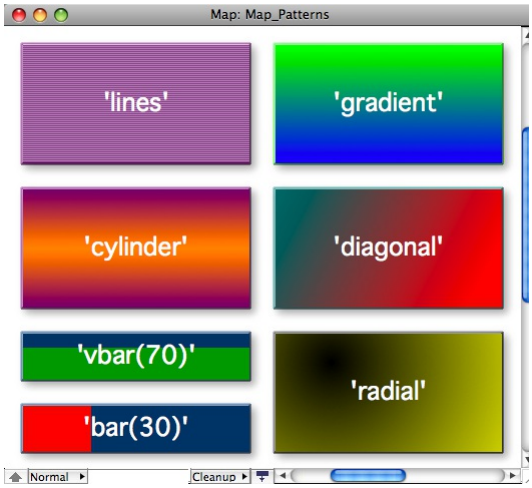
- rectangle or normal
- Pattern: lines
- Pattern: gradient
- Pattern: diagonal
- Pattern: cylinder
- Pattern: radial
- Pattern: bar(30)
- Pattern: vbar(70)
- Pattern: bargraph() (for container plot only)
- Pattern: plot() (for container plot only)
- Pattern: xyplot() (for container plot only)
- Pattern: ring() (for container plot only)
- Pattern: pie() (for container plot only)

rectangle or normal

This is default note $Shape.

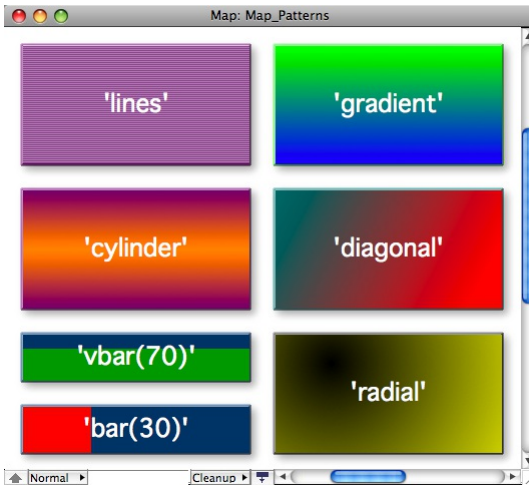
Pattern: lines

Horizontal lines in $Color/$AccentColor.



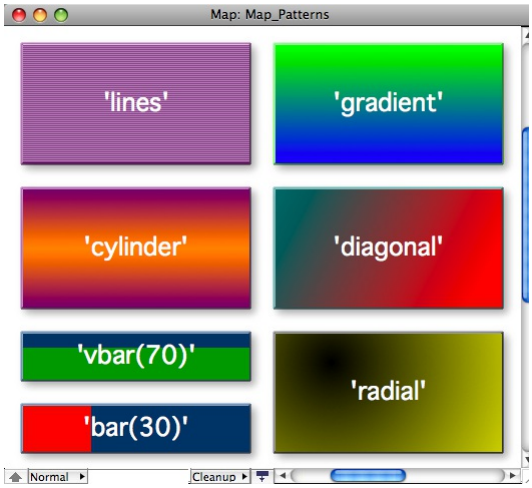
Pattern: gradient

A gradient fill from $Color (top) to $AccentColor (bottom).

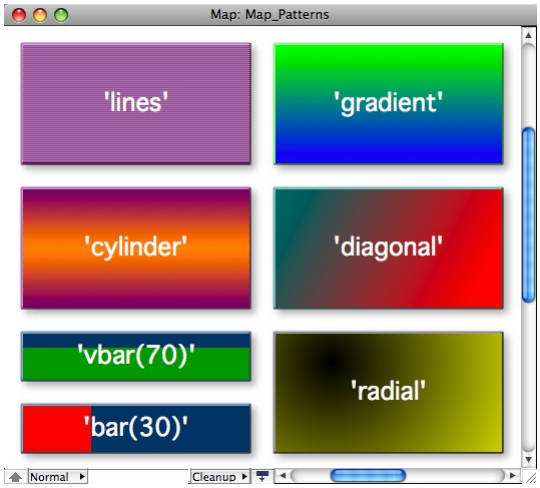


Pattern: diagonal

A graduated fill from $Color (top left) to $AccentColor (bottom right).
This is the default for $Pattern.



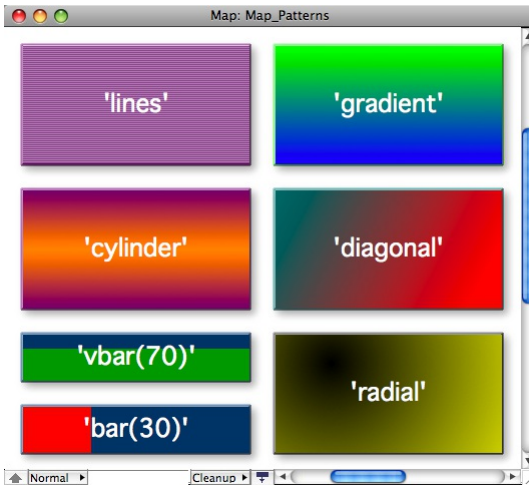
Pattern: cylinder



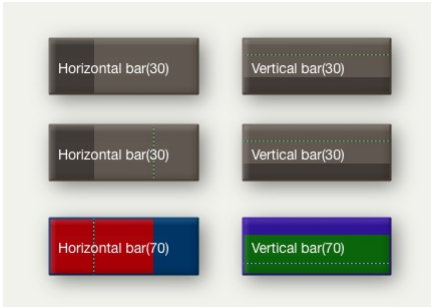
Graduation from $Color (top) through $AccentColor (middle) and back to $Color (bottom).

Pattern: radial

A graduation from $Color (1/3 from top/left) through $AccentColor (edges). Added to Inspector's $Pattern pop-up menu.



Pattern: bar(30)



Note that this pattern cannot be set via the Interior Inspector, use the Plot Inspector. The pattern requires configuration via its arguments. This pattern can be used in shaped notes.

bar(value,[min],[max][,target])

This draws as a horizontal 'progress bar', using $Color and $AccentColor. The 'progress' block is drawn in $AccentColor. These colours are set via the Interior Inspector.

The arguments work this way:

- **value**. If used alone it is a percentage, otherwise it is a value between min and max with the bar being set using value as a percentage of **max-min** (which must thus be specified). Thus `bar(25)` is the same as `bar(37.5,25,75)`.
- **min** & **max**. Numerical minimum and maximum values for computing the degree of 'progress' represented in $AccentColor by value. Both arguments may be negative numbers; if **min** is not supplied, a value of 0 is assumed and negative attribute values are plotted as if zero.
- **target**. The target represents a nominal or desired result. For example, if normally writing between 0 and 4000 words on any given day, it might be useful to set a **target** at 1500 words/day. If setting a **target**, a **min** and **max** must also be set. The **target** line is drawn in a dashed link alternating $PlotColor and either $Color or $AccentColor) whichever is the current background colour in the position of the target line.

The arguments in the brackets may be numbers or expressions that can be evaluated as numbers. As $Pattern is a string argument, remember to enclose the statement in quotes so the program coerces the output to a string. If a note's $Width is 3.0, then

```
$Pattern="bar(15*$Width)"
```

is the same as writing

```
$Pattern="bar(45)"
```

Note that if the value to be use is an action code variable, a slightly different encoding is needed to ensure the value and not the name of the variable is used. Here, our progress percentage is in a variable 'vPercent':

```
$Pattern="bar("+vPercent+")"
```

or more accurately:

```
$Pattern="'bar('+vPercent+')'"
```

… though both forms seem to work - use which ever feels more intuitive.

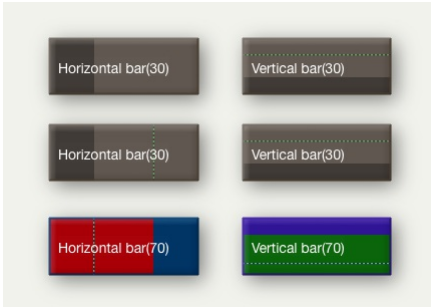
If the pattern is written without arguments or brackets, it evaluates as if at 50%, e.g. `bar()` and `bar` are the same as `bar(50)`.

The same pattern can be rendered on a vertical access using the vbar() pattern.

Use in Note flags

From v9.5.2, this pattern expression can also be using in $Flags to make progress bar note flags in Map and Outline views.

Pattern: vbar(70)



Note that this pattern cannot be set via the Interior Inspector, use the Plot Inspector. The pattern requires configuration via its arguments. This pattern can be used in shaped notes.

vbar(value,[min],[max])

This draws as a vertical 'progress bar', using $Color and $AccentColor. The 'progress' block is drawn in $AccentColor. These colours are set via the Interior Inspector.

The arguments work this:

- **value**. If used alone it is a percentage, otherwise it is a value between min and max with the vbar being set using value as a percentage of **max-min** (which must thus be specified). Thus `vbar(25)` is the same as `bar(37.5,25,75)`.
- **min** & **max**. Numerical minimum and maximum values for computing the degree of 'progress' represented in $AccentColor by value. Both arguments may be negative numbers; if **min** is not supplied, a value of 0 is assumed and negative attribute values are plotted as if zero.
- **target**. The target represents a nominal or desired result. For example, if normally writing between 0 and 4000 words on any given day, it might be useful to set a **target** at 1500 words/day. If setting a **target**, a **min** and **max** must also be set. The **target** line is drawn in a dashed link alternating $PlotColor and either $Color or $AccentColor) whichever is the current background colour in the position of the target line.

The arguments in the brackets may be numbers or expressions that can be evaluated as numbers. As $Pattern is a string argument, remember to enclose the statement in quotes so the program coerces the output to a string. If a note's $Width is 3.0, then

```
$Pattern="vbar(15*$Width)"
```

is the same as writing

```
$Pattern="vbar(45)"
```

Note that if the value to be use is an action code variable, a slightly different encoding is needed to ensure the value and not the name of the variable is used. Here, our progress percentage is in a variable 'vPercent':

```
$Pattern="vbar("+vPercent+")"
```

or more accurately:

```
$Pattern="'vbar('+vPercent+')'"
```

… though both forms seem to work - use which ever feels more intuitive.

If the pattern is written without arguments or brackets, it evaluates as if at 50%, e.g. `vbar()` and `vbar` are the same as `vbar(50)`.

The same pattern can be rendered on a horizontal access using the bar() pattern.

Use in Note flags

From v9.5.2, this pattern expression can also be using in $Flags to make progress bar note flags in Map and Outline views.

Pattern: bargraph() (for container plot only)

Note that this pattern cannot be set via the Interior Inspector, use the Plot Inspector. The pattern requires configuration via its arguments. This pattern can be used in shaped notes. This pattern is only available for use in container plots and has no visual effect if applied to other objects.

The graph is drawn in the form of a bar graph plot across the viewport area of the container. The graph is drawn in colour $PlotColor. The container viewport is still accessible for drag/drop, etc., as if the plot were not there. Think of the plot an overlay.

For example, to graph the word count of each child note in the container in bar graph form, set the container's map Pattern attribute to:

```
bargraph($WordCount)
```

The Y-axis of the graph will run from the minimum value of the specified attribute, for the notes being plotted, to the largest value.

The pattern accepts optional minimum and maximum values:

bargraph($ValuesSource,min,max)

Thus:

```
bargraph($WordCount,0)
```

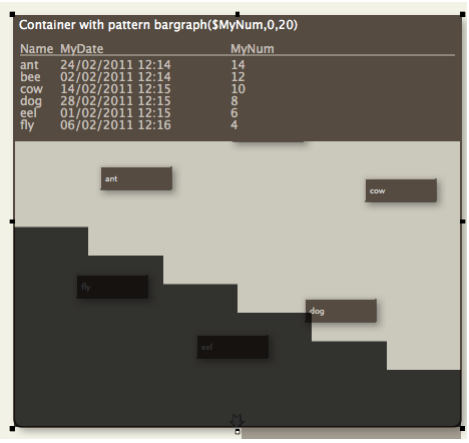
graphs the word count of each note whilst ensuring the Y-axis is based at zero, with all attribute values including the maximum being plotted. Note that while zero is the default value of an 'blank' number type attribute, the type allows minus values. The above example would treat all of them as if their value were zero. If a negative **min** value is supplied, negative item values above that threshold are plotted. For:

```
bargraph($WordCount,10,900)
```

the plot graphs the data from a baseline of 10 to a maximum value of 900. Values outside these are plotted appropriately as the **min** or **max** values.

An alternate container plot type is plot() which draws a sparkline-type plot of each child item's value or xyplot() which draws a similar sparkline-type plot but where data can be specified for both axes.

The background of the plot is set via $PlotBackgroundColor with opacity of $PlotBackgroundOpacity.



Pattern: plot() (for container plot only)

Note that this pattern cannot be set via the Interior Inspector, use the Plot Inspector. The pattern requires configuration via its arguments. This pattern can be used in shaped notes. This pattern is only available for use in container plots and has no visual effect if applied to other objects.

The graph is drawn in the form of a sparkline-type plot across the viewport area of the container. The graph is drawn in colour $PlotColor. The container viewport is still accessible for drag/drop, etc., as if the plot were not there. Think of the plot as an overlay.

For example, to graph the word count of each child note in the container, see the container's map $Pattern attribute to:

```
plot($WordCount)
```

The Y-axis of the graph will run from the minimum value of the specified attribute, for the notes being plotted, to the largest value.

The pattern accepts optional minimum and maximum values:

plot($ValuesSource,min,max)

Thus:

```
plot($WordCount,0)
```

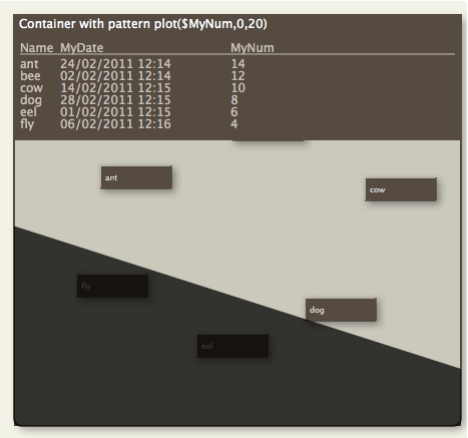
graphs the word count of each note whilst ensuring the Y-axis is based at zero, with all attribute values including the maximum being plotted. Note that while zero is the default value of an 'blank' number type attribute, the type allows minus values. The above example would treat all of them as if their value were zero. The above example would treat all of them as if their value were zero. If a negative **min** value is supplied, negative item values above that threshold are plotted. For:

```
plot($WordCount,10,900)
```

the plot graphs the data from a baseline of 10 to a maximum value of 900. Values outside these are plotted appropriately as the **min** or **max** values.

An alternate plot type is bargraph(), which draws a bar graph of each child item's value or xyplot() which draws a sparkline-type plot but where data can be specified for both axes.

The background of the plot is set via $PlotBackgroundColor with opacity of $PlotBackgroundOpacity.



Pattern: xyplot() (for container plot only)

Note that this pattern cannot be set via the Interior Inspector, use the Plot Inspector. The pattern requires configuration via its arguments. This pattern can be used in shaped notes. This pattern is only available for use in container plots and has no visual effect if applied to other objects.

xyplot($XValuesSource, $YValuesSource [,yMin][,yMax[,Target])

The graph is drawn in the form of a Cartesian graph of the container's immediate children. **$XValuesSource** and **$YValuesSource** are expressions, evaluated in turn of each child and are normally attribute values in the simplest forma of use. Thus:

```
xyplot($Date,$Price)
```

will plot $Date (ascending Date type order) on the X-axis of the container plot and $Price (ascending Number data type) on the Y-axis.

By comparison with plot():

```
plot($Price)
```

is equivalent to

```
xyplot($SiblingOrder,$Price)
```

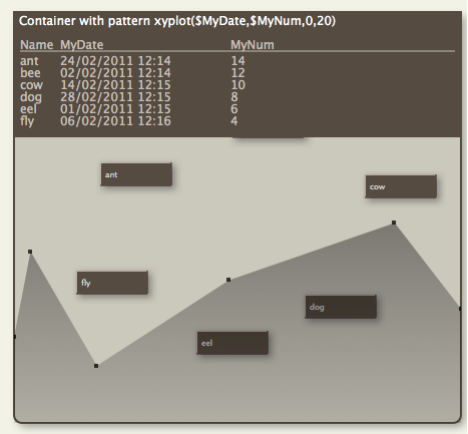
Thus if using an non-outline order related attribute for xValuesSource, consider either showing **$XValuesSource** and **$YValuesSource** as a $TableExpression or sorting the plot's container on **$XValuesSource**.

Optionally, a **yMin** and/or a **yMax** may be specified to lessen expansion of the Y-axis; otherwise, the Y-axis plots from the lowest value of **$YValuesSource** for the container's child notes, to the highest value. The X-axis automatically starts at the lowest **$XValueSource** and runs to the highest **$XValueSource**.

The **target** represents a nominal or desired result. For example, if normally writing between 0 and 4000 words on any given day, it might be useful to set a target at 1500 words/day. If setting a **target**, a **min** and **max** must also be set.

Alternate container plots are: bargraph() and plot().

The background of the plot is set via $PlotBackgroundColor with opacity of $PlotBackgroundOpacity.



Pattern: ring() (for container plot only)

Note that this pattern cannot be set via the Interior Inspector, use the Plot Inspector. The pattern requires configuration via its arguments. This pattern can be used in shaped notes.

ring(value[,min,max,target])

In the simplest usage

```
ring(70)
```

displays an arc representing 70% of a complete circle. The circle is always drawn centred on a 9-o'clock position; a 50% completion would fill from 6 through 9 to 12.

Additional optional arguments allow specification of a minimum value, a maximum value, and a target value.

All four inputs may be a literal number, an attribute's value or the result of a simple action code expression.

value is a number (between **min** and **max**). Progress, as shown by **value**, is plotted in $PlotColor.

min is the minimum **value** number (default: 0).

max is the maximum **value** number (default: 100).

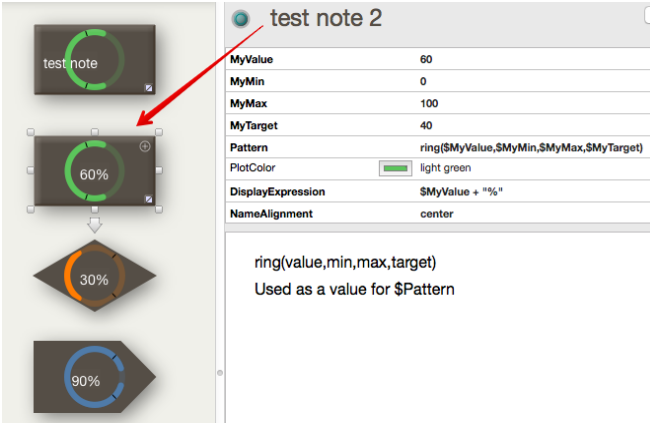
target (no default) is a desired target value (for **value**) between **min** and **max** and which may be smaller or greater than **value**. If a target is desired, min and max must also be specified. Target is plotted as two narrow black lines, centred as per the progress bar. **target** is always drawn in $PlotBackgroundColor.

Example with all inputs

```
ring(70, 0, 100, 50)
```

where the progress is 70%, exceeding the 50% target. The same could be computed:

```
ring($MyValueNumber, $MyMinNumber, $MyMaxNumber, $MyTargetNumber)
```



Pattern: pie() (for container plot only)

Note that this pattern cannot be set via the Interior Inspector, use the Plot Inspector. The pattern requires configuration via its arguments. This pattern can be used in shaped notes.

The pie() plot can be used in normal map note icons including shaped notes. Originally, this pattern was only available for use in container plots.

pie(ValuesSourceAttribute)

The graph is drawn in the form of a pie graph of the container's immediate children. ValueSource can be an attribute, or an expression yielding a number, reflecting the container's children. Thus:

```
pie($EditsMade)
```

will plot $EditsMade for each child in $OutlineOrder plotting anti-clockwise from the 90-degree position.

If using a non-outline order related attribute for **ValuesSourceAttribute**, consider either showing it as a $TableExpression or sorting the plot's container on **ValuesSourceAttribute**.

pie(ValuesSourceAttribute, Min, Max)

For non-container use of the pattern, pie() uses the additional arguments, **Min** and **Max**. Min defaults to 0 and Max to 100. If only one range limit is provided the other is used at default value. Thus:

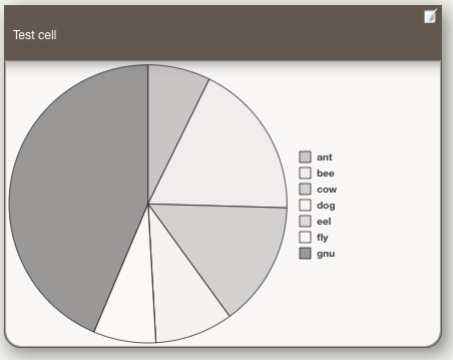
```
pie(65,0,100)
```

plots a 65% competed circle. The pie plots as a circular progress bar, anti-clockwise from the 90-degree position.

Other aspects of the pattern's display:

- Labels are drawn atop pie chart segments (avoiding other other segments overlying each others labels).
- The colours for each segment of the pie chart are taken from the attribute $PlotColorList (default list value: [2;7;3;8;4;9;0;5;1;6], using system numbered colours). The first colour designates the colour of the first segment, and remaining colours in the list are used in rotation until all segments have been drawn. If the list contains fewer than two colours, "black;white" are used inside.
- If the container is sufficiently wide, a legend is drawn to the right of the pie chart. Otherwise, pie segments are labeled in the pie chart (where the slice is big enough to fit a label) smaller slices are left unlabelled.
- If $Direction is *false* (the default), the first segment begins at the top of the pie (0º) and subsequent segments are added clockwise, as is customary in geography. If $Direction is set to `true`, the first segment begins at the right edge of the circle (90º) and segments are added counter-clockwise, as customary in mathematics.
- Pie segments are separated by a dark grey line.
- If a container or agent has a pie chart, then the alias of that container or agent will also display the same pie chart, if space allows.

The pie() plot does not use the 'target' value of other container plots.



Tinderbox built-in fonts

Tinderbox includes a number of built-in fonts, i.e. not provided by the under-lying macOS but available in all Tinderbox documents. These fonts are as below (some are used system attribute defaults):

- Archer
- Decimal
- Ideal Sans (default $NameFont)
- Mercury (default $TextFont)
- Ringside Condensed
- Sketchnote Square (default $CaptionFont)
- Tungsten

Archer (Archer SSm, Book, 16 pt.)
A quick brown fox jumped over the lazy dog
1234567890!?[]{}()&%$£

Decimal (Decimal, Book, 16 pt.)
A quick brown fox jumped over the lazy dog
1234567890!?[]{}()&%$£

Ideal Sans (Ideal Sans SSm, Book, 16 pt.)
A quick brown fox jumped over the lazy dog
1234567890!?[]{}()&%$£

Mercury (Mercury SSm, Book, 16 pt.)
A quick brown fox jumped over the lazy dog
1234567890!?[]{}()&%$£

Ringside Condensed (Ringside Condensed SSm, Book, 16 pt.)
A quick brown fox jumped over the lazy dog
1234567890!?[]{}()&%$£

Sketchnote Square Sketchnote Square, Regular, 16 pt.)
A quick brown fox jumped over the lazy dog
1234567890!?[]{}()&%$£

Tungsten (Tungsten, Medium, 16 pt.)
A quick brown fox jumped over the lazy dog
1234567890!?[]{}()&%$£

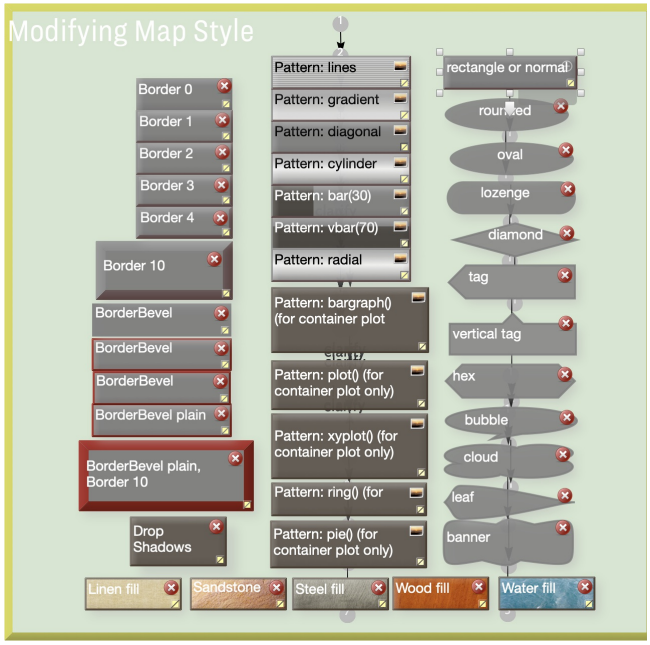
Chart of Tinderbox's defined colours

Tinderbox Named Colours

darkest black #000000	darker black #000000	dark black #000000	black #000000	light black #808080	lighter black #aaaaaa	lightest black #bfbfbf	muted black #000000	warmer black #000000
darkest blue #000e1d	darker blue #001428	dark blue #002244	blue #003366	light blue #5986b3	lighter blue #88aacc	lightest blue #a3bed9	muted blue #4c5966	warmer blue #004466
darkest bright blue #000049	darker bright blue #000066	dark bright blue #0000aa	bright blue #0000ff	light bright blue #8080ff	lighter bright blue #aaaaff	lightest bright blue #bfbfff	muted bright blue #bfbfff	warmer bright blue #002aff
darkest bright green #004900	darker bright green #006600	dark bright green #00aa00	bright green #00ff00	light bright green #80ff80	lighter bright green #aaffaa	lightest bright green #bfffbf	muted bright green #bfffbf	warmer bright green #2aff00
darkest bright red #490000	darker bright red #660000	dark bright red #aa0000	bright red #ff0000	light bright red #ff8080	lighter bright red #ffaaaa	lighter bright red #ffbfbf	muted bright red #ffbfbf	warmer bright red #ff002a
darkest cool gray #1b2024	darker cool gray #262c33	dark cool gray #404a55	cool gray #607080	light cool gray #a8b4c0	lighter cool gray #c3ccd5	lightest cool gray #d2d9e0	muted cool gray #787c80	warmer cool gray #607580
darkest cool gray dark #16191b	darker cool gray dark #202326	dark cool gray dark #353a40	cool gray dark #505860	light cool gray dark #a1a8b0	lighter cool gray dark #bfc5ca	lightest cool gray dark #cfd3d8	muted cool gray dark #5c5e60	warmer cool gray d #505b60
darkest cyan #001d1d	darker cyan #002828	dark cyan #004444	cyan #006666	light cyan #59b3b3	lighter cyan #88cccc	lightest cyan #a3d9d9	muted cyan #4c6666	warmer cyan #006655
darkest gray #474747	darker gray #636464	dark gray #a5a7a7	gray #f8fafa	light gray #fcfdfd	lighter gray #fdfefe	lightest gray #fefefe	muted gray #fafafa	warmer gray #f8fafa
darkest green #002b00	darker green #003d00	dark green #006600	green #009900	light green #66cc66	lighter green #93dd93	lightest green #ace6ac	muted green #739973	warmer green #199900
darkest magenta #1d001d	darker magenta #280028	dark magenta #440044	magenta #660066	light magenta #b359b3	lighter magenta #cc88cc	lightest magenta #d9a3d9	muted magenta #664c66	warmer magenta #550066
darkest orange #492700	darker orange #663600	dark orange #aa5b00	orange #ff8800	light orange #ffc480	lighter orange #ffd8aa	lightest orange #ffe2bf	muted orange #ffe2bf	warmer orange #ff5d00
darkest poppy #493a1d	darker poppy #665128	dark poppy #aa8844	poppy #ffcc66	light poppy #ffe6b3	lighter poppy #ffeecc	lightest poppy #fff3d9	muted poppy #fff3d9	warmer poppy #ffb366
darkest red #2b0000	darker red #3d0000	dark red #660000	red #990000	light red #cc6666	lighter red #dd9393	lightest red #e6acac	muted red #997373	warmer red #990019
darkest violet #0e002b	darker violet #14003d	dark violet #220066	violet #330099	light violet #8866cc	lighter violet #ac93dd	lightest violet #bface6	muted violet #807399	warmer violet #190099
darkest warm gray #24201b	darker warm gray #332c26	dark warm gray #605850	warm gray #807060	light warm gray #c0b4a8	lighter warm gray #d5ccc3	lightest warm gray #e0d9d2	muted warm gray #807c78	warmer warm gray #807560
darkest warm gray dark #1b1916	darker warm gray dark #262320	dark warm gray dark #403a35	warm gray dark #605850	light warm gray dark #b0a8a1	lighter warm gray dark #cac5bf	lightest warm gray dark #d8d3cf	muted warm gray dark #605e5c	warmer warm gray #605550
darkest white #494949	darker white #666666	dark white #aaaaaa	white #ffffff	light white #ffffff	lighter white #ffffff	lightest white #ffffff	muted white #ffffff	warmer white #ffffff
darkest yellow #3a3a00	darker yellow #515100	dark yellow #888800	yellow #cccc00	light yellow #e6e673	lighter yellow #eeee9f	lightest yellow #f3f3b6	muted yellow #cccc99	warmer yellow #ccaa00
0 #000000 / 1 #a19f9e	2 #b8b6b4 / 3 #c7c5c3	4 #d4d2d0 / 5 #dedcda	6 #e6e4e1 / 7 #eeebe9	8 #f4f2ef / 9 #faf8f5				
'normal' colour warm gray dark	'transparent' colour transparent	'automatic' colour automatic	(not set)					
$AccentColor #403a35	$BorderColor automatic	$CaptionColor #003366	$Color #605850	$GridColor #ffffff	$MapBackgroundAccentColor #f2f2ed	$MapBackgroundColor #f2f2ed	$MapBodyTextColor automatic	$MyColor #000000
$OutlineBackgroundColor #dc490b	$PlotBackgroundColor white	$PlotColor #66cc66	$PrototypeHighlightColor #cceecc	$ShadowColor black	$SubtitleColor automatic	$TextBackgroundColor automatic	$TextColor #000000	$TextColorBlue #0000ff
$TextColorGreen #00ff00	$TextColorRed #ff0000	$TextHighlightBlue #3399ff	$TextHighlightGreen #33ff66	$TextHighlightMagenta #ff66ff	$TextHighlightRed #ff5533	$TextHighlightYellow #ffff00	$TimelineBandLabelColor black	$TimelineColor lighter warm gray
$TimelineScaleColor cool gray	$TimelineScaleColor2 #f2f2e6							

This chart uses the standard 'normal' colours as seen in the built-in colour scheme 'Tinderbox 7'. Note the default palette selection for new TBX documents in v8+ is 'modern' for OS light mode and 'dark coral' for OS dark mode, these are explained further under the Document Settings ▸ 'Colors' tab.

Chart of example shapes, borders and patterns



Note Colours

There are 28 defined 'primary' colours plus 3 'magic' colours that may hold varying colour values: normal, transparent and automatic. The pop-up colour menus, e.g. in the Appearance Inspector's Interior sub-tab, all list the 'primary' colours and such 'magic' colours are contextually appropriate. Note colours are case-sensitive - 'blue' and 'Blue' are treated as separate colours.

Each primary colour can be assigned 3 darker or 3 lighter shades: dark, darker, darkest and light, lighter, lightest. There are 3 further shades: muted, warmer, cooler. Thus **warmer** creates an analogous colour that moves counter-clockwise on the colour wheel, "warmer yellow" is a reddish yellow; **cooler** creates an analogous colour that moves clockwise on the colour wheel, "cooler yellow" is a greenish yellow; **muted** creates an analogous colour that is less saturated than the original colour. The latter 3 shades are not included in the colour swatch displayed on the Colors pane of the Attributes dialog.

Thus for each defined colour there are 9 predefined shades of created dynamically from the base primary colour value, all of whose hexadecimal RGB values are listed in this section. Note that when setting a shade of a named colour in a string, e.g. via action code, the shade value (lighter, darkest, etc.) comes *before* the colour, whereas in the Inspector, the shade pop-up lists shades after the colour being modified. Thus action code would use " *dark* warm gray" as opposed to a supposed "warm gray *dark*" listing as suggested by the Inspector UI layout, " *light* cyan" not "cyan *light*", etc.

If using the TBX file, look at the content of note 'Tinderbox defined colour map' in Map view for a better display of the colours (if reading the website, look here). Colours illustrated in this section list one colour per line with the shades listing from left as per the top-to-bottom order on colour pop-up selectors. The modifier is listed before the 'primary' colour as per usage in action code. The map can also be viewed as an image. This layout is a useful reference to the descriptions below. The list shows examples of border and pattern types as shown in the picture followed by the hex values of all the named Tinderbox colour and their 10 default shades (set automatically). In HTML view use the above linked graphic to see the colour listings.

The most normal application of this colour is to set the on-screen colour of Outline view list items and Map note icons. This is done by setting the selected note's $Color attribute. Incidentally, the colour of note text on Map icons is set automatically for black or white so as to give best contrast with the chosen $Color setting; this can be over-ridden by setting an explicit $NameColor value.

The user can alter a primary colour's actual value or add new primary colours by means of the Attributes dialog's Colors pane. As you add or edit a colour Tinderbox shows you a swatch with the primary colour and its six automatically generated shades. To add a new primary colour you simply give it a new colour name and a hexadecimal colour value and then click the 'change' button. Users wanting a customised set of colours available by default in all their TBXs should investigate use of Configuration files, specifically a colors.xml file; such changes will affect any TBXs created after the custom colors.xml file has been created and the app re-started. This method does not allow the user to customise the colour of 'normal' either at app or TBX level.

Custom colour schemes are also possible; this allows a custom colour set to be shared from one TBX to another and even with other users.

When exported to HTML, literal (named) colours are exported as Hex RGB strings, i.e. 'red' exports as '#FF0000'. Hex values are case insensitive in TBXs, i.e. "#ff0000" is the same as "#FF0000".

Note that the colour 'gray' must use the American English spelling with an 'a'. A colour with the normal English spelling of 'grey' will be treated as a different colour (colour!). Also, when referring to colour Tinderbox in UI captions, code, etc., uses the American English spelling variant 'color'.

The colours listed below are named as per action code usage:

- 'normal'
- 'transparent'
- 'automatic'

'normal'

The colour listed as 'normal' (without the quotes) is not a colour *per se* but rather it corresponds to the **default** or **inherited** colour. Note the lowercase initial letter, **n**ormal, as the spelling is case-sensitive.

In fresh Tinderbox installations, normal defaults to 'warm gray dark' (#605850).

This value cannot be customised by the user, unlike other named colours.

'transparent'

The colour listed as 'transparent' is specific to the $Color attribute of Map adornments alone; note the lowercase initial letter, **t**ransparent, as the spelling is case-sensitive.

Setting a value of transparent makes the body and border of the adornment colourless (i.e. transparent). If the $NameColor attribute ('Text color' on dialog) is still set to automatic then it will use the container's background colour as that against which the appropriate black/white colour will be used for the adornment title (if any).

'automatic'

The colour listed as '`automatic`' is used by the $NameColor and $BorderColor attributes; note the lowercase initial letter, **a**utomatic, as the spelling is case-sensitive. Importantly to note is this colour value *cannot be customised by the user* unlike other named colours.

- For $NameColor, Tinderbox toggles the map title colour from black (#000) to white (#fff) and vice-versa depending on the colour value in 'Color'. If neither black nor white offers sufficient contrast, a yellow colour is used.
- For $BorderColor, the bevel colours used are lighter and darker gradations of the value specified for $Color so as to not contrast unduly with the main 'face' colour ($Color/$AccentColor) of the note (especially in Map view).

If automatic is displayed as one of a note's Displayed Attributes, the colour chip for 'automatic' always shows as a dark red (#6F0000) regardless of the colour actually used, e.g. $NameColor may show as white in Map view but as dark red in the note's colour chip. If Color-type attributes that do not support the 'automatic' token are set to the value 'automatic' the resulting colour actually used is #6F0000. In some later versions, this 'mis-set' colour equates to a green, #00D001.

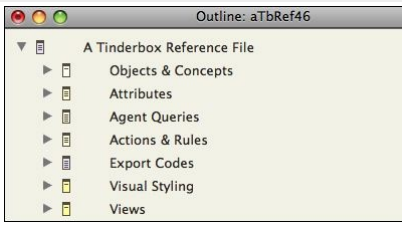
Misc. User Interface Aspects

Age Colouring of Notes

The background colour of Outline view note icons, or dogears in other views, changes according to the age of the note.

The colour reflects the time passed since the note's date of creation (or subsequent latest edit); it is initially light blue, "dries" after about a day to white, and then gradually yellows over the course of a year. The duration since creation/edit for triggering colour ageing transitions is not documented. The JPG compression in the illustration has greyed the light blue of new/recently edited notes, which is more pronounced when seen live in Tinderbox.

In the illustration, 'Export Codes' and 'A Tinderbox Reference File' have been edited within the last day or so, 'Object & Concepts' less recently than that but more recently than 'Agent Queries', etc. 'Views' shows the fully aged colour.

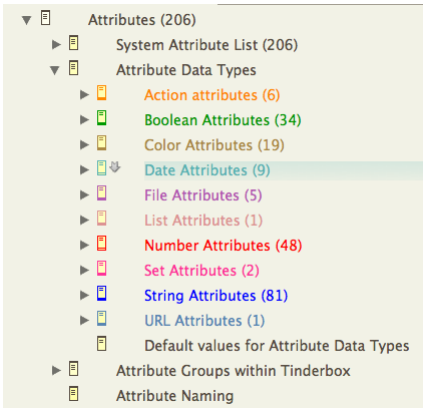


Agent's AgentPriority Status shown in Icon

The $AgentPriority status of an agent's query is shown in its view icon, for all views except Maps.

Disabled (frozen) agents are distinguished by a hollow bar (the screen grab's compression makes it less clear than it actually is on screen!).

Outline, Chart and Attribute Browser are the only views to offer this visual tell-back of $AgentPriority status.



Anonymised Data Sharing

Tinderbox anonymously reports some simple usage statistics to help Eastgate make features easier to discover, and to understand what languages and which versions of macOS are actually used by Tinderbox users. This is controlled via the

Tinderbox Preferences/Register dialog.

Eastgate collect only very general features, such as what kinds of views are being used or how frequently tabs are deleted. Such indications about how users use the app, and which parts they use (few use every feature).

As a long-lived app (born 2001) and as a multi-purpose tool there are a lot of small features added because someone had a real need for it. Often these become obsolete (or simply unused). It can be helpful for a small company like Eastgate to understand how best to prioritise limited resources to best serve its users.

No identifying personal information and no information about the content of a document's notes is transferred. Care has been taken to ensure that this will not impede work when offline or when the server is busy.

The default setting (In Preferences/ Register) is set to opt in (un-ticked). Tick the box if you do not want information to be shared. This might be a generic personal privacy point of view, or because of use on a network whose organisation's rule dislike or ban such background features.

However, aTbRef's author strongly advises allowing sharing unless you have a specific reason not to. It can only help make Tinderbox better.

Attribute name styles in listings

Where attributes are displayed (tables) or listed (pop-up menus) a number of conventions are used to help explain aspects of the attribute:

- Active, defined attribute. Normal text label colour/weight
- *Calculated* (read-only value). Italic in pop-up menus, but a grey label in tables
- ~~Deprecated~~ (do not continue to use!). Strikethrough label: black line in pop-ups, red line in tables (from v9.5.2)
- In tables only:
  - a label in **bold** indicates an attribute with a local value (i.e. discrete value set in this note, vs. default/inherited).
  - (Displayed Attributes) a label in grey indicates a table item defined for a non-existent attribute, e.g. occurring when a note with Displayed Attributes is imported form another TBX document

Locations of listings:

- Inspector Menus:
  - Document: System Attributes
  - Properties: Quickstamp
  - Action: Sort
- Tables:
  - Get Info Attributes pane (including torn-off)
  - Displayed Attributes table
  - Add Displayed Attributes pop-over
  - Attribute Browser view column picker
  - Stand-alone Text window

Autocompletion of input

Tinderbox offers a number of auto-completion methods:

- Displayed Attributes value autocompletion
- Attribute name autocompletion
- Action and query code
- OS text auto-complete

Displayed Attributes value autocompletion

String-valued Displayed Attributes table values offer autocompletion when edited in the Displayed Attributes section of text windows. The auto-complete is quite flexible and does not just match from the start of possible matching values. For example, if you type "kin", autocompletion will offer matches such as "Laurie R. King."

Use the up-arrow and down-arrow keys to select alternative autocompletion possibilities.

To not use the currently suggested auto-complete value, or to use a value that is a substring of the suggested value, use the backspace key (⌫). The edit box shows only the characters actually typed by the user.

If there are very large numbers of possible values, the list used for auto-complete matching is that also used to populate the pop-up value listings for string-based attributes and has the same limits.

If the value being typed begins with a digit, a decimal point, or a - sign, then autocomplete is inhibited for that value. Autocomplete is inconvenient when used with string attributes that happen to have numeric forms, such as Dewey Decimal numbers or numeric IP addresses or social security numbers. Previously in this case, Tinderbox eagerly made suggestions, but they were unlikely to be very helpful.

Set and list attributes offer autocompletion based on discrete list values (as opposed to all list values stored for a given note). For Set or List type attribute only, value autocompletion regards the character '@' as starting a word; this is to support GTD-style tag values with an '@' prefix.

Autocompletion supports values with diacriticals allowing matches beyond the basic ASCII character set. Auto-completion is also used with edit-in-place of Outline view column data.

The token field of the Displayed Attributes picker offers autocompletion options for all attributes that contain the current string, not only those beginning with that string.

Attribute name autocompletion

Where attribute names need to be entered, for instance in the Displayed Attributes selector, setting up new columns in column view or picking an attribute in the system or User Inspector tabs, Tinderbox will offer autocompletions from existin defined attributes.

If the part-typed string only has one possible auto-match, that match is used to complete the current string. If this completion is not wanted, e.g. due to a typo, use the Escape key (*not* the Backspace) to roll back the auto-complete after wh other keys such as Backspace will work as expected to edit the user-typed part of the string.

When searching for attributes in the Get Info/ attributes popover, Tinderbox looks for the search string at any position in the note. Thus, 'URL' offers potential completions of $URL, $ReferenceURL, $SourceURL, and $NoteURL.

Action and query code

Code boxes in Tinderbox will autocomplete action and query code. This occurs in relevant Inspector and Get Info tabs. Dot-operators are offered after a period is typed after an operator or attribute name.

Only in the Action Inspector ' query' tab and the 'query' box of the Get Info ' agent' tab, '==' is offered as an auto-complete for '=' to help users apply the correct syntax. There is legacy support for '=' in queries but its use is deprecated.

If '=' is typed after $Prototype, auto-completions are offered for existing defined prototypes. As the process covers both query and action use, care should be taken to use '==' in a query context, or in a find(), and '=' elsewhere. Note that a find() contains a query so this, for instance, in a $Rule's action a '==' may still be pertinent.

Auto-complete will not try to complete '!=" (not *incorrectly* as '!==').

When autocompleting a dot operator, the autocompletions offered for the first character include only operators that begin for character. For example, typing '$MyString.c' will offer completions of "contains" and "count" but not "icontains". However, if the operator being autocompleted has 2 or more characters, such as "$myString.co", all operators that *contain* that substring will be offered thus including "icontains".

Code editor fields will provide autocompletion for designators in many contexts that accept a designator.

OS text auto-complete

Most input boxes support the Option+Escape (⌥+⎋)shortcut to invoke OS level autocomplete of word in text input boxes. The completions offered are discrete from those offered by the app for things like attribute name or value completion

The main text ($Text) pane also supports this feature.

Automatic Geocoding

Tinderbox will automatically attempt to find the location of notes where an $Address is set and both $Latitude and $Longitude are 0 (zero, i.e. not set). This is done using Google's geocoding service. If a value for latitude or longitude has already been entered no further lookup occurs. Thus if $Address changes, the user must reset $Latitude and $Longitude to zero to update them. The checking of values is done as part of the agent/Rule update cycle.

- Latitude: positive values are North, negative are South
- Longitude: positive values are East, negative are West.

The geocoding service is more likely to work for addresses in the more developed parts of the globe. For such areas including national zip/post codes in addresses will certainly help improve accuracy (or getting response at all).

It is important to note that Tinderbox has no control over the results offered, it simply uses the result—if any—returned by Google's service. Thus Tinderbox must poll Google with $Address data but this does not guarantee $Latitude or $Longitude values or their accuracy. Google also rate limit geocoding accesses, normally suspending service access for a day or so if over-used. Thus this feature should not be relied on for bulk geocoding.

If a geocode action is successful, the resolved address is stored in $GeocodedAddress.

Automatic re-open of last-used TBX

There is no Tinderbox setting for this. However, in the OS System Preferences, General, there is an option 'Close windows when quitting an app'. If left un-ticked, Tinderbox will re-open any documents open when the app was last closed.

Automatic Saving

From version 6, Tinderbox uses the OS's build-in versioning capabilities. Past versions are accessed via the File menu ▸ Revert To option. Recent versions in the current session may be listed otherwise past version can be accessed via time-Machine-like interface. Note that this does *not* require the user to be running Time Machine.

The past versions are stored by the OS and are not designed to be readily accessible to the user. Either make use of the 'revert to' option if seeking past versions or use Time Machine if available.

The document autosaves in the background whenever a semantic change occurs. In practice this means that should a (thankfully rare!) crash occur very little recent data is lost. Even unsaved document, if open for a short while, will get saved/recovered after a crash.

Badges

Notes viewed in any of Outline, Map, Chart or Timeline view may be decorated with an optional 'badge'. This is a small image that appears in the upper right hand corner of the note in map view and to the left of the note name in other supported views. Badges in Map view are described in more detail elsewhere.

Badges may be selected from a badge picker pop-over by clicking on the '+' symbol at top right of a map icon or to the right of the note icon in other view types.

The note's badge is controlled by a string attribute $Badge, which holds the name of the badge (app-installed badges use all-lowercase names). The value of Badge may be set through inheritance, rules, or actions.

Users can alter/extend the range of badges by adding user-created badges, placing suitable PNG files (with the .png extension) in the support folder described. Badges are usually 32 x 32 pixel PNG format files. Square icons are expected but different sizes can be used (e.g. they were created by some third party and cannot easily be re-edited). In such case, consider use of $BadgeSize to control the size at which the icon is drawn at normal view zoom level. $BadgeMonochrome is used to indicate use of monochrome icon sets.

In Chart view, the maximum display size of badges is limited to 16px.

*flag blue

Blind typing in view window

View pane has no existing notes

When you type 'blind' into a new document window or view pane that contains no notes, Tinderbox will create a new note. This occurs Outline and Map views only as other views always include the document as a top level note so a view of that type is never empty.

View pane has existing notes

If the view pane already contains notes, in all main views except Attribute Browser, Tinderbox will attempt to select the first note matching the blind-typed string and scroll it into view. In all cases, potential matches must be 'visible' even if in currently off-screen part of the view. In the case of Outline and Chart views, items in collapsed branches are not matched.

Tinderbox has slight delay in scrolling the view in response to the blind-typed value as the scrolling could inhibit further input; this should aid using the feature where there are many items with a common stem to their $Name.

Thus, simply typing will cause Tinderbox to select the note—if in the current view—that matches the typed string. Note that:

- where a $DisplayName is set, typeahead matches $DisplayName instead of $Name.
- pressing [Esc] clears the typeahead buffer
- waiting 1.5 seconds before blind typing again also allows the typeahead buffer to clear.
- typeahead accepts flexible abbreviations. For example, "Saf4" will match a note named "Safari 4". When typing an abbreviation, Tinderbox tries to find the best available match if no exact partial match to $Name/$DisplayName is available.
- matching is triggered when 2 letters or more are blind typed into the view.

While blind-typing to locate notes in the view pane, the (backwards) Delete key deletes the most recently-typed character, not the selected note. The Escape key cancels blind typing.

Built-in composites

The File menu allows the addition to the current document of pre-defined composites. The first time a built-in composite is added a root-level 'Composites' container is created. Any subsequent addition go in the same container.

If a built-in composite is added to a document in which the default value of $NeverComposite is `true`, the components of the built-in composite have $NeverComposite set to `false`, ensuring the newly added composite works correctly. If Tinderbox needs to create the 'Composites' container, it sets the OnAdd action to `$NeverComposite=false` to ensure any user-created composites work correctly when added there.

The OnAdd actions for the built-in containers for Prototypes and Templates set `$NeverComposite=true`. This reduces the likelihood of unwanted accidental composites.

The built-in Composites container is set to export neither itself ($HTMLDontExport) nor its children ($HTMLEXportChilren). This can be verified in the Export Inspector.

The built-in composite types are:

- List composite
- Expense composite
- Lecture composite
- Task composite

List composite

The built-in composite for lists has a single dark header, followed by one or more items. When a new item is dragged to the bottom of the list, its colour, size, and position are set automatically by the list item's $OnJoin action.

The composite contains 2 items:

- **(header)**.
- **(item)**.

Expense composite

The built-in composite for expenses is useful for tracking costs and purchases, e.g. for work, research, holiday or household.

The composite contains 5 items:

- **where**.
- **what**.
- **amount**. The amount is stored in $MyNumber and set with a currency prefix (default is "$") via the notes $DisplayExpression
- **who**.
- **notes**.

Lecture composite

The built-in composite for lectures is useful for conferences and syllabus planners. It is typical of a number of situations where it is desired to capture various facets about an event.

The composite contains 7 items:

- **where**.
- **title**.
- **author**.
- **author 2**.
- **abstract**.
- **note**.
- **question**.

Task composite

The built-in task composite is good for tracking a task that combines dates with people and resources.

The composite contains 5 items:

- **goal**.
- **who**.
- **due date**.
- **prerequisites**.
- **facilities**.
- **task**.

Built-in export Templates

A method for quickly adding export templates follows that for built-in Prototypes. A File menu item displays a sub-menu of built-in export templates that can be added to the current document.

If no export templates already exist, a root-level container 'Templates' is added. The 'Templates' container's $OnAdd is set to `$IsTemplate = true;`. Also the built-in prototype 'HTML' ('HTML Template' in older versions) is added to the existing built-in Prototypes container or the latter container is also added. If a 'Templates' and/or 'Prototypes' container exists these are used for any newly added notes instead of creating new containers.

From v8, the built-in Templates container is required to be a top-level note; an interior note names "templates" or "Templates" will not be adopted as the document's built-in templates container.

From v9.6.0, when a document is loaded, Tinderbox checks all notes that are used as (a) an HTMLExportTemplate, (b) an EmailTemplate, or (c) a PosterTemplate. If a note is in use as a template, it is marked so $IsTemplate is true. Notes that are not currently in use as templates are not marked with $IsTemplate as false; a note my advertise its willingness to serve as a template even if it is not in use.

The offered built-in templates are:

- **HTML**. HTML5 page export, with includes for all descendants. This inserts two templates in the form of a wrapper template, which is applied to the main note to export and which then recursively calls a per-item template to include all the main page's descendants. The item template is inserted as a child of the wrapper, emphasising the usage. To export only the current note's content use the new **HTML Single note** template.
- **HTML Single note**. (v9.5.0) HTML5 page export, but exports **only** the current note.
- **OPML**. OPML export, with includes. This inserts two templates in the form of a wrapper template which is applied to the main note to export and which then recursively calls a per-item template to include all the main page's descendants. The item template is inserted as a child of the wrapper, emphasising the usage.
- **Scrivener**. OPML export, with includes. This is actually the same code as for OPML, but offered to aid non-technical Scrivener users and to allow easy branching for any possible Scrivener-only variations.
- **Preview**. This is the template auto-inserted for zero-configuration internal preview, but can be imported from the Templates menu as it may be desirable to configure the template before use. This replaces the 'preview' note in /Hints. This is the default template for Markdown notes. The template automatically includes the CSS styles defined in the Preview style note. The Preview template can be customised to add headers, footers, or other elements for more structured internal preview use.

Once templates are added the user can modify the template code.

In addition to the above, the users own selection of templates stored in the application support ' templates' folder are also available.

Note that a prototype that is also an export template (e.g. 'HTML Template', below) will not be listed in the Template pop-up menu. This is so that a note can inherit template status via a template without cluttering the template listing with prototypes.

Built-in Hints container

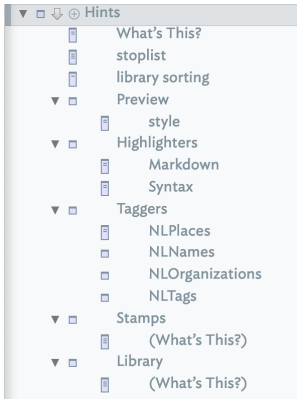
An optional (from v9+) built-in set of containers can be added to a document from the File menu. The container is added at document root and usually at the end of the outline:

`/Hints`

The hints container holds a number of notes and containers that customise Tinderbox to specific tasks and problem domains. Hints will allow Tinderbox to automatically control:

- the document's stoplist.
- the library sorting term list.
- configuration for template-less styled preview of (Markdown-formatted) pages.
- configuration and addition of highlighters.
- configuration and addition of taggers.
- configuration and addition of stamps.

All features controlled via Hints are optional. Some features may require macOS 10.15 or later.

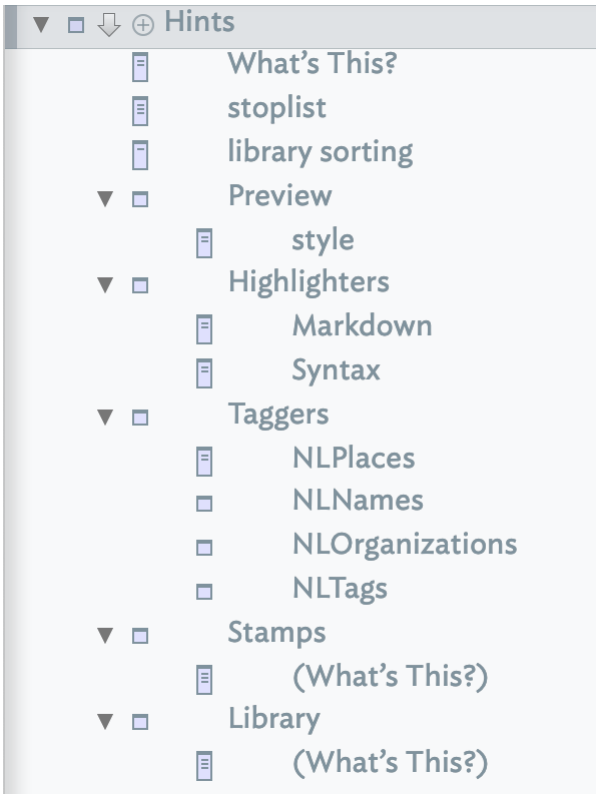


The stoplist note

`/Hints/stoplist`

The stoplist note created by Built-In Hints takes precedence over any other stop list notes in the document. Therefore, if Hints are added to the document, pre-existing stop lists should be merged with the Hints-based version.

Stoplists may contain comments: any lines that begin with # or // will be ignored.



The library sorting note

`/Hints/library sorting`

New to v9.6.0, the container sort transform 'library sorting' permits notes to be sorted as they might be filed in a library. By default, library sorting ignore initial words "a", "an", and "the".

The sorting note at `/Hints/library sorting`, if present and not empty, has $Text containing a list of words that will be ignored for library sorting. This note can be customised for different languages and filing practices.

Library sorting also respects some locale-specific rules for handling diphthongs and diacritics, and is always case insensitive.

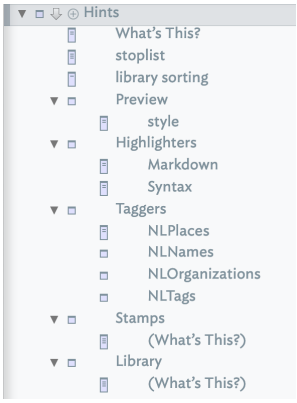
Preview

`/Hints/Preview`

This container within the built-in Hints container holds notes that help configure the documents template-less preview using Markdown notes. When previewing Markdown, Tinderbox shows internal text links to other notes as well as web link

The container holds one item:

- a style sheet.



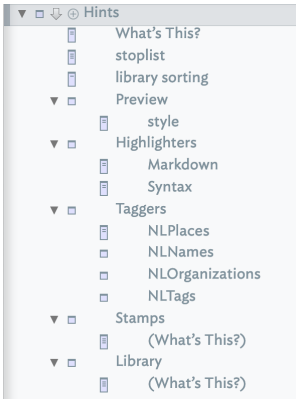
Preview style sheet

`/Hints/Markdown/style`

Styles for Markdown previews are now taken from this note is created as part of the built in Hints.

If a stylesheet already exists in the Markdown support folder, it will be imported automatically; if not, a simple starting set of styles will be created.

Highlighters



`/Hints/Highlighters`

This container holds notes that describe ways to highlight the text of a note. Any note can opt in to highlighting by setting the $SyntaxHighlighting value. For example, setting $SyntaxHighlighting to `markdown` would tell Tinderbox to use the highlight methods found in `/Hints/Highlighters/markdown`. Adding the built-in Hints container, generates two specimen highlighter notes in the `/Hints/Highlighters` container:

- Markdown. This is intended for users of Markdown.
- Syntax. A specimen file to show syntax styling in action, rather than represent any particular syntax.

The highlighting methods are a list of regular expressions and formats. The first line of a highlighting rule must not be indented; additional lines for each rule must be indented. Empty lines may be added between rules for clarity. For example:

```
pattern:parent|grandparent|nextSibling|previousSibling
    color: red
```

would locate any of these words and colour them red.

A highlighter note may contain *multiple* style declarations reflecting the fact that any given note can only apply a single highlighter.

Highlighters set the base colour of notes to their $TextColor rather than to "black" (#000000). **NOTE**: this styling sets the RTF layer styling of text. Not all styling can be exported via (HTML) Export and deleting a rule does not not 'un-set' the style previously set by that rule.

Comments are preceded by // and continue for the rest of the line, thus:

```
pattern:parent
    color: red // perhaps dark red is better?
```

Highlighters can use any of the following rules:

- **color:** the text colour, a named colour or hex code value, e.g. `color: green` or `color: #00ff00`
- **background:** the text background colour (highlight), a named colour or hex code value, e.g. `background: #ffff00` or `background: yellow`
- **size:** text size in points, e.g. `size: 14`
- **line-spacing:** line-spacing as a multiple of the normal line spacing, e.g. `line-spacing 2`
- **indent:** indent the paragraph in points, e.g. `indent: 32`
- **first-indent:** indent the first line of a paragraph in points, e.g. `first-indent: 16`
- **bold:** text bolded as a yes/no boolean, e.g. `bold: yes`
- **italic:** text italicised as a yes/no boolean, e.g. `italic: yes`
- **strike:** text struck-through as a yes/no boolean, e.g. `strike: yes`
- **underline:** text underlined as a yes/no boolean, e.g. `underline: yes`
- **wildcards:** yes/no boolean instructs the pattern to ignore regular expression meta-characters such as "*" and "+", e.g. `wildcards: no`
- **case-sensitive:** yes/no boolean instructs the pattern to regard upper- and lower-case letters as equivalent, e.g. `case-sensitive: no`

Highlighters allow you to set line-spacing as a multiple of the normal line spacing.

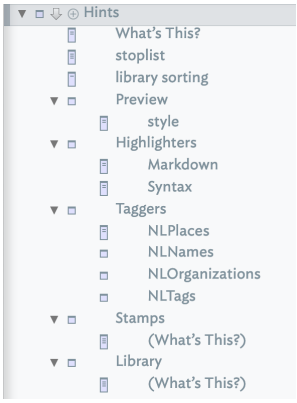
Highlighters allow you to set the indent of the paragraph, as well as its first-indent—the indentation of the paragraph's first line.

Possible conflicts with other features

Do not use this feature on notes using the built-in ' Action' prototype for action code syntax highlighting. Use one or the other.

As highlighters alter the styling of $Text, existing styling including highlighted passages, may be altered. Combine the two with care.

Taggers



This container within Hints holds notes that provide hints to Tinderbox's AI to help tag notes.

`/Hints/Taggers`

The name of each built-in note within the Taggers container corresponds to a Set-type attribute. See below for the additional set-up needed for user-created tagger notes.

Taggers, tags, and terms

A tagger note defines one or more 'tag' items that the tagger will add to/remove from the associated attribute.

Each 'tag' is defined by one of more 'term' items. At simplest a tag can be self defined as the tag name. The detail of the syntax for defining tags and terms is addressed further below.

For information about use of multiword tags and terms see detail under Tagger file syntax.

Scope of Analysis

A tagger looks at the content of a notes title ($Name) and text ($Text). Thus, it is important to understand that tag detection occurs *only* in the $Text and $Name of a note and *not in any other attribute*.

Built-in Taggers

There are the tagger notes added by default when the 'Hints' container is first added to the document. These tagger notes, all use 'NL' name prefixes, and have system attributes already in the document.

Three of built-in taggers, NLPlaces, NLNames, and NLOrganizations use underlying Apple NLP (see more on Natural Language Processing) features to match terms. Thus, for NLPlaces, Apple NLP is leveraged to detect any $Text content that appears—to the NPL algorithm—to be a location or place-name (indeed, exactly what those are is undefined), placing that term in the note's $NLPlaces attribute. As a result there is no tell-back as to why a term is auto-populated to the associated attribute.

Although 'NLTags' has an associated system attribute, it does not use any NLP and is essentially a quick stubs for using user-created tags without undue extra set-up. For this and user-defined taggers, the relationship is more simple. A tag addition/removal is linked to a case-*insensitive* match to any term defined for a tag.

Detection is dynamic: a match to a term in a note's $Text or $Name adds the relevant term to associated attribute for that tagger, removal (of all instances) of the term from $Text or $Name removes the tag from the associated attribute.

Attributes used by taggers

A tagger works with an associated Set-type attribute. Once assigned a tagger, Tinderbox makes the note read-only and this cannot be undone—manually or automatically—short of editing the raw XML of a document. The reason the attribute is marked as read-only is to ensure only the tagger may add or delete values in the course of its work.

As well as adding a tag if a term is matched, if a tagger formerly applied a tag but source term(s) are no longer detected, the tagger will remove that tag from the note's attribute. In effect the attribute keeps a dynamic reference of current matched terms in a note's $Text or $Name.

Naming of taggers

Every discrete note in the 'Taggers' container is a distinct tagger. Each tagger works with s set-type attribute of exactly the same (case-sensitive name). Thus the tagger names should be one that meets attribute naming rules.

Built-in taggers are provided for $NLPlaces, $NLNames, $NLOrganizations, and $NLTags. Thus, the built-in tagger note 'NLTags' maps to system attribute $NLTags.

A note will be flagged for tagging after its $Text or $Name is edited, and the tagging will be performed when the note is visited by the rule manager (i.e. the same cycle of operations that runs note rules).

Setting up for first run

There is some internal work associated with setting up a tagger, regardless of the size of the document. So, depending on the speed of the host Mac, it may take up to 30 seconds before the initial use of a new tagger (i.e. results are seen in the associated attribute). Once set up, performance—i.e. as in term detection—is essentially instant.

How often does tag detection run?

Tag detection runs all the time, but is throttled in large complex documents for pragmatic performance reasons. Despite this, taggers are not a full-on, always-updating service as with a rule. This reflects that they are a form of advisor and they need to run (using AI routines in the OS) in harmony with other Tinderbox processes.

If in doubt about tagger output being up to date, close and re-open the current document as all Taggers run at document start. As this feature settles into its place in the app, the control of tagger operations (without affecting other task) will doubtless be improved. See more about performance and update time.

The tagger mechanism also runs the sentiment analysis process and dominant language detection.

Editing taggers for 'NL' system attributes

When adding the Hints container to a document, the Taggers sub-container includes blank (i.e. no $Text) tagger notes for each of the NL-prefixed system attributes in the ' AI' group. Adding tags (see below).

Users should not create new taggers for any other built-in attributes. Instead, create a custom tagger as described below.

Adding custom taggers

Further taggers may be created for any user Set-type attributes. New taggers are placed in the same container as built-in ones. To add a totally new tagger involves 2 tasks (these can be done in any order):

- add a user Set-type attribute (it must be a Set) using the same name as will be used for the tagger. Pick a name not already used for an attribute or note name.
- create the tagger note with the same name as the attribute.

Tinderbox will now make the attribute read-only (this can't be undone) and will populate that attribute in any notes where the new tagger finds a match. Be aware that creating the tagger file alone does *not* auto-generate attributes, so user attributes must exist if they are to use taggers.

Customising built-in taggers

The built-in NL-prefix taggers may also be customised, if desired, using the same syntax (see below) as used for custom taggers.

Tagger file syntax

Described here.

Term matching and case

A tagger's defined terms are matched case-intensively in any note's $Text. Thus a term `Tinderbox` with match both "Tinderbox", "tinderbox" or any other case variant of the word.

Terms and NLP

Unlike user-defined taggers, the NL taggers may add tags the NLP process considers a match but will be unable to explain why. This is a current limitation and weakness of such NLP tools: they cannot explain their decisions, so triaging the cause of false positive (things that should not be matched) or false negative (things that should be matched and aren't) is difficult. Note that the NLP frameworks are Apple's and so any such errors are not caused by Tinderbox, but the underlying Apple code.

The tagger matches in the $Text are not marked or indicated in any way. In the case of custom taggers, or custom additions to built-in NL taggers, it is assumed the user knows the terms they have defined. For the built-in NL taggers additional Apple NLP frameworks are addressed.

Regular Expressions

Taggers **do not support** regular expressions. Why? Because taggers work with the OS' NLP libraries and those do not understand regex.

Taggers and agents

Described separately here.

Technical requirement

Taggers are require macOS 10.14 or later.

Other aspects of Tagger use:

- Tagger file syntax
- Taggers and agents
- Tagger performance and update time

Tagger file syntax

This information applies both to user-created taggers and customisation of built-in taggers.

Each line of a tagger note's $Text describes one possible 'tag' (i.e. term to be added) followed by one or more 'term't is term terms that are being searched for in $text by the tagger process.

The general syntax is

```
tag: term 1; term 2; term 3;
```

The first term is the 'tag' separated by a colon from the trigger 'term(s)'. Each discrete term is delimited from others by a semi-colon (in the convention of Tinderbox list delimiters) *but unlike other lists in Tinderbox, there **must** be a semi-colo after the last item*: if a terminating semi-colon is omitted the last listed term may not get evaluated (without any error waring).

White space. Space after colons and before/after semicolons is ignored. The layouts below are thus functionally equivalent:

```
tag:termX;termY;termZ;
tag: termX; termY; termZ;
```

Shorthand definition method. If the only term defined matches the tag, as in:

```
Eastgate:Eastgate;
```

…then a shorthand form may be used, i.e. just the word, alone, on a single line (note no line terminator colon/semicolon is used):

```
Eastgate
```

Tags should be single word. Although a multi-word ('price-sensitive') or multiple word ('price sensitive') tags can be defined they are not recommended/supported. Tags should thus be defined as a single word.

Terms and multiple words. Unlike a tag, a term may be a multi-word string (e.g. 'first-name last-name'). Note, however, that this style of term *is not intended for matching long texts* (there is no fixed limit, just use multi-word terms sparingly)

Definition ending markers. Lines forming tag definitions **must** end with a semi-colon. The latter does not apply to comment lines (code comments are described below).

Line wrap. Although the tagger file's $Text may wrap in the text pane, a single definition runs from the start of a line to the first line break encountered regardless of soft wrapping.

Line breaks line break character is treated a a break between tag definition (noting the need to mark the end of definition string (above).

Blank lines. Any blank lines in the tagger note are ignored so may be used to separate the definition strings for clarity of reading.

Comments in taggers. Any line starting with a hash (#) symbol is treated as a comment and is not evaluated for tags/terms.

Summary

The format is thus 'tag:term1;term2;etc…' or simply 'tag'. Note the short form uses no end marker but must be followed by a line break.

Therefore, if the NLTags tagger file contains the line:

```
Eastgate:Eastgate;Tinderbox;Storyspace;Mark Bernstein;
```

… then $NLTags will add the tag "Eastgate" if the text or title of the note mentions any one or more of the words "Eastgate", "Tinderbox", "Storyspace", or "Mark Bernstein".

Thus each discrete tag configuration forms *a single line*, i.e. line through to the first following line break character. Note that lines with tag definitions **must** end with a semi-colon.

Each tagger note can define one or more discrete 'tags'—values to add to/remove from the associated Set-type attribute if matched, one tag per line. In turn, each tag may be defined by a semicolon-delimited/terminated list of one or or more 'terms'.

Taggers and agents

Taggers are particularly useful to agents. Often, an agent needs to look for significant or interesting terms in the text:

query: `$Text.icontains("Caesar") |$Text.icontains("Cato") …`

In projects that have lots of text, these agents have a difficult task: each clause might need to search all the text in the document, and each clause needs to build a regular expression matcher. The tagger is more restricted—it searches only fo entire words, not for regular expressions—but it is significantly faster and it runs only when the text of the notes changes. In the example above, we could replace the query with the simpler query

query: `$NLTags.contains("Rome")`

and add a single line to the $Text of the /Hints/Taggers/NLTags note:

```
Rome: Caesar; Cato; Cicero; Aggripa; Brutus; Decius; Catullus
```

Now, the agent needs only to examine set $NLTags, which requires far less work than searching all the text of a large document.

Tagger performance and update time

This a new feature and Tinderbox is, at present, very conservative about the frequency at which Taggers run. It may be too conservative. A concern is that not all Macs have dedicated neural engine cores, and setting up a tagging scan requires a certain amount of prep work. The general idea here is that you want rules and agents to run now, but taggers are a sort of background discovery task.

The original notion for taggers is that they are slowly mining data and marking down relevant things they discover. They are envisioned like good research assistants; they may not get things done right away, but eventually they will get answer that are interesting and usually useful.

As taggers all run on document open, saving and re-opening the document is a reliable way to force the process to update on demand, at least until this feature evolves further control methods. Editing a note's text ($Text) will also update the tagger. However, it will **not** update the displayed note. So, to see the effect—e.g. in the Displayed Attributes table—it will be necessary to re-select the note (or click the current tab's label to refresh the whole tab display).

Another way to force everything to be retagged is to edit the $Text of the tagger-defining note. However, tagging tasks and indexing tasks might be delayed in large documents when lots of things are going on, e.g. complex actions in rule or edicts using a lot of regex tasks, or very complex find() queries.

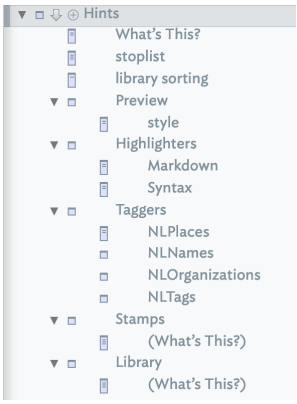
Editing a tagger note marks the tagger as needing to be reloaded when the index is next revised.

Stamps

From v9.1.0, if the default Hints folder set is added to the document., existing stamps are now also stored and edited as notes within the Stamps Container of the built-in Hints container. The container sets the built-in " Action" prototype as OnAdd (also adding the prototype to the document if not already present.

Re-adding the File ▸ Built-In Hints will create the default Stamps container, even if the TBX already a Hints container. Any pre-existing/modified Hints content is not altered by this action. This will be needed for notes adding the hints container under v9.0.0.

Once the Stamps container is added, the Stamps Inspector can still be used for managing stamps.



> Note: use of the new mechanism can be reversed, by deleting the Stamps container, though this is **not** envisaged as a regular task. Some of the newer features are ignored in the old style of management, e.g. comment and unlisted stamps (see below) function as

The default container includes a comment note '(What's This?)'—see Comment stamps below.

Individual stamps

Within the /Hints/Stamps container, each Stamp is stored as an individual "Action"-prototyped note; the stamp code is still stored in the TBX as a stamp rather than a note.

Nesting of stamp notes. Individual stamp notes must **not** be nested if they are to function correctly via both Inspector and the stamps menu. But note the next below…

Nesting stamps in Stamps menu. To make stamps that appear as a Stamps menu submenu, continue to name them using the format 'sub-menuName:stampName'. This nesting behaviour applies *only* to the stamps menu, and not the Inspector or the actual storage of stamp notes.

Separators in the Stamps Container

From v9.6.0, when building the Stamps menu, separators in the Stamps container now become separator menu items (if not named) or disabled menu items (if the separator has a name).

Stamp name. The name of the note is the name of the stamp, as seen in the stamps menu. If a legacy stamp and a new stamp note have the same name, the stamp note's action will be performed. This stamps should have a unique $Name, at least within the Stamps container.

Stamp Action code. The stamp note's $Text hold's the stamp's action code. Use of the Action prototype ensure notes use action code syntax colouring and offer auto-completion of operator and attribute names.

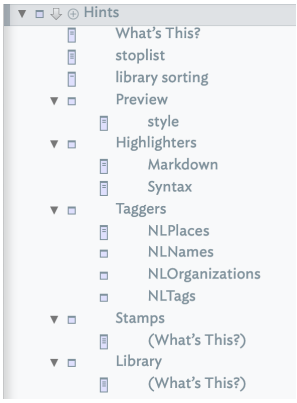
Comment stamps. Any notes in the Stamps container with the name in parentheses are treated as commentary rather than active stamps. For example, the default note '(What's This?)' explains the container and contains no stamp. Stamps may still contain inline action code comments.

Unlisted stamps. Notes in the Stamps container with a name beginning with a period are not listed in the Stamps menu, but may still be used with the stamp("stampName") action.

Note: although the stamp notes reflect the Stamps Inspector's stamps, the two lists' ordering is not synchronisedote that the Stamps menu uses the Inspector's order and not that using in this folder, if different.

Library

From v9.1.0, functions written in action code may be defined in notes placed in the `/Hints/Library` container.



Notes here can contain one or more discrete action code function(s). The names of the notes are not the function so may simply be indicative of purpose. Note how this differs from stamp notes.

Tinderbox examines all library notes for function(s) and runs an action code in the note when the note $Text is edited (i.e. when the $Text is altered, not just when the note takes focus). This means Library notes can be a useful method for testing code. So if the contents of a Library note is:

```
$Color = "green";
```

… on selecting the note noting happens, the note's color is 'lightest black' as inherited via the 'Action' built-in prototype. If the note's $Text is edited, even just adding a space or line return, the note's colour turns to 'green'.

As this evaluation includes functions beware functions that alter other notes, as accidental changes may occur is a function is triggered by accidental 'evaluation of the note.

If wanting to store action code without it being evaluated (e.g. a complex rule's code) make a note elsewhere in the document using either the 'Code' or 'Action' built-in prototypes

Library notes can be nested with no implication other than for convenience of organisation and storage. Notes can be copied between documents, though it is left as an exercise for the user to ensure any referenced attributes are present in the new document. It is recommended the if planning to share library notes between files that any needed user attributes (and their type, etc.) are noted as a comment in the function.

Built-in Prototypes

The File menu has a sub-menu listing built-in prototypes. These are a series of prototypes pre-configured for quick use. On first addition of one or more of these predefined prototypes to a TBX, Tinderbox adds a root-level container called 'Prototypes'. After the first addition, any other built-in prototypes added are put in the same container.

The built-in types only use system attributes for their customisations, though the user can further customise the basic prototype with user attribute once added to a TBX. The sub-menu is disabled in read-only documents.

Note that a prototype that is also an export template (e.g. 'HTML Template', below) will **not** be listed in the Template pop-up menu. This is so that a note can inherit template status via a template without cluttering the template listing with prototypes.

Some built-in templates are not listed in the normal menu listing and these are denoted in the list below with a '†' suffix. Such prototypes are added by Tinderbox on first use of the associated feature. Once so added to a document they can be viewed and edited as with any other built-in prototype. Below is a listing of built-in prototypes:

- Action.
- Code.
- Dashboard.
- Event.
- Exploded Notes.
- HTML.
- Imported From DEVONthink (†).
- Imported From Finder (†).
- Imported from Notes (†).
- Imported From Tot (†).
- Markdown.
- Person.
- Place.
- Poster.
- Reference.
- Task.
- any shared prototypes.

The prototypes of watched features (except Tot) specify that watched notes are imported with a $TextBackgroundColor of white, even if using dark styles in Tinderbox, since external documents are most likely compatible with light background colours.

There is nothing stopping the user from further modifying these prototypes once they are in a TBX, for instance by adding user attributes to the Displayed Attributes, etc.

The default 'Prototypes' container

The TB-default-created "Prototypes" container is set with $OnAdd of `$IsPrototype=true;` to ensure that any child notes added by the user as automatically set as prototypes. The container is also set with $HTMLDontExport as `true` ar $HTMLExportChildren as `false`. That change ensures that neither the container or its content get exported as this is usually the desired condition. Adding a built-in Template will cause this container to be added, in order to hold the HTML Template prototype needed for the templates.

Storing prototypes for re-use

See Shared prototypes.

Prototype: Action

Action

This marks the note as holding action code, as opposed to other types of code. Its primary intended use is for support of internal stamp notes.

Based on the Code prototype, this also sets $IsAction for a note, enabling Tinderbox to offer auto-complete for action code operators and syntax colouring of code. Note this highlighting latter does **not** use the general syntax highlighting methods but uses a separate in-app colouring previously reserved for code input boxes.

So this uses broadly the same settings as the Code prototypes except that $IsAction is set to *true*. Use for things like storing long actions such as deeply nested if() statements, or for command line code for use with runCommand().

For notes using this prototype and which with be exported, e.g. CSS code, boilerplate text, the prototype-inherited default for $HTMLDontExport (as seen below) should be reversed locally in the code note itself.

This prototype's non-default customisations other than $IsPrototype, $Name, and intrinsic map features (height, width, etc.) are listed below.

$Badge: "design".
$Color: "lightest black".
$OnAdd: `$NeverComposite=true;`
$IsAction: `true` (ticked).
$HTMLDontExport: `true` (ticked).
$HTMLFirstParagraphEnd: "" (empty string).
$HTMLFirstParagraphStart: "" (empty string).
$HTMLIndentedParagraphEnd: "" (empty string).
$HTMLIndentedParagraphStart: "" (empty string).
$HTMLParagraphEnd: "" (empty string).
$HTMLParagraphStart: "" (empty string).

$NoSpelling: `true` (ticked).
$SmartLinks: `false` (un-ticked), v9.5.0. Existing prototypes need updating manually.
$SmartQuotes: `false` (un-ticked).
$ParagraphSpacing: 0.
$Tabs: "0.375;0.375;0.375;0.375;0.375;0.375;0.375;0.375;".
$TextFont: "Andale Mono".
$Ziplinks: `false` (un-ticked).

Prototype: Code

Code

Broadly the setting same as the HTML prototype except that $IsTemplate remains as the default *false*. Use for things like storing long actions such as deeply nested if() statements, or for command line code for use with runCommand().
For notes using this prototype and which with be exported, e.g. CSS code, boilerplate text, the prototype-inherited default for $HTMLDontExport (as seen below) should be reversed locally in the code note itself.
This prototype's non-default customisations other than $IsPrototype, $Name, and intrinsic map features (height, width, etc.) are listed below.
$Badge: "design".
$Color: "lightest black".
$OnAdd: `$NeverComposite=true;`
$HTMLDontExport: `true` (ticked).
$HTMLFirstParagraphEnd: "" (empty string).
$HTMLFirstParagraphStart: "" (empty string).
$HTMLIndentedParagraphEnd: "" (empty string).
$HTMLIndentedParagraphStart: "" (empty string).
$HTMLParagraphEnd: "" (empty string).
$HTMLParagraphStart: "" (empty string).
$NoSpelling: `true` (ticked).
$ParagraphSpacing: 0.
$SmartLinks: `false` (un-ticked), v9.5.0. Existing prototypes need updating manually.
$SmartQuotes: `false` (un-ticked).
$Tabs: "0.375;0.375;0.375;0.375;0.375;0.375;0.375;0.375;0.375;0.375;0.375;0.375;" (see note).
$TextFont: "AndaleMono".
$Ziplinks: `false` (un-ticked).
Note: newly added Code prototype notes offer a dozen tab stops; existing older Code prototypes are not affected, so $Tabs may need to be updated manually if desired.

Prototype: Dashboard

Dashboard

Preconfigured for map dashboard use.
This prototype's non-default customisations other than $IsPrototype, $Name, and intrinsic map features (height, width, etc.) are listed below.
$DisplayedAttributes: "SubtitleColor;Subtitle".
$DominantLanguage: "en".
$Subtitle: "42". (This specimen value of '42' to aid initial subtitle display/configuration.)
$NameAlignment: "center".
$NameFont: "RingsideCondensedSSm-Book".
$SubtitleColor: "light green".
$SubtitleOpacity: 100.
$SubtitleSize: 250.
$TitleOpacity: 50

Prototype: Event

Event

Intended for use with calendar/timeline related notes.
This prototype's non-default customisations other than $IsPrototype, $Name, and intrinsic map features (height, width, etc.) are listed below.
$DisplayedAttributes: "StartDate;EndDate;DueDate;TimelineBand".
$Badge: "clock".
$AccentColor: "dark red".
$Color: "muted red".
$HoverExpression. If a $StartDate is set, show $StartDate, and $EndDate (if set) else no expression data is generated:

```
if($StartDate){$StartDate.format('l')+if($EndDate){' - '+$EndDate.format('l')}}
```

$Participants: from v9.6.0.

Prototype: Exploded Notes

Exploded Notes

This is added automatically to the child container of an exploded note. In order to allow customisation before first use, this prototype can be added to a document from this menu without first invoking an Explode operation.
Note that the prototype is capitalised deliberately. This ensures it is not matched by queries looking for, and acting on, the 'exploded notes' container that is generated by all 'Explode' operations.
This prototype's non-default customisations other than $IsPrototype, $Name, and intrinsic map features (height, width, etc.) are listed below.
$DisplayedAttributes: "ChildCount".

Prototype: HTML

HTML

This used for correctly configuring notes used as export templates. Historically those were, and most often still are used for exporting HTML but can be used for formatting any export template. Here, normal writing affordances like changing straight quotes to typographic quotes are harmful so are disabled.
In older versions, this prototype was called **HTML Template**: existing files are unaffected. The change is in name only and not function.
Although this prototype has the $IsTemplate flag set, app dialogs (correctly) do not list it as an available template.
This prototype's non-default customisations other than $IsPrototype, $Name, and intrinsic map features (height, width, etc.) are listed below.
$DominantLanguage: "en".
$Badge: "tools".
$Color: "lighter black".
$OnAdd: `$Prototype="HTML";`.
$HTMLDontExport: `true` (ticked).
$HTMLFirstParagraphEnd: "" (empty string).
$HTMLFirstParagraphStart: "" (empty string).
$HTMLIndentedParagraphEnd: "" (empty string).
$HTMLIndentedParagraphStart: "" (empty string).
$HTMLParagraphEnd: "" (empty string).
$HTMLParagraphStart: "" (empty string).
$IsTemplate: `true` (ticked).
$NoSpelling: `true` (ticked).
$ParagraphSpacing: 0.
$SmartQuotes: `false` (un-ticked).
$Tabs: "0.375;0.375;0.375;0.375;0.375;0.375;0.375;0.375;0.375;0.375;0.375;0.375;" (see note).
$Text: "<!-- Tinderbox: Prototype: HTML -->" (pus two trailing line breaks)
$TextFont: "Andale Mono".
$Ziplinks: *false* (un-ticked).
The aim of the default $Text is to help indicate, when exporting pages composed from several notes, from which note content has been drawn. The prototype's default $Text is only inherited _once_ from the prototype, if deleted in the inheriting note, it is not re-applied. If the default text is not of help to a user, it can also be deleted at source, i.e. in the prototype itself.
Note: newly added HTML prototype notes now offer a dozen tab stops; existing older HTML prototypes are not affected, so $Tabs may need to be updated manually if desired.

Auto-set Prototype: Imported from DEVONthink

Imported from DEVONthink

Applied to any note generated by a DEVONthink watched group. DEVONthink import uses a dedicated built-in prototype (as with watched DEVONthink folders), rather than setting Displayed Attributes directly. This allows customisation of Tinderbox's behaviour when importing from DEVONthink v3.

This prototype's non-default customisations other than $IsPrototype, $Name, and intrinsic map features (height, width, etc.) are listed below.

$DisplayedAttributes: "URL;SourceURL;SourceCreated;NotesModified;Tags;ReadOnly"

$Color: "lighter warm gray dark".

$LineSpacing: 120.

$ReadOnly: `true` (ticked).

$TextBackgroundColor: "white".

The use of a $TextBackgroundColor of white is because even if using dark styles in Tinderbox, external documents' styling is most likely compatible with light background colours.

Auto-set Prototype: Imported from Finder

Imported from Finder

Applied to any note generated by a Finder watched folder.

This prototype's non-default customisations other than $IsPrototype, $Name, and intrinsic map features (height, width, etc.) are listed below.

$DisplayedAttributes: "File;NotesModified;LastFetched;ReadOnly;Tags";

$Color: "lighter warm gray dark".

$TextFont: "IdealSansSSm-Book".

$LineSpacing: 120.

$ReadOnly: `true` (ticked).

$TextBackgroundColor: "white".

The use of a $TextBackgroundColor of white is because even if using dark styles in Tinderbox, external documents' styling is most likely compatible with light background colours.

Auto-set Prototype: Imported from Notes

Imported from Notes

Applied to any note generated by a Notes watched folder.

This prototype's non-default customisations other than $IsPrototype, $Name, and intrinsic map features (height, width, etc.) are listed below.

$DisplayedAttributes: "NotesID;NotesModified;LastFetched;ReadOnly".

$Color: "lighter warm gray dark".

$TextFont: "IdealSansSSm-Book".

$LineSpacing: 120.

$ReadOnly: `true` (ticked).

$TextBackgroundColor: "white".

The use of a $TextBackgroundColor of white is because even if using dark styles in Tinderbox, external documents' styling is most likely compatible with light background colours.

Auto-set Prototype: Imported from Tot

Imported from Tot

Applied to any note generated by watching the Tot application. This sets $Tot and $ReadOnly as Displayed Attributes.

This prototype's non-default customisations other than $IsPrototype, $Name, and intrinsic map features (height, width, etc.) are listed below.

$DisplayedAttributes: "Tags;ReadOnly".

$Color: "lighter warm gray dark".

$TextFont: "RingsideCondensedSSm-Book".

$LineSpacing: 120.

$ReadOnly: `true` (ticked)

Prototype: Markdown

Markdown

Allows notes to use Markdown, for local preview or export, without further fiddling with scripts post-export. The prototype presets $HTMLPreviewCommand to use a Markdown distribution inside the Tinderbox app although other builds or different render scrips can be user-configured. $HTMLParagraphStart and $HTMLParagraphEnd to "" (i.e. no code string) in order to eliminate unwanted paragraph markup tags. $Ziplinks is set to *false*.

This prototype's non-default customisations other than $IsPrototype, $Name, and intrinsic map features (height, width, etc.) are listed below.

$Color: "lighter black".

$HTMLBoldEnd: "" (empty string).

$HTMLBoldStart: "" (empty string).

$HTMLItalicEnd: "" (empty string).

$HTMLItalicStart: "" (empty string).

$HTMLMarkdown: `true` (ticked).

$HTMLMarkupText: `false` (un-ticked).

$HTMLParagraphEnd: "" (empty string).

$HTMLParagraphStart: "" (empty string).

$HTMLPreviewCommand: "CommonMark".

$LineSpacing: 120.

$ParagraphSpacing: "4".

$Tabs: "0.375;0.375;0.375;0.375;0.375;0.375;0.375;0.375;".

$TextFont: "IdealSansSSm-Book".

Prototype: Person

Person

The main items here are to add some of the 'People' group of system attributes as Displayed Attributes.

This prototype's non-default customisations other than $IsPrototype, $Name, and intrinsic map features (height, width, etc.) are listed below.

$DisplayedAttributes: "FullName;Organization;Address;Email;Telephone;Twitter;URL;NLNames". (pre-v7 versions also included the now defunct $AIM).

$Badge: "person".

$Color: "green".

$Pattern: "" (empty string).

$Shape: "lozenge".

$HoverExpression: `$Organization+"\r"+$Email+" "+$Telephone`.

Prototype: Place

Place

Useful for notes holding geographical location data.

v9.5.0 adds a badge to the prototype.

This prototype's non-default customisations other than $IsPrototype, $Name, and intrinsic map features (height, width, etc.) are listed below.

$Badge = "pin"

$DisplayedAttributes: "Address;NLPlaces;Latitude;Longitude".

$Color: "5".

$Opacity: 65.

$Shape: "oval".

Prototype: Poster

Poster

Useful for notes that will be used as a poster, especially adding poster-related attributeas as Displayed Attributes.

This prototype's non-default customisations other than $IsPrototype, $Name, and intrinsic map features (height, width, etc.) are listed below.

$Color: "green".

$DisplayedAttributes: "PosterTemplate;PosterURL;"
$HTMLDontExport: `true` (ticked).
$HTMLFirstParagraphEnd: "" (empty string).
$HTMLFirstParagraphStart: "" (empty string).
$HTMLIndentedParagraphEnd: "" (empty string).
$HTMLIndentedParagraphStart: "" (empty string).
$HTMLParagraphEnd: "" (empty string).
$HTMLParagraphStart: "" (empty string).
$NoSpelling: `true` (ticked).
$ParagraphSpacing: 0.
$Pattern: "" (blank - no value)
$ScreenHeight: 23
$ScreenWidth: 606
$SmartQuotes: `false` (un-ticked).
$Tabs: "0.375;0.375;0.375;0.375;0.375;0.375;0.375;0.375;".
$Width: 4 (note $Height remains default
$Ziplinks: `false` (un-ticked).

Prototype: Reference

Reference

If not already present this prototype is auto-set via the app when dragging in reference data to Tinderbox.

Initially implemented to support Bookends integration. Useful when adding structured reference data, e.g. via RIS data import. The includes $URL as well as $ReferenceURL. $URL holds a the URL to the reference in the reference manager (e.g. local pseudo-protocol `bookends://…`), whilst $ReferenceURL holds the reference URL for the reference source (if it has one).

This prototype's non-default customisations other than $IsPrototype, $Name, and intrinsic map features (height, width, etc.) are listed below.

$DisplayedAttributes (mainly from the References group of system attributes): " Authors;ArticleTitle;BookTitle;Journal;Publisher;PublicationCity;PublicationYear;RefKeywords;Abstract;URL;ReferenceURL".
$DominantLanguage: "en".
$Color: "muted green".
$AccentColor: "dark green".
$Pattern: "gradient".

Prototype: Task

Task

Useful for notes in to-do workflows and the like.

This prototype's non-default customisations other than $IsPrototype, $Name, and intrinsic map features (height, width, etc.) are listed below.

$DisplayedAttributes: "DueDate;StartDate;EndDate;Checked.
$Badge: "calendar".
$Color: "cool gray".
$AccentColor: "dark cool gray".
$Border: "dark cool gray".
$Shape: "tag".
$DisplayExpression displays the number of child items (if any):

```
if($ChildCount){$Name+" ("+$ChildCount+")"}else{$Name}
```

$OnAdd: `Prototype|="Task"`.

Closing pop-overs

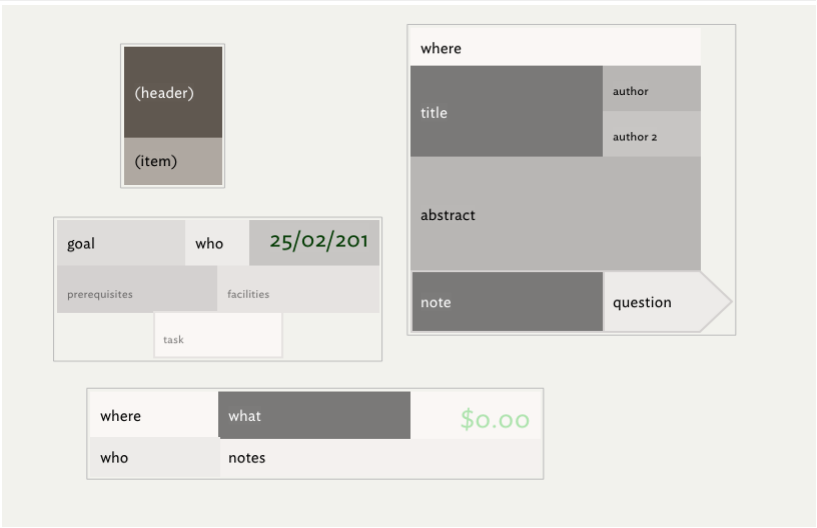
Many of the UI elements that were previously dialogs are shown as pop-overs with no obvious close mechanism. The expected close method is for the user to click anywhere outside the pop-over. For pop-over's based on a selection, e.g. Info, do not click on the source selection, but rather *anywhere else* outside the pop-over.

A pop-over can also be closed using the **Escape** key (⎋) but, depending on the task at hand, this may also revert the last edit on the pop-over so use with caution—unless that is the intent!

Composites

A *composite* is a group of Tinderbox notes that work together to describe something larger than themselves. For example, when taking notes (in Map view) in a conference, an individual talk might be a composite of notes for the title, the speaker, the content of the talk, and action items requiring follow-up action. Each note has its own text and attributes, but it may sometimes be useful to treat the composite as an object – for example, to move all the notes in the composite to a new map location. Further examples of usage, as envisaged by Eastgate, include the following (giving in parentheses the items the composite might contain):

- shopping lists (a store, followed by one or more things you want to buy)
- books (author, title, reading notes, editorial notes on the review we will write)
- lesson plans (date, topic, assignment, homework, special requirements)
- restaurant visits for a reviewer (name, phone, hours, credit card info, who dined with you, what you ate, when the review is due)



Composites are a feature primarily intended for map view use and Tinderbox notes form a composite when their map icon touches another note. Composites make it possible to make compound notes which have individual identities but also are linked to their collaborators in a new (visual) way.

Some built-in composites can be created from the File menu. This creates a root level 'Composites' container. User-created blank composites can be added to the container. Any composite in this container can be used to create a new copy by choosing Create Composite from the Note menu or the map background's contextual menu.

When selected, composites are outlined with a darker and thicker bounding box. The composite name and edit widget (a little pen icon at top left of the bounding box) are also displayed when the composite is selected. To (re-)name the composite, click the pen symbol: a pop-up naming box appears over the windows text pane. Enter the desired name for the composite and press the Return key to set and save the new name.

Clicking on any note within a composite selects the composite; clicking on non-note space within the composite's bounding box does not select the composite. Cmd+click to select individual note(s) within a composite. To edit an individual note, or to see only its text and Displayed Attributes, the note must be individually selected. When a composite is selected the resize handles are not drawn on the individual constituent notes.

Edit ▸ Break Composite can be used to break up a composite into individual components. Individual notes can be removed from a composite by (Cmd+click) selecting it and dragging it away from the other notes. When using ⌘-drag to remove a note from a composite, the note is removed from the composite immediately. (Previously, it was removed from the composite at mouse-up. The new approach makes the effect of ⌘-drag more clear.)

When a composite is selected, the text pane displays the text from each member of the composite, as for multiple selections. Composites may have a name. When any item in a composite is selected, the name of the composite is shown above the composite, along with a control widget that allows you to rename the composite.

When a note joins a composite, its moves in the outline to become the younger sibling of the last member of the composite. Thus, all members of a composite are adjacent in outline order. The relative order of items within a composite should remain unchanged.

Further composite features:

- Composite masters
- Roles for composite items
- Join actions
- Suppressing formation of composites
- Composite action codes

Composite masters

Adding any built-in composite creates a root level 'Composites' container. User-created blank composites can also be added to the container. *Any* composite in this container can be used as a 'master' to create a new copy.

To create a copy from a master, use the Create Composite option from the Note menu or the map view background's contextual menu. Note that composites with no composite name are not included in the pop-up lists of possible available composites.

Roles for composite items

Items in a composite may have a $Role, which indicates what part they play in a composite. For example, a composite that describes a research paper might include notes with roles like Author, Abstract, and References. If a note has a $Role but no $Subtitle, the Role is displayed where the Subtitle would normally appear.

Some kinds of composites may have multiple notes with the same role. For example, a list with a header would have one note with the role *Header*, but might have any number of notes with the role *ListItem*. The boolean attribute $IsMultip indicates that a note with that role may occur more than once in a given composite. If $IsMultiple is `true`, when a note is moved into the composite touching that note, the moved note inherits the note's role. Thus, items moved into a composite and touching a *ListItem* will themselves become a *ListItem*.

Join actions

An optional action may be performed when new composite members first touch existing members. This action is stored in the $OnJoin action. When a note that was not previously a member of a composite is dragged to touch a note with a $OnJoin action, the $OnJoin action is performed on the dragged note. If a note touches more than one note in the composite, each $OnJoin action is performed in turn. When an $OnJoin action is run, the 'this' designator is bound (refers to) t the note newly *joining* the composite and 'that' is bound to the *existing* composite note running the $OnJoin action.

Suppressing formation of composites

A $NeverComposite Boolean can be used to mark a note that should never be added to composites. By default `false`, if set to true at document level, a document will suppress composite behaviour in maps altogether.

When opening legacy files created in previous versions, Tinderbox scans the file in search of overlapping notes that will be interpreted as composites. This situation most often arises in documents that are seldom if ever used with maps. If Tinderbox finds apparently accidental composites, it marks the notes in the container as $NeverComposite.

Composite action codes

Actions and queries can use information about composites (see a listing of such actions). All such date is assumed to be read-only unless specified. Documentation references to 'theNode' mean the note in focus. The path list returned by compositeFor() or compositeWithName() may be used as a designator, or to obtain additional information about the composite.

Container Sorting and Transforms

Containers, agents and smart adornments may be sorted on an attribute's value, i.e. other than in default order, and additionally apply a transform before sorting string attributes. In a normal container the default order will be static and the order in which its children were created (or subsequently manually re-ordered). For agents and smart adornments, the default is less obvious as it depends on the complexity of the query; for trivial examples it *usually* equates to outline order.

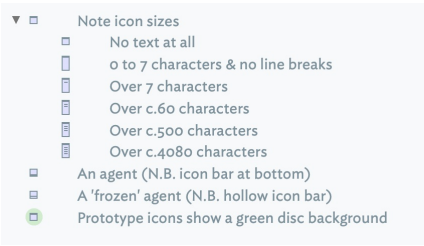
The sort attribute and transform type are selected from pop-up menus on the Sort sub-tab of the Action Inspector.

Containers also re-sort after:

- changing a Displayed Attribute's value
- changing an attribute value in Quickstamp
- changing an attribute value in the attributes pane of the Get Info popover
- editing a column value will update sorting
- changing the sort criteria in the Action Inspector

This re-sort trigger matters most when automatic agent updates are disabled.

Content and Type Dependent Icons



Depending on the presence and amount of content in a note, the outline view (and views with the same icon) will show a slightly different icon (and in Attribute Browser, chart and Timeline views).

- No note text. A simple half-height rectangle.
- Zero to 7 characters. A vertical rectangle with no horizontal bar inside.
- More than 7 characters. A vertical rectangle with a single horizontal bar inside.
- More than about 60 characters. A vertical rectangle with two horizontal bars inside.
- More than about 500 characters. A vertical rectangle with three horizontal bars inside.
- More than about 4080 characters of text. A a vertical rectangle with four horizontal bars inside.

In Map view (and views with the same icon), the presence of note content is shown by a 'dog-ear' icon on the note; the above granularity of content is not represented.

Agent icons are similar except the thick bar is at the bottom, not the top. Disabled agent ('frozen' via $AgentPriority) have an empty bar at the bottom of the icon.

The varying agent/note icon is not immediately obvious to the eye in Outline view, but once you realise it is there the difference it becomes more intuitive as a way to tell apart Notes and Agents. Maps have a different, but similar, visual layc to differentiate note vs. agent containers.

The fill colour of the icon, indicates the age of its content.

Prototype notes also show a light green coloured circle behind the icon (here darkened slightly from default value for better contrast). The colour used is set in the Map Preferences.

Copying or Moving notes within Tinderbox

Working within a single view

- Drag. Always moves the selected note(s). There is no key override to force a copy. Retains all formatting except links, though link mark-up (coloured text) is retained. Option drag of a *single* object duplicates the object and drags the duplicate. Option-Shift drag of a *single* object makes an alias of the object and drags the alias.
- Copy/paste. Retains all formatting and links
- Cut/paste. Retains all formatting except links/link mark-up.

Working within a single TBX file

- Copy/paste. Retains all formatting and links
- Cut/paste. Retains all formatting except links/link mark-up. Thus, to move text *and* retaining its links you must copy/paste before returning to delete the source text.
- Drag. There is no support for dragging objects between document windows.
- *See notes below about intrinsic attributes and prototypes* .

Working across different TBX files

- Copy/paste or cut/paste. Retains all local formatting except for links: only those links between notes in the pasted selection are retained.
- Drag. There is no support for dragging objects between documents.
- *See notes below about user and intrinsic attributes, and prototypes* .

Intrinsic attribute values

The values of intrinsic attributes are not maintained when copied within or between documents, because the value is intrinsic to the source note. If it is important to capture and copy the source note's exact value for an intrinsic attribute, consider caching the value in a user attribute (or one of the System's 'Sandbox' group).

Copying between documents and User attributes (and their values)

If a note is copied and pasted to a new document, only those attributes existing in both documents are are fully transferred. Thus it is important to implement any necessary user attributes in the destination TBX before the actual copying of notes. N.B. $DisplayedAttributes holds the names of notes as text strings so a pasted note's Displayed Attributes may continue to list attributes from the source document even though they do not (yet) exist in the current document.

Duplication and links

If a note with links is duplicated, any outbound links are duplicated, none of the inbound links. All inbound links remain linked only to the source of the duplicate note.

Prototypes

Duplicating prototype notes. (see also) Duplicated notes retain the source note's prototype ($Prototype). For notes that are prototypes themselves, if Note A is a prototype for note B, then duplicating A makes a new prototype note but the duplicate no longer has a prototype link to B (i.e. B retains inheritance from A). The duplicate does have copies of other non-prototype outbound links for A, it is only A's prototype links that are not duplicated. Copying and pasting a container within a document results in the notes in the pasted container using the same prototypes as notes in the source container.

Prototypes between documents. If a note or agent is copied and pasted to a different TBX document, the new item does not inherit its source's $Prototype value, unless the source note's source prototype is included in the selection. This i a defensive assumption by Tinderbox as it cannot be sure pasted notes are from the current TBX and thus that all source prototypes exist. If it is intended some/all pasted notes will use prototypes from within the destination document, these assignment must be made manually (or via agents, etc.) after the notes have been pasted into the destination document.

Copying or Moving text within Tinderbox

Working within a single note window

- Drag. Always moves selection. There is no key override to force a copy. Retains all formatting except links, though link mark-up (coloured text) is retained. If drag appears not to work, press and hold for a short pause before beginning the drag.
- Copy/paste. Retains all formatting including links.
- Cut/paste. Retains all formatting including links.

Working within a single TBX file

- Drag. Retains all formatting except links.
- Copy/paste. Retains all formatting including links.
- Cut/paste. Retains all formatting including links.

Working across different TBX files

- Drag. Retains all formatting except links.
- Copy/paste. Retains all formatting and links.
- Cut/paste. Retains all formatting except links.

Creating draft emails

From v9.6.0, Email-type attributes show an envelope icon button in Displayed Attribute tables and Get Info/attributes. Pressing this button creates a new draft email message using the host Mac's current email client

The email message is created and opened (*but not sent*) with the following already filled in:

- 'To' addressee.
- 'Subject' is optionally set via $EmailSubject.
- Styled $Text is added as the body copy of the message.

Adding a subject and any CC addressees, etc., is left for the user to edit before they send the email.

This feature is not design to auto-send complete messages but to pre-populate predictable parts of a draft email to make it easier to generate email from Tinderbox data.

Using an email Template

It is possible to generate more complex email content using an email export template ($EmailTemplate). The latter uses standard export template code so can use not only $Text but content from other attributes (and/or from other note(s).

If the current note's $EmailTemplate specifies a valid template, that template is applied to the current note and the result of that export is used to set the body copy of the draft email message.

Creating Footnote notes

The Footnote sub-menu of the Note menu provides method to create a new note from a selected word or phrase, with the intent that it acts as a reference to the source text. In one combined action, Footnote creation:

- creates a new note
- creates a text link, of link type "note", from the selected text to the new note
- creates a link, of link type "Note +", from the new note back to the selected text
- the current selected text becomes the name of the new footnote.

Depending on which footnote creation option is chosen the location of the new note is either as a:

- sibling: next sibling to the source note
- child: a new sibling container 'Notes' is created and the footnote is a child of this container. All child footnotes of sibling notes share the same 'Notes' container. Different containers use discrete 'Notes' containers for their footnotes.

It might seem odd to have this placement choice and 'child' might seem the generally logical choice. In an outline, a container can easily be expanded/collapsed to show footnotes. However, if working in map view and desiring to keep everything in view on the same map, the 'sibling' option avoids having to find child footnotes and drag them up to their parent's level just so they show up on the map.

The different link types used for links to/from the footnote make it easier to identify notes with footnotes or footnotes themselves.

Although a possible end point for Tinderbox footnotes are article footnotes when data is exported to other formats, there is no method for collecting and inserting them at the end of an article; this must be done by the user's export template design. Using the 'child' option does however place all footnote from sibling notes into a single container, aiding export.

An alternative way to create new notes from source text is to simply drag a text selection to an Outline or Map view.

Current attribute: shared by Quickstamp and Get Info

The current attribute group and current attribute are remembered (and shared) by Quickstamp and Get Info.

Default note colours & patterns

Defaults are:

- $Color: '7'
- $AccentColor (was $Color2): 'dark warm gray dark'
- $Pattern: 'plain'.

Note that for adornments, $Colour is 'normal' as adornment use as their fill colour a tint of the normal default $Color value.

Display Expressions

Tinderbox can use calculated screen titles, based on the $DisplayExpression attribute. The output of this (single) expression is displayed as $DisplayName. For each note, Tinderbox evaluates that note's $DisplayExpression to create the note's name.

Be aware that if $DisplayExpression is empty, or not a valid action syntax expression, or the expression returns no value (such as an an empty string ""), then Tinderbox will instead display the note's $Name value as it will assume you want some form of title displayed.

Although the display expression may include conditional logic (see below) generally it is a good idea to avoid complex calculation within the expression. If the display expression needs to use pre-calculated values that are not provided by system attributes, then a rule (or other such action method) should be used to calculate the value and place it in a user attribute which the display expression can then evaluate.

Example rule:

```
$Name +" (" + $WordCount + ")"
```

…will display the note's word count in brackets at the end of the note. Thus, "Trip Report" becomes "Trip Report (579)". $Name will continue to return "Trip Report".

This attribute is of the 'action' data type. It takes a string argument which must be valid action syntax.

Subtracting two dates returns the number of days between two times, even if the result is a string such as $DisplayExpression. For example:

```
$Name+": "+($DueDate-date("today"))
```

will generate strings like: "Buy groceries: 7"

Note the parenthesis around the second part of the expression; if omitted, the expression would be interpreted as an attempt to subtract two strings:

```
($Name+": "+$DueDate)-(date("today")
```

If just including a date attribute in an expression it is possible to confuse Tinderbox's parser. Assume $DemoDate for note "Project X" is 28/07/2009 17:57 (on a system using day/month order dates). Consider the following display expressions

```
$DemoDate
```

gives the string: '28/07/2009 17:57'

```
$Name + " : " + $DemoDate
```

gives the number string: '3313478996'. This is because Tinderbox assumes the user is trying to do some form of date arithmetic. The way around this is to use the format(data,formatString) operator. The latter emits a string, telling Tinderbox that all it has to do is concatenate the strings. Thus:

```
$Name + " : " + $DemoDate.format("D/M0/y h:mm")
```

gives the string: 'Project X : 28/07/2009 17:57'.

Setting $DisplayExpression's code via action code

Another context where care is needed is if setting the $DisplayExpression of another note, e.g. via an $OnAdd action. In this cases it is important to ensure the application understands the result is a string. A good example is when the code returns something that might be a sum, like a date in "20/5/2010" form: The following is not good $OnAdd code:

```
$DisplayExpression=$Created
```
WRONG

```
$DisplayExpression=$Created.format("l")
```
WRONG

Although both the above right side values can safely be added into a Text Inspector Title sub-tab's Display Expression box, they do not work for code-based assignments. Instead, it is necessary to force a string:

```
$DisplayExpression='""+$Created+'"'
```
GOOD

```
$DisplayExpression='$Created.format("l")'
```
GOOD

Notice, in both how single quotes are needed. In the first case they enclose each instance single instance of a double quote (quote characters *cannot be escaped* in Tinderbox). In the second case they enclose the whole string as it already contains double-quotes surrounding the date format string.

Conditional Expressions in $DisplayExpression and complex expressions

Display expressions also accept conditional expressions, i.e. using ' if' statements. For example, in a TBX listing books there might be agents listing & sorting by ISBN and by Author. It would be useful if in each of the latter cases the matched note's alias showed the relevant attribute as part of the alias' screen title whilst using $Name in all other cases. Note that with a conditional expression we still need to define something for the 'no match' branch or else notes matching that

condition get a display expression of nothing, which results in 'untitled' being shown on screen. Here's the expression:

```
if($Name(parent)=="ISBN"){$ISBN+" - "+$Name}else{if($Name(parent)=="Author"){$Author+" - "+$Name}else{$Name}}
```

That is quite a long and complicated piece of code, due to the nesting and that's for only 3 conditions (match 'ISBN', match 'Author, no match). For longer pieces of code it can be useful to put the expression in a note, as the note's text. Conveniently, when doing this, line breaks in the text are ignored making it easy to split apart the various clauses in the expression to check nesting, etc. Assume the expression above has been places in a note called "c_BookDisplay". Now, for tour book notes (or their prototype!) you use this Display Expression:

```
eval($Text(c_BookDisplay))
```

The result is the same as putting the initial code into each note's $DisplayExpression except that now it is easier to maintain and edit.

Another approach can be to store the output of all the complex code as a user string attribute ('MyDisplayString') and use the latter as the display expression. Thus:

$Rule: `$MyDisplayString = if($Name(parent)=="ISBN"){$ISBN+" - "+$Name}else{if($Name(parent)=="Author"){$Author+" - "+$Name}else{$Name}}`

$DisplayExpression: `$MyDisplayString`

Displayed Attributes replace Key Attributes

The term 'Key Attributes' has been superseded by the term 'Displayed Attributes' as this better reflects their purpose.

All system attributes containing the string 'KeyAttributes' are deprecated in favour of new system attributes using the term 'DisplayedAttributes'.

Thus, attribute $KeyAttributes has been renamed $DisplayedAttributes. Related changes include $DisplayedAttributesFont, $DisplayedAttributesFontSize, $DisplayedAttributesDateFormat, and $HideDisplayedAttributes.

Existing documents may continue to refer to $KeyAttributes and will continue to operate.

Dragging notes between TBXs

A highly-experimental facility allows you to drag notes from a Tinderbox map, outline or chart view to Tinderbox map views in other Tinderbox windows. The destination document need not be in the same document as the source.

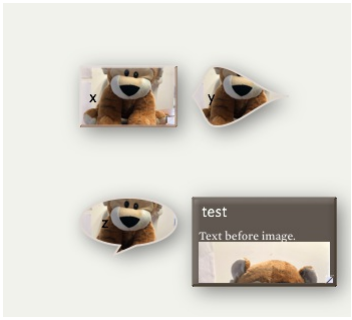
When dragging to a new window, Tinderbox will make a copy of the dragged note and will copy the dragged note's attributes to the copy.

Attributes not defined in the destination document will be ignored. If a note is dragged to another application, the note's text and styled text are available to that application.

Embedded image fills

If a note's $Text begins with an embedded image, the note's map icon image is drawn to fill the bounds of the note. This applies to all note shapes.

Any character, even just a space, will revert to the normal render; note the latter is only used in rectangular notes (i.e. the default shape).



Features needing more recent OS versions

Although this version of Tinderbox needs a minimum of macOS 0, some new features may require a more recent OS version in order to function. These include:

- Features with OS limit higher than app base specification
- Window menu
- Colors
- Exploding Notes
- Built-in Hints container
- Taggers
- Sentiment Analysis
- macOS Dark or Light modes
- String.paragraphList()
- String.wordList()
- String.nounList()
- favorites folder
- Find results pop-over
- Find toolbar (view pane)
- Find results stand-alone dialog
- wordsRelatedTo(dataStr[, wordsNum])

Finder Quicklook

Finder's quicklook feature shows a preview of the the view pane of the current tab, as last saved. Older documents will need to be re-saved once using Tinderbox v8+ in order to generate a quicklook image.

Focus View, Expand View

Views are, by default, shown at root level. However, focus can be shifted to focus, (root) at a different level. In map view, the focused element forms the map background (its container); in other views the focused item is the root of the view.

Focus View ('hoist')

In Outline, Chart or Treemap views, Focus View focuses the view on the currently selected note and its descendants. A focused view is subtly like a map except that in a map you cannot access the contents of the parent container. Conversely, in a focus view you can see/edit the container but you cannot add siblings to it; this is because the view's scope for adding/removing notes restricted entirely *within* the root notes descendants.

Thus, in a focused view, siblings cannot be added to to the root (top) note, only children/descendants.

Expand View ('drill down')

Expand View moves the current view 'up' one level. Although there is no menu item, the opposite keyboard short cut to this method achieves a 'drill down' one level if the current note is a container.

Navigating view focus

To navigate from a focused view you can move scope:

- Up (expand/hoist)
  - Click the desired item in the view pane breadcrumb bar (moves directly to that scope).
  - Map view: Up-arrow key (↑). Expand view one outline level.
  - View menu ▸ Expand View. Expand view one outline level.
  - Expand View shortcut.
- Down (focus/drill-down)
  - Treemap: Select and double-click a descendant. Focus view one outline level down.
  - Outline/Chart: double click a note's icon. Focus view one outline level down.
  - Map: double-click a container's viewport area (Map). Focus view one outline level down.
  - Map: 'Drill-down' shortcut. This is a useful trick in that it can drill down a level even if a container has its viewport hidden by expanding the title area.

Full Screen mode

Hovering over the green 'expand' button on any document window (but *not* secondary/stand-alone windows), shows the full screen toggle. Click when this is showing to enter the normal macOS full screen mode. It is not possible to use this for tear-off pop-up windows (e.g. Get Info, etc.) or for note text windows.

Full screen split view is supported. In normal (non-full) screen mode press and hold the green "full screen" button in the main window's upper left-hand corner; the window will shrink to permit the user to select the right or the left half of the screen.

Hoisting view on childless containers

It is possible to hoist focus to a childless container (or drill to the map of a childless note). This can be useful if the user needs to create a container and to then add notes to that container.

If a note is 'outdented' in a hoisted outline (i.e. raised in $OutlineDepth), so that it no longer falls inside the section of the outline being viewed, Tinderbox removes the note's view from the outline.

Hover Expressions

Hover Expressions are action code expressions, like Display Expressions, but which are displayed in a tooltip-like floating pane to the right of a map icon when the icon is moused over. The same hover display method is also used in Outline, Chart and Timeline views, with the hover panel displayed below the hovered-on item.

This expression allows extra information to be viewed without having to open the note. A trivial example might be to show the word count for a note's $Text. This feature is not intended to preview very large amounts of $Text but rather to do things like expose extra contextual information, much in the way Displayed Attributes and Display Expressions are used.

The expression is stored in attribute $HoverExpression and is a string of action code. It might be an attribute reference:

```
$WordCount
```

or a simple string:

```
"Do not forget to review this!"
```

or an expression:

```
"Word count: " + $WordCount
$Price * $NumUnits
```

Line breaks are allowed in strings using a the '\n' construct:

```
"Word count: " + $WordCount + "\n\n" +$Text
```

The expression's hover panel draws text in $HoverFont, with the font size controlled by $MapTextSize. The panel is always a translucent dark panel with white text. The panel is drawn 350 pixels wide with height never exceeding half the Map view's window height. The hover expression box uses rounded corners and is suppressed if the expression only returns white space (i.e. no visible text); also, the box is offset further right so as not to obscure $Name data.

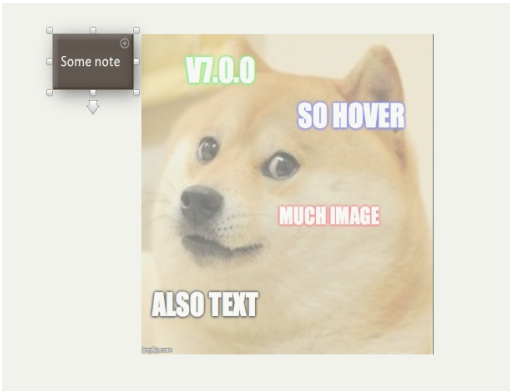
The hover panel can also include an image.

If a note has both a hover image *and* a hover expression, the hover expression is now displayed even if the hover image doesn't exist (i.e. the image source cannot be resolved or the image file is missing).

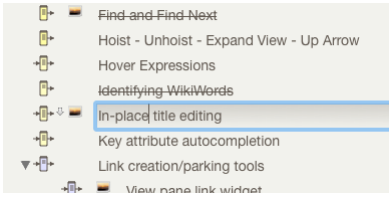
Hover Images

It is possible to display an image when hovering over a note. $HoverImage contains the path to an image file. The easiest way to set $HoverImage is to drag the file into the image area in the Hover inspector.

If a note has both Hover Expression and an image set, the text value of the $HoverExpression is displayed in large type at top of, and in front of the hover image.



In-place title editing



Otherwise referred to as 'Edit-in-Place' or 'E-I-P' for short. In any view pane view click-hold an (already selected) note to enable editing of the title. A click-pause of an already selected note will also trigger edit-in-place. In Outline view, column values can also be edited (assuming the attribute is not read-only).

Clicking in the title of a note in outline view opens an editor only if you do not choose to drag or double-click the note. This makes in-place editing feel natural, at the cost of a slight delay before the editor actually appears. The delay before the edit is invoked is the same as that used by Finder when invoking edit mode on filenames.

To finish editing the name, do any one of:

- click the note's icon
- select another note
- press the **Return** key

Cmd-Shift-Return enters edit mode for title of currently selected note.

Esc finishes editing and restores the title to its initial state.

Cmd+Z (Undo) finishes editing and restores the title to its initial state.

When adding a new note in edit-in-place mode, if the edit mode is exited without a title being created the new note is not created. There are two exceptions to this—adornments and separators—where no name might be a desirable and deliberate choice.

Up-Arrow. If the edited title has several lines, up-arrow moves the selection to the previous line. If the selection is at the beginning of the text, up-arrow selected the previous note.

Down-Arrow. If the edited title has several lines, down-arrow moves the selection to the next line. If the selection is at the end of the text, down-arrow selects the next note.

Tab. In map view tabbing from the title ($Name) field selects the subtitle ($Subtitle) field. Tabbing from the subtitle selects the caption ($Caption field. Tabbing from the caption selects the title field.

If the 'check spelling as your type' option is selected the title text is also checked for spelling, although to improve performance the check excludes words in the user's dictionary.

Data pasted into an edit-in-place box is treated as unformatted.

Subtitles. Subtitles can be edited in map view by clicking and holding on the subtitle, or on the blank space where it would go.

Entering title Edit-in-Place mode in Outline or Map view will cancel Hover displays and, in Map only, also Subtitles.

Support for Column view Editing

Clicking directly on a column *other than the title* ($Name) of the note promptly opens the in-place editor.

Double-clicking on a column *other than the title* ($Name) of the note also promptly opens the in-place editor.

In Outline view, when editing in place:

- **tab** moves the focus to edit the next column, if that column can be edited.
- If columns are not in use, **tab** moves the focus to the name of the following note.

When not editing in place, **tab** continues to indent the note.

Similarly:

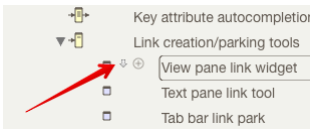
- **shift-tab** moves the focus to edit the previous column, if that column can be edited.
- If columns are not in use, **shift-tab** moves the focus to the name of the preceding note.

Link creation and parking tools

Tinderbox offers a number of UI tools to assist with the creation of links within Tinderbox:

- View pane link widget
- Text pane link widget
- Tab bar link parking space

View pane link widget



In most view types used in the View pane, a link widget is displayed allowing a basic link to be dragged to another note in the view or the tab bar's link parking space.

Placement

The link widget is a small downward facing arrow, as shown here in an outline view. Placement depends on the view type and is only displayed for a selected item:

- Outline, Chart view: between the note's icon and its badge.
- Map view: under the selected icon.
- Timeline, Attribute Browser views: to the right of the note title.
- Treemap view: not currently supported.

Using the widget

To drag a basic link, click on the widget and drag:

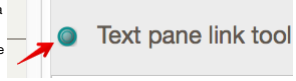
- *destination item is visible in current view* : release the drag onto the destination note. The Create Link pop-over will then be shown to complete configuration of the link. If the wrong destination has been selected, press the Escape key t cancel the process. The drag must be to a note currently accessible, i.e. currently displayed in the view pane.
- *destination item is not visible in the current view* : drag the link onto the tab bar's link parking space and use the latter to complete the process.

If more than one item is selected, using the link widget will create a single link from the note whose widget was used and ignoring the other notes in the selection; the selection is unaffected.

Links cannot be dragged between different document windows, i.e. outside the current window.

Links cannot be dragged to, or from, Treemap view items (though links can be originated from the text pane instead).

Text pane link widget



The Text view's header section shows a circle looking like a link parking space, to the left of the note title. In fact it functions like the link widget seen in the view pane. It permanently holds a basic link from the current note or, if there is a selection in text, a text link.

The widget is style similarly to the tab-bar link park. The link parking space shows a green fill by default. If used in this condition, the link dragged is a basic link from the current note. If there is a selection in the note's $Text, the tool shows a superimposed white 'T'. If used in this condition the link dragged is a text link. The widget only shows as empty if no note is currently selected.

Making a link via dragging

Click on the link tool and drag:

- *destination item is visible in current view* : release the drag onto the destination note. The Create Link pop-over will then be shown to complete configuration of the link. If the wrong destination has been selected, press the Escape key t cancel the process.
- *destination item is not visible in the current view* : drag the link onto the tab bar's link park and use the latter to complete the process.

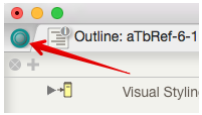
Links cannot be dragged between different document windows, i.e. outside the current window.

Links cannot be dragged to, or from, Treemap view items.

Using the pop-up

Click on the link parking space. Use the pop-up dialog presented to enter the destination note's title, after which the Create Link pop-over will be shown. Use the Escape key to cancel if details are not correct. The pop-up retains, for the session, the name of the note last successfully linked to via the pop-up (as opposed to using drag-drop completion)

Tab bar link parking space



The link park is the green circle at the left end of the current window's tab bar. As such the tool is accessible from any tab in the current window. The purpose of the tool is to allow the user to 'park' links that cannot be completed within the current view. The view, or tab, focus can then be changed allowing the link creation to be completed by dragging the link from the parking tool onto the destination note.

The type of link created, basic or text, will depend on the link currently parked. As such, the tool will show in one of 3 conditions:

- Empty. The default when a document is opened, either when new or for a new editing session. In this condition the tool is an empty green ring.
- Basic link is parked (illustrated). The tool has a solid green fill.
- Text link is parked. The tool has a solid green fill with a superimposed white 'T'

Links are parked here using either the view pane link widgets or the text pane link widget.

Links cannot be dragged between different document windows, i.e. outside the current window or between documents. Each document window, if more than one, supports its own discrete link park, i.e. the link parked in window #1 is not available in window #2 and vice versa. Links cannot be dragged to, or from, Treemap view items

Links can be created via one of two methods: drag/drop or clicking the tool and entering a destination via the link parking space's pop-over.

Dragging to complete a parked link

Ensure the destination item is visible in the current view pane. Click on the link park and drag onto the destination note. On drop, the Create Link pop-over will open to allow the link to be configured. If the wrong destination was selected simply press the Escape key (⎋) to cancel the link and start a new drag.

Using the pop-up to complete a link

Click on the link park. Use the pop-over dialog presented to enter the destination note's title, after which the Create Link pop-over will be shown. Use the Escape key to cancel if details are not correct.

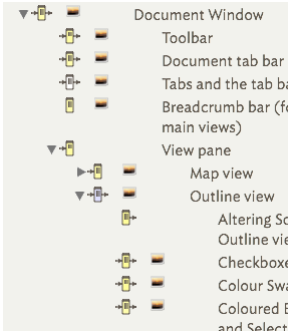
Replacing the parked link

Dragging a link to the link park does not create a link from the source note *until* the parked link is completed via one of the methods described below. Thus a parked link can be used to create one, or more, links as required during a session.

The link park only ever stores the stub of one link. To store a different link, simply drag from the new source onto the tool. The newly stored link replaces the previous one. There is no need to delete a stored link; either replace it or leave it. At the end of the current session sorted links are not retained. The link park is always empty at the start of a new session.

> *Note for users of previous Tinderbox versions* : this tool works in exactly the same manner as the three link parks on the older app's toolbar.

Link Indication in Note Icon



Note icons show incoming and/or outgoing stub arrows to indicate a not has inbound or outbound links present.

This feature is shared by other views using the Outline-style note icon: Attribute Browser, Chart and Timeline.

macOS Dark or Light modes

The main UI supports the macOS' toggled light/dark modes, introduced in macOS 10.14 (Mojave).

New documents created in OS Dark mode will default to using a use a dark Tinderbox built-in colour scheme 'Dark Coral'. The default colour scheme for documents created in OS Light Mode is the built-in scheme 'Modern'.

These built-in schemes are listed on, and can applied via, the Colors tab of Document Settings (Edit menu or [Cmd]+8 shortcut)

If it is desired to have a 'light' general background to the view pane and $Text area whilst in OS Dark Mode, apply the 'Tinderbox 7' scheme. The KA table and general screen 'chrome' (tabs, etc.) of the app will remain dark but these two key parts of the window will take a light-on-dark colour style.

Likewise, if opening documents that were made in older OS versions or older Tinderbox versions and the colours look wrong, simply apply 'Tinderbox 7'. This will update the document's colour scheme to reflect the additional colour setting changes affected by OS Dark/Light mode.

When changing to dark mode or light mode with Tinderbox open, the text colour of the tabs is updated immediately.

Message placards

Action code operators show() and String.show() allow the user to display a simple textual message in the front document window.

The message placard

The message is displayed in a placard that rises from the bottom of the window and stretches across both panes. The area shown makes the message harder to miss but does not imply it is designed for large amounts of text.

Messages are unformatted text characters only

Although a message can be passed as an attribute value or a variable, the message must consist of text characters. Images cannot be displayed, nor can the message—once passed—be code that needs further evaluating. In the latter case evaluate the expression *before* passing the message.

The 'text' is unicode, so it does allow for use of non-Roman scripts (e.g. Cyrillic, Japanese, Chinese, etc.—assuming suitable fonts are present), as well as emoji and character-encoded symbols.

As text is rendered plain, any source text styling (bolding, etc.) is not retained in the message rendering.

(Maximum) amount of text shown

This feature is designed for simple short messages. Text will flow across the whole width of the placard. Line breaks are honoured. The placard will show up to 4 lines of text, and the message cannot be scrolled to show additional content. Messages should thus be designed with this in mind.

Message duration

A message sent to the placard will be displayed for 5 seconds after which the placard closes. If a new message is issued before the first message is complete, the new message immediately replaces the existing message and starts a new 5 second clock.

Message queue

If a new message is sent while a message is already being displayed, the new message will be shown after the previous message is hidden. Thus there is an implicit queue, with each queue item being shown for 5 seconds until the queue is empty. The placard is lowered between successive messages which helps signal the change of message (even if the colours are not changed).

Placard background and text colour

Both message methods have a pair of optional arguments, **backgroundColor** and **textColor** to set either/both the colour of the placard and the message's text colour. The text default colour is black but this may provide insufficient contrast a custom background colour is applied. Either colour can be specified as a string conforming to allowed colour specification methods.

The **backgroundColor** argument may be used on its own, with default black text:

```
show("Hello World","blue");
```

but if **textColor** is specified, a **backgroundColor** value *must* be passed:

```
show("Hello World","blue");
show("Hello World","blue", "#FFF');
```

Do not try this

```
show("Hello World",,"blue");
```
WRONG! Do not use

as the result is an undefined background colour (which is normally a shade of orange brown)

Considerations for use vs. logging messages

An alternative method of feedback is to use a note's $Text, as a log, and append to the content there. Bear in mind:

- placard messages are not saved. After 5 seconds of display any message is gone and not retrievable.
- if a message is long, consider if the fixed duration of 5 seconds is enough to read the message before it clears.

Naming of duplicated notes

Originally, duplication of a note such as 'Note A' resulted in names like 'Note A copy', 'Note A copy 2', etc. The current note duplication is somewhat more considerate in choosing a name for the duplicated note. In particular, if the note name already ends in a number, that number is incremented rather than appending the word 'copy'. Thus duplicating 'Note 1', creates 'Note 2', 'Note 3', etc.

Navigating via links

How to navigate from the current note using outbound (Tinderbox) links? This depends on the link type

Using keyboard shortcuts

Pressing Cmd+Return (⌘+↩), note *but not Cmd+Enter*, will navigate the first Basic Link (if present). The first link is defined as the oldest (by creation date) basic-type link for the current note.

It is possible to 'Go back', i.e. navigate backwards along the last traversed link, using Cmd+single-quote (⌘+'). In the case of the Go Back command Tinderbox will navigate the last-used link regardless of whether a Basic or Text type link.

Navigating Basic Links

Selecting and following Basic type links, especially if not the oldest link, is done using either of the following:

- text pane's Links panel
- Roadmap view pop-over, or torn-off Roadmap window

In either case, select a target note from the column of outbound links and double-click it. The target note will be selected and focused in the text pane; see below for whether the target note is shown in the view pane.

Navigating Text Links

Simply click on the link anchor. The target note will be selected and focused in the text pane; see below for whether the target note is shown in the view pane.

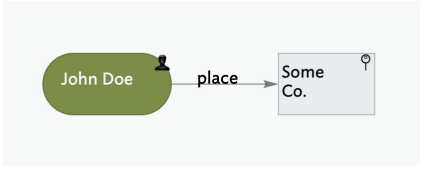
Displaying anchor text. By default, text link anchor text is coloured. Anchor text can also be temporarily underlined by pressing Cmd+Opt (⌘+⌥). Both these behaviours can be turned off via a document preference option.

Is the newly-selected target note shown in the view pane?

This depends on the view type:

- Map view. If the target note is on the current map, the map view will scroll to make the note visible.
- Outline/Chart views. In hoisted views (i.e. those views rooted on a lower level container), selecting a target outside current view scope will un-hoist the view to place the target in scope. If the target note is in a collapsed branch the branch is not expanded (though the note within it is selected).
- Other views. Behaviour will vary - there is no formal documentation of what should happen.

New note name parsing for prototypes and locations



When naming new notes it is possible to auto-add prototypes and locational data, borrowing from additional new v9.5.0 syntax for the zip-method of text link creation.

Parsing for this special markup can be disabled in Document Settings ▸ General as in some cases use of # and @ are deliberate and literal, and not mark-up for the features below.

When the setting is active parsing occurs when a new note is made or a title is modified.

Setting a prototype

Use a hash character and a prototype name after the new note name to set a prototype for the new note. For example, typing a new note title:

```
John Brown#Person
```

makes a new note with $Name "John Brown" and sets its prototype ($Prototype) to "Person". If the named prototype exists in the document it is used automatically. If the prototype doesn't name a new, un-customised note of the the specified name is add to the `/Prototypes` container. Thus:

```
John Brown#Team X
```

would generate and set a new prototype called "Team X".

When creating a new prototype, Tinderbox will set the Displayed Attributes to $Tags and add the prototype's name to $Tags. This aids rapidly finding notes after fast data entry.

Setting a (place) location

Use an '@' character and a location name after the new note name to set a location for the new note. For example, typing a new note title:

```
John Brown@BigCo
```

makes a new note with $Name "John Brown" and links to the note named "BigCo", creating the latter if it doesn't exist. The new note is linked to the (new) location note using the (new) link type 'place'. If a now location note is created it sets the built-in prototype 'Place', adding the latter if not already present in the document.

Be aware that for new locations, the address and latitude/longitude will need to be added manually by the user.

In map view, if using @place with or without #prototype notation in the root-level map (i.e. the default map in a new TBX) Tinderbox may to first create the 'Prototypes' container. If so, this is placed above the current note to leave space for the place note; normally new notes are created to the right of the current note but as the Prototypes container is more of a structural folder is is helpful that it is not inserted between actual notes. If the container already exists (i.e. there are already prototypes) or the map is at any other level than root, no difference is seen.

Using both options together

Make a new note and include both the #-based and @-based modifiers described above. The effect is as shown in the screen grab here, at top of the page.

Improved Parsing

From v9.5.2, the parsing of tags and prototypes for brainstorming is smarter about a variety of special cases. Notably, if the first character after a # is a digit, Tinderbox assumes it is an expression like "Activity #3" and does not create a prototype. Also, email addresses in titles (mark@example.com) are not treated as introducing a new place (location).

Non-editable notes

To avoid accidental editing it may be desired to make a note 'read-only', i.e. non-editable, but *there is no single setting that allows this* .

However there are two system attributes that cover aspects of this. Both date from the early days of Tinderbox before current heavier use of action code and so address only UI interactions.

Here, 'note' applies also to adornments, aliases, and agents.

ReadOnly attribute

The $ReadOnly system attribute affects only UI, i.e. manual, editing of the text pane, excepting the title bar. Thus it controls manual edit of:

- in the text pane:
  - the $Text area.
  - the Displayed Attributes table (if used by the note). If $ReadOnly is included in the table it is *always* editable.
- aliases of a note will honour the original note's setting (i.e. $ReadOnly is not intrinsic)
- use of Get Info, as a pop-up or as a stand-alone window

The original conception for $ReadOnly was to 'lock' the $Text of notes using AutoFetch to avoid the auto-imported $Text fighting against user manual edits. Later, scope was added to include Displayed Attributes. This explains the manual/UI-only nature of the setting.

A note's $ReadOnly has no effect on:

- $ReadOnly is *always* editable via any means
- manual editing in the UI of the view pane
- matching queries
- action code (including stamps and the Quickstamp Inspector).
- AppleScript or other inter-app/OS scripting

Though action code *may* be set to check a note's $ReadOnly state, this would need to be applied individually to *every* affected action. There is no centralised switch for such behaviour (beyond using prototype-inherited code).

$ReadOnly can be set via:

- Get Info ▸ Attributes
- Text Inspector, Text sub-tab.
- the Displayed Attributes table (if $ReadOnly is included as a Displayed Attribute)
- action code or external scripting of $ReadOnly

Lock attribute

The $Lock system attribute affects only the view pane and is was originally conceived for Map view use. It controls several view pane:

- Map position, i.e. $Xpos and $Ypos
- Map icon size, i.e. $Width and $Height
- Selecting locked notes:
  - Locked items cannot be drag-selected (i.e. dragged marquee/band-box selection)—locked items are always omitted from such selections.
  - Locked items can be selected by manual click. In map view (only) selected locked items show no icon resize handles. The only indication of a selected locked item is the presence of the link-drag arrow (in views that use that), and even then the arrow is only shown if a single item is selected.
- moving locked notes:
  - a locked note cannot be drag-repositioned in the view. This includes in other views, Outline, etc.
  - reflecting the fact $Lock was really designed for map use, the Note menu's Move Note Up/Down controls can be used to affect map icon stacking order.
  - similarly, the View ▸ Arrange menu and shortcuts do work. Alignment menu commands will override the $Lock status without changing the actual $Lock value and thus may alter note width, height and X/Y position values.

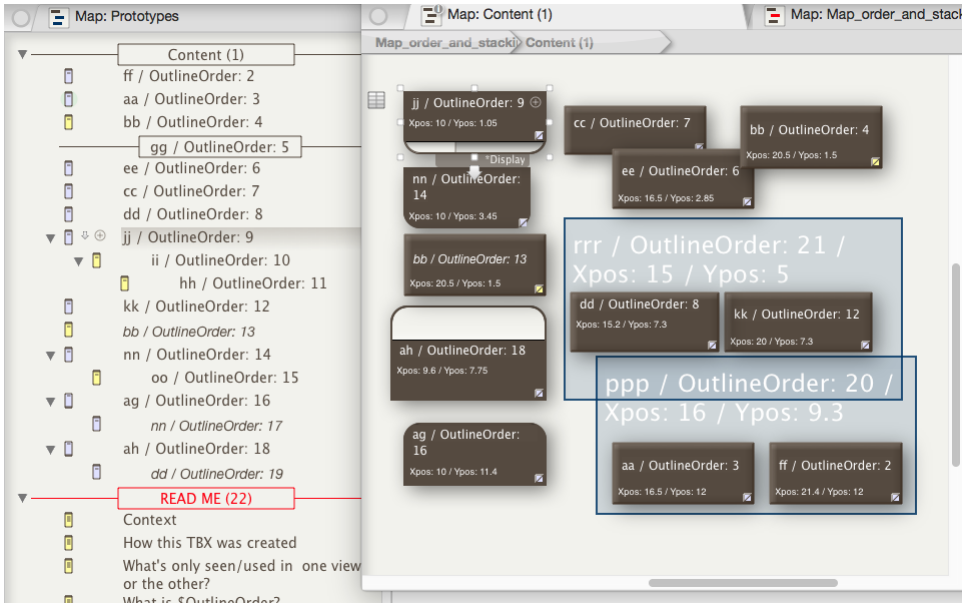
A note's $Lock has no effect on:

- manually editing title ($Name) subtitle ($Subtitle) or badge ($Badge) via the note icon in the *view* pane
- manual editing in the UI the *text* pane
- matching queries
- action code
- AppleScript or other inter-app/OS scripting

$Lock can be set via:

- for adornments only, when selected, via clicking the padlock icon. Note that the icon used for $Lock state can be customised.
- Get Info ▸ Attributes
- Properties Inspector, More text sub-tab.
- the Displayed Attributes table (if used as a Displayed Attribute, but only if $ReadOnly is `false`)
- action code or external scripting

Outline vs. Map Interface



Context

Newer users often get confused trying to compare Outlines with Maps. This is not entirely surprising as Tinderbox always starts a new document with tabs with blank Map view and blank Outline view, the Map view tab being selected. The two blank view may look similar at this point, but they differ. This set of articles attempts to answer some of the functional side of the similarities/differences of the two view types. Note: words in this article with a $ before them indicate reference to a TB attribute, e.g. $Name is the title of a note. Here, we consider the outline and map views of the same TBX's content.

So, "What's the point?" you may ask. If you set up a simple TBX like this you can see the effect of:

- moving an item in the map: $Xpos & $Ypos change, but $OutlineOrder & $SiblingOrder are not affected.
- using bring forward or send back on a map item: $OutlineOrder & $SiblingOrder change (and the item moves in the outline) but $Xpos & $Ypos are unaffected.
- moving an item in the outline: $OutlineOrder & $SiblingOrder change but $Xpos & $Ypos are unaffected.

How the TBX illustrated above was created

On the left is the outline view, on the right the map view. The notes all have a short two-l or three-letter $Name like 'bb', 'ag' or 'rrr' and the rest of the title you see is a $DisplayExpression enabling you to see the $OutlineOrder on screen. Names where the two letters are the same are notes/containers (including separators); two different letters denote an agent; three similar letters denotes an adornment. In addition, in the map, the $Xpos and $Ypos X/Y location co-ordinates are shown; in notes/containers this is by displaying body text ($Text) and for adornments, which have no (visible) $Text, the $DisplayExpression has been further amended.

How was the mark-up done? Prototype note '*Display' (in /Prototypes) has the following Rule:

```
$Text = "Xpos: " + $Xpos + " / Ypos: " + $Ypos
```

…and DisplayExpression:

```
$Name + " / OutlineOrder: " + $OutlineOrder
```

As we cannot display text in adornments, prototype adornment '*Adornment' (seen on the map of /Prototypes) uses a different $DisplayExpression, which has to be set via the adornment's Info view (General panel). Thus $DisplayExpression:

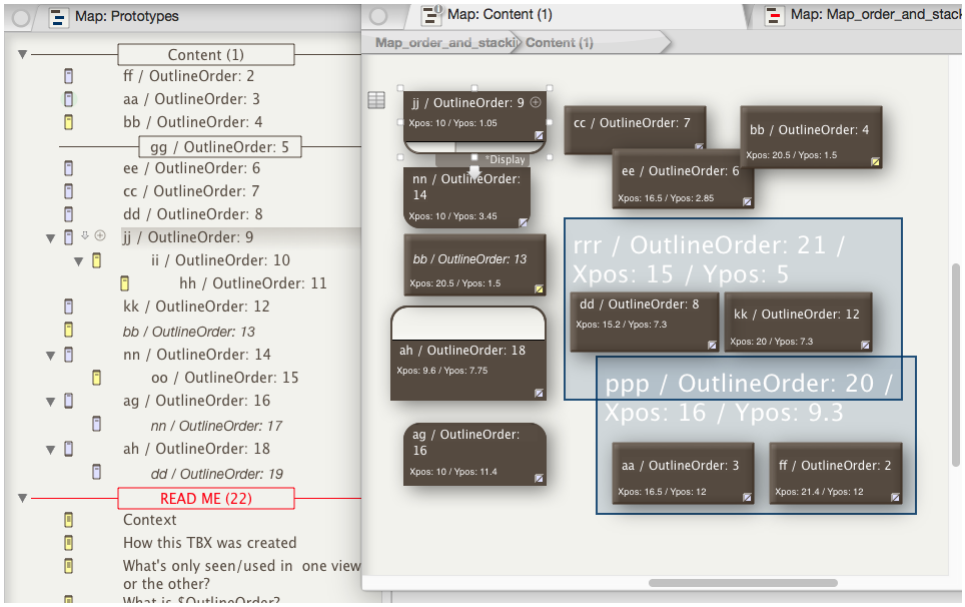
```
$Name + " / OutlineOrder: " + $OutlineOrder + " / Xpos: " + $Xpos + " / Ypos: " + $Ypos
```

You will notice that you will need to set icons a bit wider than the default and on notes/containers drag the title section down a bit to see all the captioning. Notice too that a map view container icon's viewport map shows $DisplayExpression but note text is not drawn as in a normal icon; see the icon for container note 'ii' inside the viewport of container note 'jj' at left of the map.

Next…

- Things seen in only one type of view
- What is OutlineOrder?
- What is SiblingOrder?
- Map co-ordinates
- Maps, stacking and overlapping
- Adding or Moving Items
- Hiding the map icon viewport in containers and agents

Things seen in only one type of view



What's only seen/used in one view or the other?

Siblings. A Map view is the map of the contents of a single container. As such all items on the map are siblings. Items at the same $OutlineDepth but inside different parent containers cannot be seen in the same map. Thus, no matter how b the map view window is expanded or zoomed, it will never show notes like 'Context' and 'How this TBX was created'. This is because whilst the latter are the same $OutlineDepth as note 'aa' they are children of a different parent container

and thus not siblings of 'aa'.

Separators. These are only seen in Outlines. They are not drawn on a map. Separators may have a title (as here) if the note is given a name, otherwise the separator is simply a line drawn in the outline where a note would otherwise appear. A separator is in fact a fully functional note. The separator display mode can be toggled off/on via the 'Separator' tick box on the note's Properties Inspector's Prototype sub-tab. Incidentally, if a separator is toggled back to a normal note, it will appear on the map as well! A separator can also act as a container. Making a map container a separator can be a way to hide map content, removing the separator flag (via outline view) will return the content to the map.

Adornments. These are map-only objects (but note they can display in Timelines too). On a map, adornments always show up *behind* all notes..

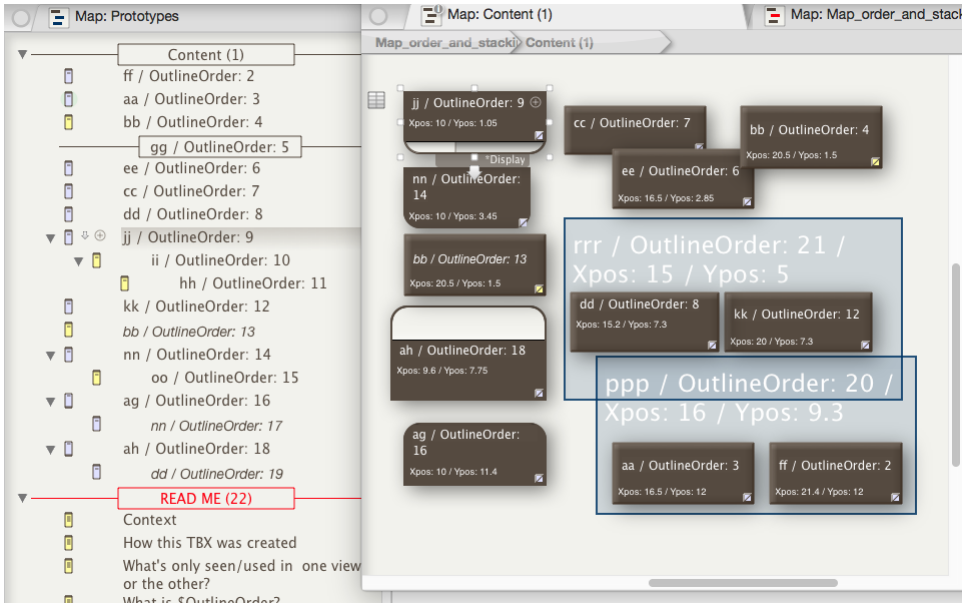
Container viewports (Notes inside containers, nested child notes). Containers (and agents) show a small part of their nested contents via their 'viewport'. In general, any descendant notes/containers (i.e. at a lower $OutlineDepth than the current one) are invisible to maps unless within the scope of a viewport.

Link lines. Not shown in the illustration, Tinderbox links are only shown in map views (assuming a link type has not had visibility toggled off).

Otherwise we have notes, containers (notes containing notes) and agents (in this context consider agents as just another sort of container); all these objects are shown on both outlines and maps.

Next… What is OutlineOrder?

What is OutlineOrder?



What is OutlineOrder?

It is a number value that indicates where, in the overall outline of the whole document, a note is to be placed. If you start a new outline and add 3 notes, they would have $OutlineOrder numbers 1, 2 and 3; the parent TBX document itself is always number 0 (zero). If you then dragged item 3 above item 2 and inspected $OutlineOrder (do this via Info view, General group) you would find old item 3 now has $OutlineOrder 2, as it is new the second item in the outline. If you nest notes and expand all the nesting, the outline numbering runs from top-to-bottom. The descendants of top level item 1 all number before top level item 2, etc. $OutlineOrder is found in the General group of system attributes.

As the name implies $OutlineOrder does affect the outline. The attribute is read-only and cannot be changed directly via action code but alter $OutlineOrder by any other means and the outline ordering will change to reflect this. For instance, add a new note higher in the outline order and an all notes below will have their $OutlineOrder recalculated and increase by one. However, and importantly for this article, $OutlineOrder is not used for controlling placement of objects on a map view. That said, $OutlineOrder does have an affect on stacking/overlapping on maps but more on that later. The lower the $OutlineOrder, the higher up the outline view that item will be drawn.

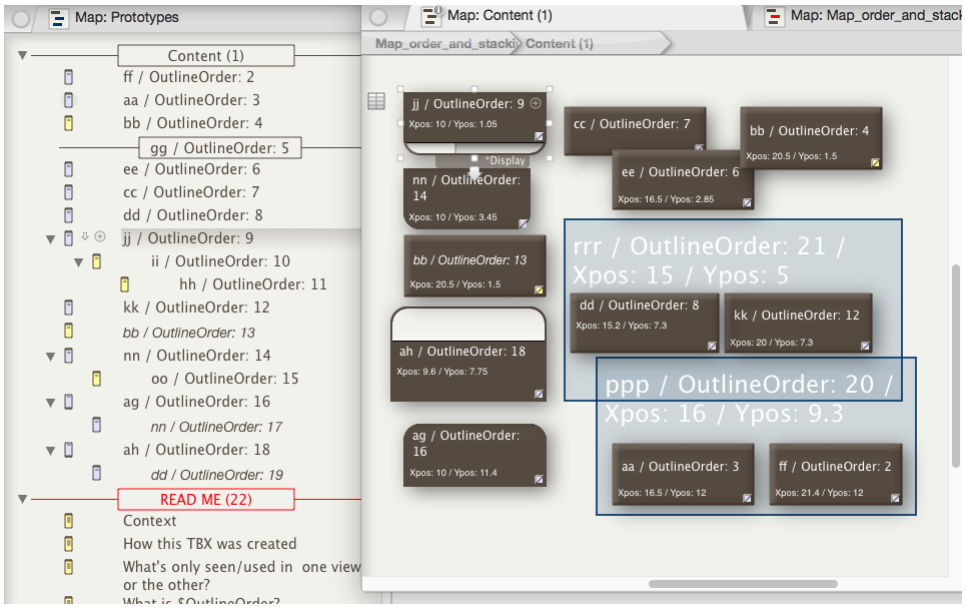
Note that the outline's separator ('gg') is not shown in the map, and not because it is just elsewhere on the map. Also notice how although note 'kk' is the next sibling to 'jj', it is not the next in $OutlineOrder as all the descendants of 'jj' are numbered in between 'jj' and 'nn'.

Aliases have their own ' intrinsic' $Xpos/$Ypos/$Height/$Width/$OutlineOrder/$SiblingOrder values. Note how the alias of 'bb' is in a different map location and has a different size (but not $Shape) from its original note It also has its own number in the outline order.

In the outline part of the picture, notice the 'READ ME' note has an $OutlineOrder of #22, whereas the last item listed before it, the alias of 'dd', is numbered #19. The missing numbers are the map adornments (#20 & #21), which of course are not visible in the outline. This is because adornments always list after all other items in the same container's map, i.e. including such items as separators.

Next… What is SiblingOrder?

What is SiblingOrder?



$SiblingOrder

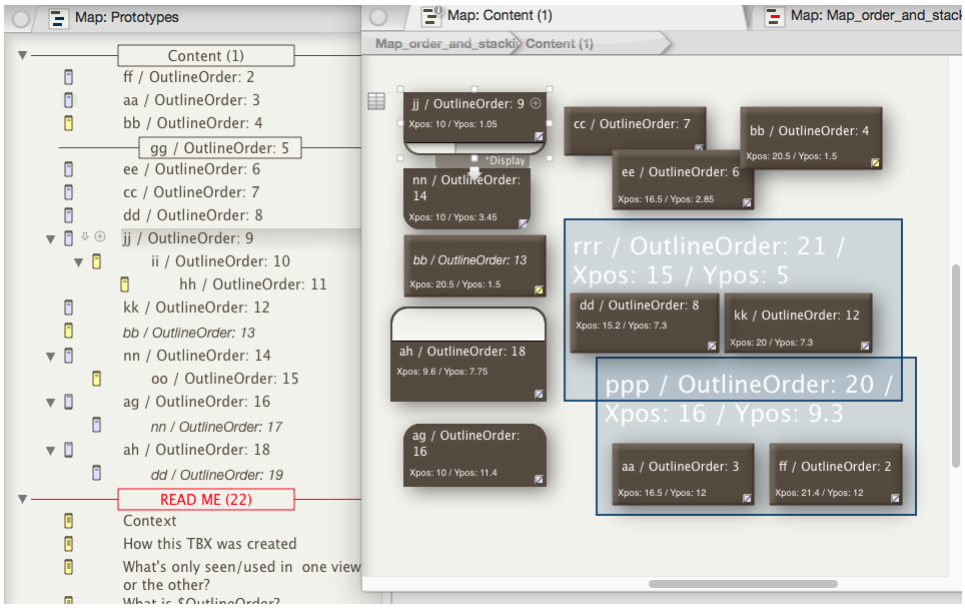
The $SiblingOrder is the top-to-bottom order, as seen in an outline, of siblings with a common container (including root level notes. The siblings number from one and the $SiblingOrder is a read-only calculated number. Within a set of siblings with no descendants, the sequence of $SiblingOrder will match the notes' $OutlineOrderSequence. However, if any note has descendants, all the latter list in $OutlineOrder before the next sibling note of their ancestor (see the previous article on $OutlineOrder for more).

In export contexts, the outline view is normally used which does not show adornments. Thus the last note in a container may not be the last outline order item in the container although it is the last exportable item. To help avoid the latter problem, the $SiblingOrder system attribute ignores adornments making it easier to detect unambiguously the first/last item in the container during export.

Whilst in Outline, $SiblingOrder is easily intuited

Next… Map co-ordinates

Map co-ordinates



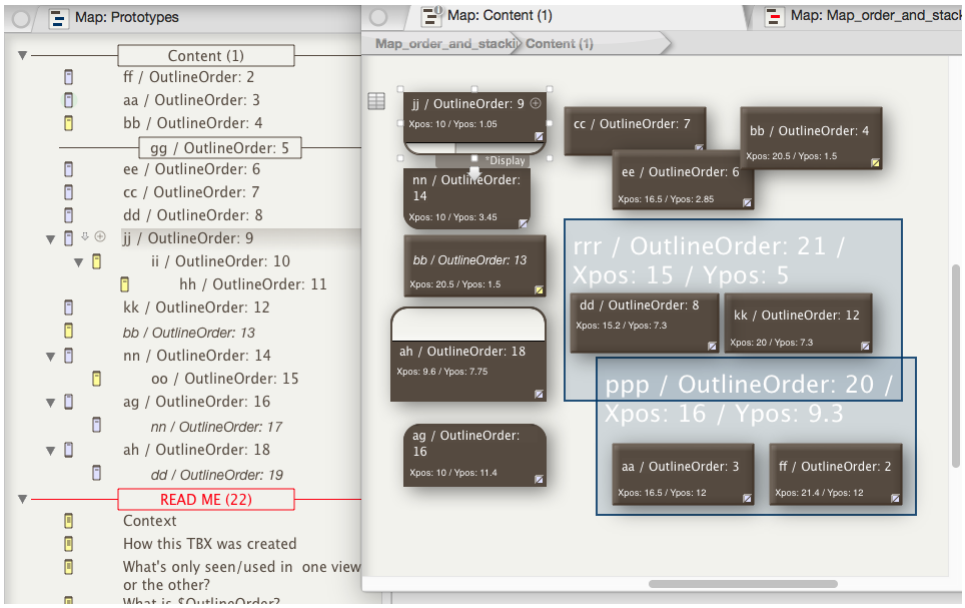
Map co-ordinates

Tinderbox uses Cartesian X/Y co-ordinates (basic graph style) to record the position of items on maps, but with one small difference; the polarity of the Y axis is reversed. So going downwards on a Tinderbox map is positive Y and up is negative Y. Why so? Tinderbox starts a new, unused, map by putting {0,0} at the top left of the map window (whether the map is visible or not) and starts drawing items down and right from {0,0}, i.e. the map is populated from the top left corner. By flipping the traditional Y polarity, new items are normally drawn with both X and Y co-ordinates as positive values. Understanding the positive/negative alignments is pertinent if using code to move items around on a map but for general use the is no need to worry about co-ordinates.

Position on a map is controlled by (stored in) the system attributes $Xpos & $Ypos for a given note (in the Map group of system attributes). These co-ordinates have no effect on the outline view, or any other view type than map.

Next… Maps, stacking and overlapping

Maps, stacking and overlapping



Maps. Stacking and overlapping

$OutlineOrder does have an indirect effect on your map, insofar as it sets the z-order (stacking order) that is applied when map icons overlap. When two container or note icons overlap, the item with the lowest $OutlineOrder is shown in front of the other icon. Thus, in the early stages of a map or outline, the$OutlineOrder tends to reflect the order in which the notes were added. In such a circumstance, in a map you can work on the principle that older notes will stack on top of newest notes (this can be slightly counter-intuitive at first encounter as we tend to think of putting the newest item on top of an older one). Tinderbox's Note menu does have 'move to front/back' and 'move up/down' controls (and associated shortcuts) which change this stacking order but do be aware that such an action also affects outline view. How? As these commands change in the map z-order, the map's overlapping order, they also change the selected note's $OutlineOrder. In turn, this moves the selected note(s) higher or lower in outline view. If there is more than one adornment on a map the same rules as above apply if adornments overlap, though this can be hard to see due to the opacity of adornments.

In the screen grab, map icons have been arranged with some overlapping to give visible proof of how $OutlineOrder affects this. Looking in the top right corner 'ee' sits over 'cc' but under 'bb'. Described in $OutlineOrder terms it is clearer: #6 sits over #7 but under #4; the lower the $OutlineOrder, the more siblings in front of which it will sit if overlapping of icons occurs.

Adornments. A new adornment is always dawn at the top of the adornment z-order (i.e. the opposite of behaviour with notes). To achieve this, a new adornment takes the $OutlineOrder of the current lowest z-ordered adornment and all others are bumped up one in the outline order. In other words, they take an initial $OutlineOrder in reverse sequence of creation within the current container, starting *above* the highest outline order numbered note. Thus adding a note also increments by one the $OutlineOrder of any existing adornments. Witness the highest $OutlineOrder on our map is #19 (given to an alias of note 'dd') but adornment 'ppp' has an $OutlineOrder of #20. The newest adornment always has the lowest $OutlineOrder of all adornments on the map (here 'ppp' was actually added after 'rrr', despite the order of naming).

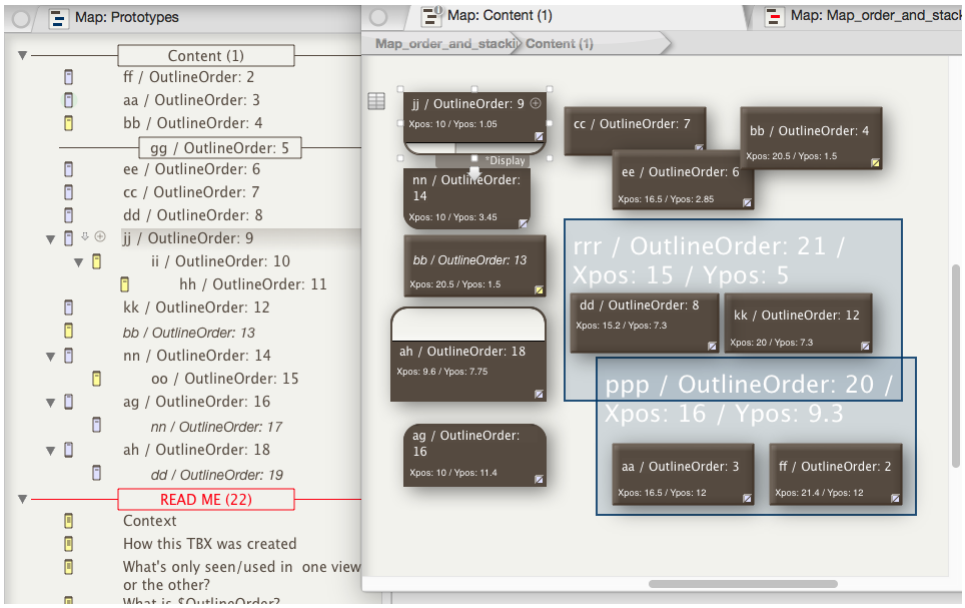
A small difference with default adornments is that they are slightly translucent so the background items border shows through. This is deliberate as it allows more creative use of adornments: for instance, to create zones to place icons that are relevant to two or more different adornment-denoted ideas. Thus, something that was pertinent to both 'rrr' and 'ppp' might be placed on the darker blue rectangle where the two adornments overlap.

Smart adornments. If a note matches more than one smart adornment on a given map, the note is moved on top of the smart adornment with the *highest* $OutlineOrder as this represents the firs created/oldest of the matching adornments.

New adornments are placed on top of existing adornments, i.e. added as the first adornment in outline/sibling order. This is the reverse of past behaviour (and continuing behaviour for notes) but should be more intuitive for users as generally a new adornment would wish to lie atop an existing one. The effect is achieved by the new adornment being inserted in $OutlineOrder before all existing adornments in the map, as opposed to after them as in the past.

Next… Adding or Moving Items

Adding or Moving Items



Adding/Moving items

In outline view, a new note is added after the selected note, or at the end of the outline if none is selected. In a map, a new note icon is added to the right of the currently selected item, to the right of the top-right-most map icon if no note is selected, or if making the note via a right click it is at the mouse cursor position. So far, so good. New items added in the map simply take the $OutlineOrder one higher than the last listed note for that container (all notes below that have their value bumped up one) so we know where it will appear.

However, notes added in Map view via a *double-click on the map background* are treated differently; Tinderbox will try and add the note in outline order one higher than the note to the left and/or above it on the map. This helps if adding note into a sequence that are later overlapped or viewed as a group in a different view. (See more on adding notes to maps).

For a note added in outline, TB will try to place it to the right of the preceding ($OutlineOrder) item on the map or close to it if the new icon would overlap other existing map note/container icons.

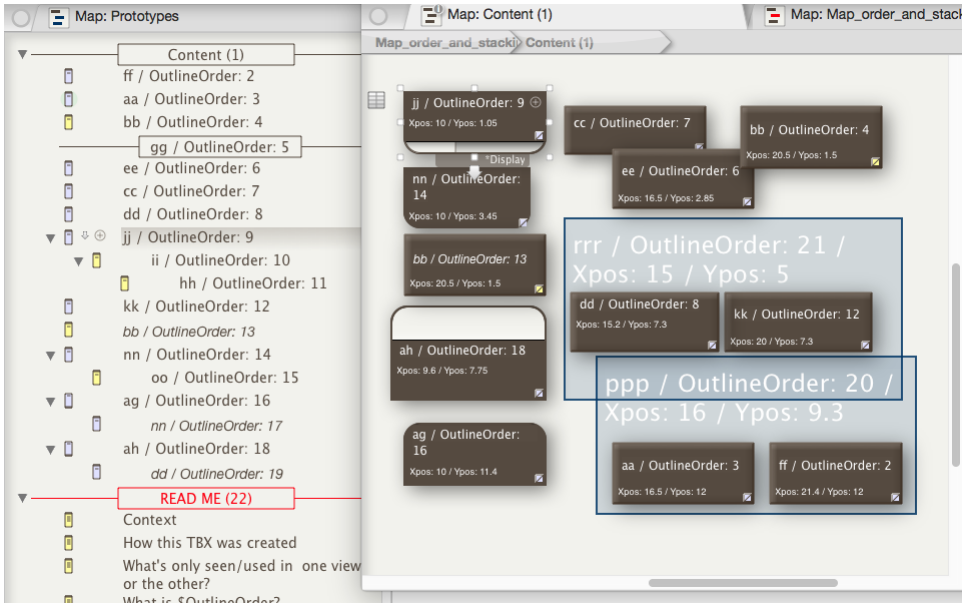
The above may all seem rather complex but just go experiment and you will see the 'rules' for order and placement are pretty simple once you have tried them out. Rules are less clear for map placement for items demoted/dropped into containers (via any view type window), but generally selection moved this way will retain their relative positions. Even if created in outline, a new note gets a $Xpos/$Ypos value pertinent to that container. If such a note is dropped into another container, it appears to try to maintain those co-ordinates unless that conflicts with those of a note already occupying that location in the destination container('s map); if such a clash occurs then TB uses its logic for new items placement to de-conflict the icon positions. Very occasionally, two items of the same height/width will overlap exactly; in such circumstances switching to an outline view can be useful in locating any inadvertently 'hidden' items.

If you start adding data in outline view, the map when first opened can look a bit messy. Conversely, you may find the outline's order looks a bit incoherent with differing topics intermingled. Remember the above $OutlineOrder vs.$Xpos/$Ypos description. Re-ordering the outline via drag drop will not move map icon but may affect stacking order if icons overlap. Re-ordering the map by moving around icons will not affect the outline view (unless you alter overlapping priorities).

Adornments and moving. Select an adornment and at top right it will show padlock and push-pin icons. Click these and the icons change. The change reflects the state of two Boolean attributes, $Lock and $Sticky (these can also be set via action code, rules, etc.). Normally, when an adornment is selected and moved, it moves independently of both the map background and any notes or other adornments. When an adornment's $Lock is `true`, it is locked to the map background. Selecting and moving a locked adornment simply scrolls the map; any items on the adornment move too but only because the whole map is moving. When an adornment's $Sticky is `true`, if moved it 'holds onto' any notes (or adornments) that are on top of it or partially overlap it. The adornment moves relative to the map background and any 'stuck' items move relative the the adornment. The latter makes it possible to easily move whole sections of a map around without complex selections. An adornment may be both locked and sticky in which case selecting and dragging the adornment scrolls the map whilst also bringing along any items stuck to it. Where adornments overlap, stickiness only applies to adornments (partially) on top of the sticky adornment, i.e. those with a lower $OutlineOrder value. It is worth noting that notes (and agents) have and can set $Lock and $Sticky attributes though doing so is generally not of great value compared to such use with adornments.

Next… Hiding the map icon viewport in containers and agents

Hiding the map icon viewport in containers and agents



Hiding the map icon viewport in containers & agents

One last trick, not really related to the above but useful for tidying up a map. In the outline, notice note 'nn' is a container but in the map, its icon (bottom left) has no viewport. How's that done? It is actually quite simple. We set `$TitleHei = $Height`, effectively hiding the viewport. $TitleHeight controls the vertical depth of the icon caption area. Notice that the bottom corners of the 'nn' icon are still rounded so the sharp-eyed should still be able to spot the container-masquerading-as-note on their map. If you lose track of 'hidden' containers, just open the map as an outline and look for any item with a disclosure ('flippy') triangle to the left of the note name.

Fine, but now we have hidden the viewport in 'nn' we cannot drill down as we cannot double-click the viewport area. Another little trick: select 'nn' and then use shortcut [Cmd]+[Opt]+[down-arrow]. Boom! The 'nn' container opens and we see the 'oo' icon. Go back up a level and 'nn' is unchanged and still hiding its content.

The agent 'ag' illustrates the same viewport hiding trick as shown with note container 'nn'. Notice how the agent retains the rounded corners at the top of its icon (for a container the rounded corners are at the bottom of the icon). I also found with the agent that if we set `$TitleHeight = $Height`, then the (display) name is written a bit too close to the top margin of the icon. Happily, if instead we set `$TitleHeight = $Height - 0.1`, it looks fine. As that also looks good with containers, using the latter expression is a better one-size-fits-all code for viewport 'hiding'. Note that as borders are drawn inside the width/height of the icon, if you set big borders you may need to reset the $TitleHeight—at least for agents.

Pane focus indicator

When the text pane or the view pane acquire the keyboard focus, either by being clicked or by Option+Tab (**⌥Tab**) shortcut, a focus ring briefly appears at its boundary. If a pane ceases to hold focus during the animation, the animation is cancelled.

The focus ring is drawn in the macOS current focus colour (default: a light blue)

Pasting notes: creation and modification dates

When a note is copied and pasted, the newly-created note has $Created and $Modified set to the time it was created. Formerly, the pasted note retained the $Created and $Modified values of the original note.

Pictures in notes

Pictures (bitmap images) can be added to Tinderbox in one of two ways:

- Image adornments (Map view only).
- Embedded in notes. In map view these images can be seen if body text preview is enabled.

Embedding pictures in notes

To add an image to a note, select a supported image format file in Finder and drag it into a Map view (but not other major views), or copy/paste from Finder. A new adornment note will be created with the image embedded in the $Text of the note. The note is otherwise a normal note and can be moved around, renamed and have additional text added to $Text without affecting the image.

Once imported in this fashion, the image can then be selected/copied from the new note and pasted into other note(s). To create a note with multiple images, import each image as a separate note and copy/cut paste the data as necessary.

Images embedded in note $Text cannot be cropped, scaled or otherwise resized (thus unlike image adornments). Select the desired image size before adding the image file to Tinderbox.

If an image pasted into the text is wider than available space, it will automatically be scaled.

From v9.5.0, Tinderbox is much more efficient in storing images placed in $ext, as well as in image adornments. Thus images may be used more freely than in the past.

Export

Notes with images in $Text will export those images during HTML Export (but not text export)

Embedded image size control

From v9.5.0, the maximum width of the stored image based on an image added to $Text, is controlled by $ImageSizeLimit. If the added image width, in pixels, is greater than $ImageMaxSize (default `1600` px) the embedded image created i scaled to the maximum allowed size; the height is scaled, if needed, in proportion the change in width. Image added below the maximum are added at their actual size.

Images and document size

If the TBX is primarily intended for export, like this document, it may make more sense to store the images outside the TBX and link the exported pages to the external assets. Given that embedded images do not export)at least not in v5.x) then it is possible for a note to have both an embedded image for viewing within Tinderbox and alongside it export code that links to an externally stored coy of the image for use in HTML.

Tinderbox compresses images; this substantially decreases the size of Tinderbox files with embedded images. Tinderbox currently uses JPEG compression, and compresses fairly aggressively; this significantly diminishes the size and loading time of Tinderbox files at the cost of some image degradation. Images are stored in the TBX file's XML using Base64 encoding.

Previewing and exporting note content

From v9.5.0, Tinderbox takes a new approach to export preview. The guiding principle is that Preview should now be readily available for any note, without configuration and without understanding templates. Where in the past this happened behind the scenes, Tinderbox now creates templates that operate normally and that can be extended and modified.

In overview, Preview is now much simpler to use. If there in not already have a template, Tinderbox will add built-in simple template into `/Templates/Preview`. That template can be customised *if desired* but this is not a requirement for u of Preview. Similarly, if the document does not yet have an export folder set, Tinderbox makes a temporary folder for the preview, and cleans up after preview is no longer in use. This process is explored in more detail below.

Template (and CSS styling) assignment

If a note is previewed and it has an assigned export template, Tinderbox continues to use that template. This is essentially the original use, that of checking (HTML) files before export.

If a note is previewed but has no assigned export template, Tinderbox adds a built-in template ' Preview' to `/Templates/Preview`. If the latter does not already exist, Tinderbox does the following:

- if not present, the Hints container is added to the document.
- at `/Hints/Preview/style` is makes a new note, which contains a style sheet that is automatically included in the preview template.
- assigns the newly-created 'preview' template to the previewed note. If either or both of these notes exists ('preview', and or 'style', the pre-existing note is used.

Both the template and the CSS styles can be altered by the user. To reset to the original would involved deleting either file and re-invoking its addition. Hint: perhaps keep a copy of the original alongside the edited version using a different $Name for the note with the original data.

Note: in older versions the preview template was stored under Hints at `/Hints/Preview/preview`.

Support for legacy markdown stylesheets, stored in the application support Markdown folder, has been discontinued. Copy any preferred style definitions to `/Hints/Preview/style`.

Preview character encoding

if a previewed note's template has no character encoding tag (e.g.), Tinderbox adds a preview-only framework to ensure that the preview pane uses the expected character encoding. This stops non-ASCII characters appearing corrupted in the preview.

Files generated via use of preview

When previewing HTML for a document that has an export folder, Preview now exports (a) the current page, and (b) any pages to which the current page has text links, if those pages are not currently already exported to disk. If the current page is already present on disk, the current version is saved and exported later (i.e. reverting back over the preview file.This stops preview of possible changes from overwriting a previous 'good' copy of the export file.

If there is no pre-existing export the new preview file is left in the export location. In addition, previewing any note will generate and file at the export location.

If there is no export folder defined for the document, Tinderbox uses a hidden temporary folder to hold the exported files and cleans up afterwards.

Why do this? This arcane waltz is necessitated by ill-documented macOS security measures which appear to offer little if any benefit to security, but which consumes time and resources that might do someone, somewhere, some good. So, not ideal, but forced by macOS limitations.

Considerations for users of both preview and export

Because preview activity may result in files in the export location that are not the result of deliberate export, users are advised to delete any current exported files before doing a full export of a document. Given the speed of export now, re-doing all files has little difference on the time taken.

Selecting notes

Select. **Click** on an item (note, agent, adornment, etc.) to select it.

Adjust selection. **Cmd+click** on a note to add/remove from the current selection. Cmd+click on the background does not clear the selection. This makes it easier to do non-contiguous selection on busy/small-scale maps and where the chance of accidentally clicking the background by mistake is greater.

Adjust selection (Outline or Chart view only) . Shift+Click on a note to extend the existing selection up/down from the last selected note.

Band-select. Option+drag to band-select notes.

Select All. Cmd+A (**⌘A**) selects all items in the current view (excluding non-displayed child/descendant notes).

Deselect all. Cmd+Opt+Ctrl+A (**⌘⌥^A**) deselects all of the current selection. Also, click on the view pane background to clear the current selection.

Multiple-item selection. Bear in mind the effect on any nested descendants (i.e. in collapsed containers). The contents of collapsed containers are ignored, i.e. only visible items are selected. This allows a set of containers to selected and acted upon without affecting their descendants. However, if such a selection is:

- copied, the result will take into account any descendants.
- option+dragged, only the visible selected items as copied.

Sentiment Analysis

Tinderbox's tagger mechanism also attempts to analyse the general tenor or *sentiment* of each note. Sentiment is measured on a scale from +1 to -1; an enthusiastic note like :

> "This cheese is tasty, wholesome, and creamy. It is a delight!"

is scored near 1, whereas a critical note like:

> "This cheese is rotten, slimy, and stale. It tastes terrible and should be thrown away."

is scored near -1.

The *average* sentiment for the entire note is stored in $Sentiment, and the score for each paragraph of the note is stored in $Sentiments.

Sentiment analysis is available in seven major languages using macOS 10.15 and later.

Setting a prototype

Once some prototypes have been added to a document, they can be applied to a note in a variety of ways. There is no one 'correct' way as the method chosen depends on the type of view in use and the task being undertaken. Amongst the ways to set a note, adornment or agent to use a prototype are:

- In Map view, right-click the prototype tab and select from the pop-up list of prototypes; applies to for notes, agents and adornments.
- In Outline or Chart view, right-click the item's icon and select from the pop-up l ist of prototypes.
- In any view, select the note, open the Prototype sub-tab of the Properties Inspector, and select from the pop-up list of prototypes.
- In any view, select the note and use the Quickstamp sub-tab of the Properties Inspector to set that item's $Prototype attribute using action code: e.g. using this code `$Prototype="X"` as the quickstamp will set the prototype called for all currently selected items.
- If you frequently need to set a particular prototype, make a stamp whose action code sets a $Prototype value. Then (re-)use the stamp, either from the Stamps menu or the Stamps sub-tab of the Document Inspector.
- If most or all the notes in a container use the same prototype, use the container's $OnAdd action to set a $Prototype, saving you the bother of doing it yourself.
- If all the notes that meet some criterion need to use the same prototype, create an agent whose agent action sets that prototype. In this case, the action code in the agent's action will set a value for $Prototype.

Spell Checking scope

From v9.5.0, to improve support, spell checking is enabled by default in the title field of the main window text pane, and some other text fields—such as that for note titles (i.e. $Name). Previously the checking was limited to the $Text area.

It may be toggled by choosing menu Edit ▸ Spelling and Grammar ▸ Check Spelling While Typing. This state is saved with the document.

Suggest Attribute value lists

This is a means of pre-populating attribute value pop-up lists with values that may not yet be in use by any note. Suggested values apply to String/Set/List data types only. If the list of likely values is known in full (or part) before the such data is added to the document, it is easy to pre-populate the attributes value list via the 'Suggested' box on the relevant Inspector for User and (some) System attributes.

Suggested values *always* appear in the value menu of Displayed Attributes, and are always offered for autocompletion, even if no notes currently use them. Suggested values may be added to attributes

Pop-up value lists do not indicate which are suggested values. Thus it is not possible to tell apart currently used values solely by inspecting a pop-up that includes suggested values.

Defining suggested values

In the relevant Inspector, enter the list of suggested values as a semi-colon delimited list without any enclosing quotes; this follows the normal Tinderbox convention for defining a list in action code. Thus, to pre-populate an attribute with values 'Ant', 'Bee' and 'Cow' the unquoted string to enter into the Inspector's suggested box is `Ant;Bee;Cow`. A terminating semi-colon may be used but is not required. It can be helpful to prepare the list in the text of another note if it is lon and will extend beyond the visible portion of the suggested box; copy the text and then delete the note.

If adding a list to a document that already has values, it may be useful to first test the new list in a new document to ensure the correct values show up in a Displayed Attributes pop-up. Indeed, this method does not prevent new values (i.e. those *not* in the suggested list) being used. If trying to assert a fixed list of values and new values are seen in the pop-up list, note the new value(s) and use an agent to find and correct the notes with the unwanted values.

Automatically setting suggested values

See Pre-populating Displayed Attributes pop-up lists .

values() and suggested values

Values that appear only in the suggested list, i.e. are not (yet) used by an actual note, are not included when using the values() action. Indeed, suggested values do not show up in values(), collect() and such, if not actually used by any note.

Legacy

A note for long-time users: this method replaces an older approach of making 'seed' notes as placeholders for desired prepared values.

Tear-off windows

Some pop-over dialogs, listed under Secondary windows, can be turned into stand-alone windows and are sometimes described as 'tear-off' windows. Pop-overs supporting tear-off advertise the possibility with an icon in the top right corner of the pop-over.

Tearing-off is done by clicking-and-dragging on the margin of a pop-up. After a small distance of drag, the pop-up will change to discrete window.

Tear-off windows have normal window controls and can be closed or minimised like other windows.

The torn-off state does not persist beyond the end of a document session, i.e. when the parent document is closed.

Currently, the pop-overs offering a tear-off option are:

- Get Info.
- View pane Find Results.
- Roadmap.

Text for multiple selections

When selecting a composite or multiple notes, the text pane shows the concatenated $Text of all the selected (or composite) items. The texts appear in outline order ($OutlineOrder with a grey rule is drawn beneath each note's text. Each note is headed by its title drawn in the same light grey at in small font size. The latter help the user understand from which source note that section of text is drawn.

Embedded Images

If images are present in the $Text of selected items, these are included in the concatenated view.

Editing

This concatenated text view is not (currently) editable; to edit any text, de-select all and select the individual note requiring edit.

Copying

The concatenated text may be selected and copied. Pasted into a text space in Tinderbox or another RTF text program, the concatenated text and images are pasted but not the rulers and not titles.

Title Strike-Through

In most major views, titles can be rendered with a strike-through. This can be triggered a number of ways:

- Directly setting the $NameStrike attribute via code
- Format ▸ Font sub-menu
- Shortcut [Cmd]+[Shift]+[-] (⌘⇧-)

This can be useful for applications like 'To Do' lists to help indicate completed or actioned items.

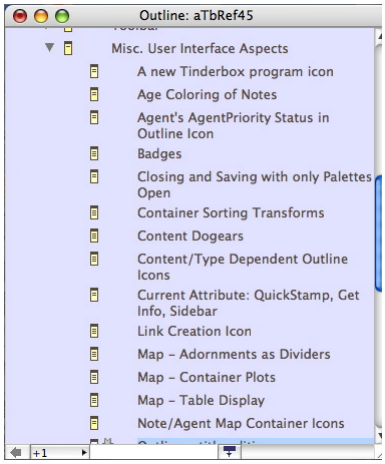
View filters

View filters are currently implemented in Outline view only.

Wrapping of long titles

In Outline, Map, Chart, Timeline and Treemap view, long titles wrap to successive lines. In occasional cases, if all titles do not wrap as expected, re-size the view window slightly to force a re-draw of the window and wrapping should be correct.

In map view, there is a preference to auto-expand notes horizontally or vertically. The same commands can be called via menu or shortcut if the preference is not set.



Formatting

Command Line

Tinderbox allows users to extend its capabilities by giving users access to the command line (CL) in a number of ways:

- Action code:
  - runCommand()
  - back-tick marker (now deprecated, use runCommand() instead)
- Attributes:
  - $HTMLExportCommand
  - $AutoFetchCommand

The above actions and attributes enable execution of stored command lines or action code expressions. For the attributes above, the command line is run on the post-processed of HTML output or AutoFetch input.

The most flexible method is runCommand(). See the referenced note for more detail. The runCommand() operator can also be used to call external scripts, i.e. script files stored in the OS outside the TBX document, using an POSIX-style path (i.e. slash-delimited, not colon-delimited); paths can be absolute or relative.

Scripts used without without a path are assumed to be in the shells current working directory ('pwd'). Tinderbox's default directory is working directory is the user's home folder (i.e. `/users/[shortusername]` or in short form `~`); *each* new runCommand call thus starts at even if it was altered in a previous call.

CL code can also be stored in string attributes, such as $Text, but only when using runCommand(); the other methods above require literal CL code. Referenced CL code is called thus:

```
runCommand($Text("CL example"))
```

Using a note's $Text to hold code can be very useful for CLs too long or complex to work on easily in a $Rule or $OnAdd box. In the text window you can use a monospace font, crank up the font size, etc., Tinderbox provides an easy method via the built-in prototype 'Code'.

Because of the external calls involved, CLs used in action code such as agent actions, can have significant impact on agent cycle time. This is especially so if using CL code to process text. In an agent context, consider turning the agent off (via $AgentPriority) when processing is not needed.

It is worth noting Tinderbox's support for single-quote and double-quote string quoting but bear in mind that there is no method for escaping a double quote character.

Working directory location

When accessing the command line this way the targeted location is the host Mac's home directory. Regardless of other Terminal sessions that may be active, the current directory for Tinderbox access is the root of the current volume, i.e. ' - Thus, if calling scripts or programs elsewhere on the host system, such as in the user's account's 'Documents' folder, remember to prefix an appropriate path to the script/program (or set a new directory via the method described below).

If the working directory is changed as part of the runCommand call from Tinderbox, that change will hold for the remainder of the call. For instance, a runCommand() string `"cd ~/Documents; pwd"` with return the full path to the user's Documents folder (e.g. '/Users/jdoe/Documents') and will become the working directory for all subsequent commands passed in that string, but not for calls from other notes. Observe the trailing semi-colon after the `cd` command, this tells the command line that the first command is complete allowing a follow-on command to added with is executed after the previous one. In the example given, a follow-on command would execute within a non-default current working directory.

Setting the working directory

runCommand(command[,input(s),directory])

If the optional **directory** is specified as a POSIX path, it sets the working directory in which **command** is executed. Otherwise, by default, the working directory is the user's home folder (i.e. `/users/[shortusername]` or in short form `~`).

Script Permissions

External script files also need to be executable. It is not enough to make a new file in something like TextEdit and save code into it, extra permissions are needed. By default a file is 644, a script needs to be 755. Assuming a script in your home folder, and a root working directory, this can be done with a runCommand() call:

```
runCommand("cd ~; chmod 755 somescript.pl")
```

Of if the script name is stored in a notes $MyString:

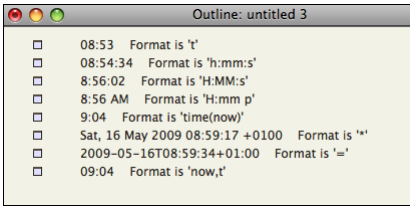
```
runCommand("cd ~; chmod 755 "+$MyString)
```

Copy to Clipboard

When note(s) selected in a major view and copied to the OS clipboard, Tinderbox adds a textual list of the note names with one note name per line (paragraph) when pasted back outside Tinderbox. This should be handy for copying to word processors and email.

Within Tinderbox different results apply.

Date Formats



Tinderbox offers numerous date formats. An example date Tue, 29 Apr 2003 14:32:18 - 0500 is used to show the following various format strings below. Note that from v5, seconds cannot be set (and are ignored for comparisons) and are always shown/exported as '00'.

Do remember that the individual codes can be defined in a format string. The codes below are thus like placeholders in a literal string:

```
$MyString = Today is " + date("today").format("W, d MM y")
```

On 29 February 2012 on a UK-locale system this makes $MyString:

Today is Wednesday, 29 February 2012

Note above how the spaces and comma within the format string are retained.

Below, the colons after each (bolded) code are not part of that format code but just a divider from the text that follows it; codes are all one or two characters. In places, clarification is given, e.g. number zero versus letter o. Dates before 1 January 1904 (start of the macOS calendar) are supported: see more.

Important note: the examples here are for US & UK system settings so as to illustrate day/month and month days variations. The exact format you see on your system from the same format string may differ depending on your computer OS' local system settings (macOS, see System Preferences, Languages & Text, Formats). For reference, this TBX was authored on a system using UK settings, so generally you will see example dates use day/month order.

Time offset calculation: the time offset is always applied for the current time in the current locale as derived from the host macOS.

Keywords used in specifying dates

Although not related to _formatting_ dates, there also keywords, e.g. `now`, `yesterday`, etc., that can be used when _setting_ Date-type attributes.

Date/Time formatting codes

Specific complete formats:-

L : local date/time, as in the 'long' format of the host system format settings. Examples:

US: `April 29, 2003`

UK: `29 April 2003`

l : local date, in short format, using the system format settings (**l** is lowercase L, not numeral one). Examples:

US: `4-29-03`

UK: `29/01/03`

***** : date/time in RFC 822 format. Example: `Tue, Apr 29 2003 14:32:00 +0500` (see note above on time offsets)

= : date in ISO 8601 format. Example: `2003-04-29T14:32:18+05:00` (see note above on time offsets)

U : formats the data in Unix epoch time, i.e. the number of seconds before or since the beginning of 1 January 1970.

n (for "normal"), which displays the system medium date string and short time string. Example; `21 Jan 2020 at 17:12`.

Formatting code for specific parts of date/time, combine as required for custom format strings:-

Formatting codes for the day part of a date:-

d : day of the month, not zero-padded. Example: `9` (note, not '09')

D : formats the date of the month as a two digit number, with a leading zero as required (01-31). Example: `29`

o : day of the month as an ordinal (lowercase 'o'). Example: `2nd`, `10th`, but conversion respects the current locale's abbreviations. (v9.5.0)

w : abbreviation of weekday. Example: `Tue`

W : name of weekday. Example: `Tuesday`

Formatting codes for the month part of a date:-

m : number of month, not zero-padded. Example: `4` (note, not '04')

M0 : formats the month as a two digit number, with a leading zero as required: 01-12. Example: `04`. Note the second format character is zero, i.e. the format code is 'em-zero' not 'em-oh'.

M : abbreviation of month. Example: `Apr`

MM : name of month. Example: `April`

Formatting codes for the year part of a date:-

y : 4 digit year. Example: `2003`. (On some countries' settings this may still be 2 digits.)

Y : 2 digit year (last 2 digits of year). Example: `03`

Formatting codes for time:-

t : time, in local format. Examples:

US: `2:32 PM`

UK: `00:32`

a : accepted as an input abbreviation for "am".

p : accepted as an input abbreviation for "pm".

Formatting codes for the hour part of a time:-

h : hour of the day on a 24-hour clock, zero-padded for single digits. Examples: `14` from 14:32/2:32 PM, `05` from 05:32/5:32 AM.

H : hour of the day on a 12-hour clock. Example: `2` from 14:32/2:32 PM, `5` from 05:32/5:32 AM. Use with 'p' to show AM or PM suffix, e.g. "H:mm p" gives `2:32 PM`.

p : the A.Mr P.Mf the hour, always uppercase, with no periods. Example: `AM`

Formatting codes for the minute part of a time:-

mm : minute of the hour, zero-padded for single digits. Examples: `32` for 32 past or `05` for five minutes after the hour.

Formatting codes for the seconds part of a time:-

s : second of minute, zero-padded for single digits. Examples: `02`, `18`.

Escaping any of above as literals:-

Prefacing any character with a backslash, \, includes the that character literally, even if it otherwise has a special meaning in the above list. Thus a format string of "\dd \mm \yY" gives the output `d29 m4 y03`. Any other character includes t character. Thus, the format string "Local time:- h:mm:s" gives output of `Local time:- 14:32:00`, except that the seconds are always zeroed (in v5+).

Dragging text within $Text

Within the text pane you can drag a text selection containing text links and move the links along with the text.

Number Currency Formats

Number-type data can be formatted using the format() or .format() operators:

$ for local currency. When formatting numbers format("$") and .format("$") apply conventional formatting for your local currency – for example, "$1,063.52" from number 1063.52. Usage:

```
$MyString = $MyPrice.format("$")
```

$0 for rounded local currency. The second character is a zero and not letter 'o'. When formatting numbers format("$0") and .format("$0") apply conventional formatting for your local currency, rounding to the nearest currency unit – for example, "$1,064" from number "1064.32".

The above currency format converters will use explicitly-set locales, rather than be fixed the user's default system locale.

Duplicating items

Duplicating and note, separator, container, agent or adornment should create a new copy of the item appending the word "copy" to the note title. If a note's name contained characters other than digits but ends in a digit, the number is incremented. For example, duplicating "42" creates "42 copy", while duplicating "Catch 22" creates "Catch 23".

However, note that a container's contents are not duplicated if only the container is selected (regardless of whether its contents are displayed or not).

If more than one item is selected, each item creates a copy of itself, alongside its sibling and preceding it in sibling order (i.e. one lower in $OutlineOrder). Nested items, which must be visible and selected, are created at same outline depth a their source note, and as siblings of their source note, not children of their duped parent. Selections need not be a continuous block of items but can be non-contiguous selections and from one or more outline levels though they all need to be visible in the current view (so as to make the selection).

Links. Outbound basic and text links (including web links) are created in the copy. However inbound links to the original note are not and need to be recreated for the duplicate, if such are needed. Thus if splitting a note by duplicating and then deleting different parts of its text, consider reviewing inbound and outbound basic links as these will need some manual triage.

Option+dragging a selection adds all the new items in a single contiguous sequence although any nested items are created at the outline depth of the top-most item in the selection. For instance if a root level note and one of its children and grandchildren are selected, and Option+drag will result in three sibling notes at root level. There is no way to drag+copy and maintain relative outline structure.

Last-used OS folder

Be aware the Tinderbox uses the same stored folder name for:

- opening new files
- saving files
- doing a 'save as'
- HTML export
- text export

If unaware, this sharing of one location can easily confuse the user as export location(s) are usually different from these used for file open/save operations.

Linking to local files (File-type attributes)

The System attribute $File and File-type User attributes can link to a single file or folder per attribute. When displayed in the Displayed Attribute table, this attribute type has a folder icon to the left of the value box. Clicking the button allows setting a target folder or file. Or, if set, files are opened using Finder and folder are shown in Finder.

If the destination document may move around on the user's system, it may prove more useful to make an alias for the destination document, place the alias in a location where it will not get moved and link the alias to the Tinderbox attribute. I the alias method is used for many notes/files, it can prove useful to create a single Finder folder for all these aliases and thus stop the aliases themselves from being moved and so breaking the Tinderbox attribute's link.

As stated, the link can be to a folder. Using the latter allows a single note/attribute to link to a collection of files, for instance a folder containing all files relating to a particular project.

Markdown preview rendering

***Note**: Markdown is not a core feature of Tinderbox, although it supports its use as detailed below. Thus, problems using Markdown syntax to get the correct HTML output are best researched in the Markdown community. Or, if asking in the Tinderbox user forum be mindful that not all Tinderbox users know about or understand Markdown.*

How does Tinderbox know to parse a note for Markdown markup?

This is done if $HTMLMarkdown is set to `true` for a note. This occurs in a number of ways:

- zero-configuration use of the text pane 'Preview' tab. Doing this in a document with no export templates, adds the Hints folder (and its preview-related content. This includes setting the note's $HTMLMarkdown to `true`.
- setting a note to use the built-in 'Markdown' prototype (which sets $HTMLMarkdown to `true`).
- setting $HTMLMarkdown to `true` either manually or via action code.

As well as parsing for Markdown syntax such notes also evaluate the $Text as ^text(Markdown preview rendering,PDF-source-item):: text tries to include itself including any incline export code; inline HTML code is treated as deliberate code (as opposed to literal text) and so not escaped.

Details of how Markdown and its various flavours work is out of scope for aTbRef and should be researched in Markdown resources such as markdownguide.org.

If no export template is defined for the note, the built-in default of ^text^ is used for selecting the content shown in the preview.

Tags embedded in the text such as ^value()^ are evaluated before being passed to Markdown.

Do not use place Markdown style marking around Tinderbox links in $Text (neither text nor web types of links) or the Markdown parser will double-encode the web link; it will look like a link but not work. In such a scenario, *always* check the source HTML of the exported page, or if working in the in-app Preview tab, look at the HTML code in the text Export tab.

Markdown variants available

Unlike HTML, evolution in the Markdown community means there are a number of 'flavours' of Markdown available. Each shares core Markdown feature support but then adds additional syntax/features some or none of which are compatible with other Markdown flavours.

The Tinderbox app ships with two flavours including:

- `'CommonMark'`. This is an internal pointer to the bundled copy of the newer and faster rendering fork of the original Markdown. Details of CommonMark features and how they work are at commonmark.org. CommonMark preview permits embedded HTML.
- `'Markdown'`. This is an internal pointer to the bundled copy of the original Markdown processor. Details of CommonMark features and how they work are at daringfireball.net.
- OS path to some other Markdown script. This allows Tinderbox to use a specified Markdown variant (e.g. one to which the user has added extra plug-ins) stored in the host macOS system.

In all the choice can be altered by setting the $HTMLPreviewCommand. If the latter is left blank (the default) 'markdown' is the assumed variant to use.

When using export, the text pane's Export tab displays the current note's $HTMLExportPreviewCommand value and allows the Markdown processing to be toggled on/off to help with troubleshooting its use.

Details of Markdown syntax and its various flavours work is out of scope for aTbRef and should be researched in Markdown resources such as markdownguide.org.

Preview streaming to other apps. When a Markdown-prototyped note in Tinderbox is selected or edited, its export and Markdown code-evaluated contents will be sent to apps that support preview streaming. Markdown apps currently known to use preview streaming with Tinderbox:

- Marked 2 (see more).

Negative Dates for years BCE or BC

Tinderbox has the ability to handle negative dates, i.e. those years BCE (or BC). Negative integers from -1 to -2500, when coerced to dates, are interpreted as years BCE (or BC). This is particularly useful for timeline view. Negative dates **mu**

be entered as just years. Do not use day/month data or the process, i.e. '-0145' not '-1/12/0145'. Also, take care to always supply a four-digit year, i.e. '-0045' not '-45'.

However, it is important to note that default date settings (macOS System Preferences:Language & Text:Formats) do not display AD/BC in dates. As your systems 'short' date format is used for the display of Displayed Attributes data in text windows, if you wish to see negative dates correctly displayed, you may wish to customise your short date format in order to distinguish eras. Do bear in mind such changes may impact other applications that also use the system 'short' date and which my not expect/handle a non-default setting.

If Tinderbox is open when the system settings are changed, close and re-start the application to see the new date formatting.

OS Services

The normal macOS services are available from text windows. Use the Services sub-menu under the Tinderbox menu.

Quick Lists

The term 'quick list' describes Tinderbox's ability to recognise certain characters in $Text as indicating an indented list and to export that list in several forms. Thus Tinderbox recognises as unordered lists paragraphs that begin with one or more asterisks or bullets:

```
* like this example
** use two or more asterisks to embed lists within lists
```

The above gives this sort of HTML output :

* like this example
    - use two or more asterisks to embed lists within lists
* like this example
    - use two or more asterisks to embed lists within lists

…also as ordered lists, those paragraphs that begin with one or more hash marks

```
# like this example
## use two or more hashes to embed lists within lists
```

The above gives this sort of HTML output:

1. like this example
    1. use two or more hashes to embed lists within lists.

List types can be mixed. For instance:

```
* like this example
## use hashes to get numbered lists
## numbered lists can be in bulleted lists
* and so on
```

That gives:

* like this example
    1. use hashes to get numbered lists
    2. numbered lists can be in bulleted lists
* and so on

Tinderbox will indent whole paragraphs beginning with an asterisk or hash, by one tab-width per quick list symbol. A quick list item starting with 3 asterisks will show in TB as a paragraph indented 3 tab-widths.

Several attributes and preferences control quick list handling in note windows and at export:

- At app or document level quick list indentation can be disabled entirely by a Text Preference, though HTML export of quick lists will still work.
- Indenting of quick list text within a note text window *only*, and without affecting list creation on export, can be controlled by $AutomaticIndent.
- If $HTMLMarkupText is set to `false` (the non-default setting), all list indenting is turned off and all mark-up for export, including quick lists, is disabled e.g. paragraph tags, HTML entities, etc. This choice is useful if exporting notes holding code (CS, JavaScript) which must be exported verbatim; the export template would still just use ^text^ but the note's $Text would be inserted into the output verbatim, without any modification.
- If $IsTemplate is `true`, the effect is the same as setting $HTMLMarkupText to `false`, effectively overwriting the latter's current setting to ensure templates perform as expected.
- If the $HTMLListItemStart and $HTMLListItemEnd attributes are empty, all quick lists, as defined by lines starting with an * or a #, are disabled and content is exported verbatim (i.e. including the * or # markers).
- Outputting text using ^value($Text)^ ignores quick list markup and the list source is exported verbatim.

Unordered lists can be indicated using either an asterisk (*) or a bullet (•).

To stop a quick list character at the start of a line from being interpreted as quick list markup, either:

- Place an Option+space character at the start of the line.
- Use the HTML encoded symbol for that character, e.g. replace '#' with '#', or '*' with '*' etc.

Reset $Text formatting

Now $Text is a fully RTF writing space, reformatting entire note texts is more complex, not least as discrete fonts are used for setting bold and italic text—albeit from the same font family. Indeed, if the font family of the $TextFont used in th note (or current selection) lacks a bold or italic variant, then those forms of styling cannot be applied unless a different $TextFont is selected. (A historical note. The Lucida Grande used in v5 had this problem: default Mac fonts included a bold font face but no italic).

Text-space formatting is controlled via three options on the Style sub-menu of the Format menu:

- **Standard Size**. (Cmd+Shift+T). Sets the selection to the default font size ($TextFontSize). If selected from View pane, the styling is applied to the whole text of the current note. A text selection is not required so a note with no $Text ca be reset.
- **Standard Font**. (Cmd+Opt+Ctrl+T). Sets the selection to the default font ($TextFont). Note that this destroys any bolding or italics as these are set using variant font faces. However, the process does try to respect passages that are bold or italic; it changes the font family to the note's default font family, using the note's $TextFontSize size. If selected from View pane the styling is applied to the whole text of the current note. Standard Font is more aggressive in removing indentation, background colours, text colours, embedded tables and list formatting. This is often what one wants when pasting from formatted sources such as Web pages. A text selection is not required so a note with no $Text can be reset. As this option often also changes embedded rules, it is often necessary to also use the next option below when 'resetting' a note's text.
- **Reset Margins**. This resets paragraphs in the selected range to use the standard margins and line spacing. There is no keyboard shortcut.

Document-level changes are also possible. In Document Settings/ Text, when the default Text Font is changed, Tinderbox scans the text of every note in the document and changes each usage of the former text font to adopt the new font family while retaining the current size. As long as only one font family is in use for the $Text any bold and italic sections of text should adopt the new font (do note the need for the new font's font family to include installed bold and italic variants). Note text using other fonts are ignored. For instance, it would be unwelcome if such an update changed the monospace fonts used in Template- and Code-prototyped notes.

Spotlight support

Spotlight should catalogue Tinderbox document files that have extensions ".tbx" or ".xml"

Smart Dashes

Tinderbox uses underlying Apple frameworks to automatically substitute a pair of successive dashes -- (as via keyboard minus key) with a long en-dash –, or 3 dashes --- with an em dash —. Note: for historical and practical reasons this feature is not used in aTbRef.

Smart Dashes are also controlled via $SmartQuotes. This makes it easier for those with more precise quote/dash control to disable these smart features which are on by default

Either change $SmartQuotes for the note or use the Substitutions menu either via the Edit menu or the text area's right-click menu. *Warning*: this menu change only lasts until the note is de-selected, on re-selecting the note the default value re-applied.

Smart Links and URLs

Tinderbox uses underlying Apple frameworks to automatically create 'Smart' Links (also call Smart URLs) where URLs are detected in $Text. Automatic detection of such URLs is enabled by default. It can be disabled using Edit ▸ Substitutions ▸ Smart Links. The setting applies to the current note rather than per-note and only applies while the note is still selected: on losing focus the setting returns to the default 'on' state.

To control Smart Link creation more persistently, use the $SmartLinks attribute. This preserves state and can be inherited via prototypes, etc. Templates notes are configured in this manner via their prototype as such link creation is unwantec in a template context.

Unless disabled in the Edit ▸ Substitutions menu or $SmartLinks, if Tinderbox encounters a valid URL being typed or pasted into $Text is will create a usable Web links. Previously, these links only worked from within (the RTF of) $Text, but now any detected URLs are added as full Tinderbox links and may be seen/edited in the note's Browse Links pop-over.

When Tinderbox adopts a smart link found in $Text as a Tinderbox web link, it gives sets the path of the new web link to *untitled (i.e. no (visible) link type).

From v9.5.0, Smart Links are now disabled in the built-in prototypes for Code and Action. This prevents expressions like `$MyString.at(0)` from being treated as URLs in Austria.

From v9.6.0, when Tinderbox converts a smart link in the text to a Tinderbox web link, it now assumes that the appropriate scheme is https if no scheme was specified.

Smart Quotes

Tinderbox uses underlying Apple frameworks to automatically substitute typographic ('curly') curly single or double quotes instead of straight ones where they are found in $Text. Note: for historical and practical reasons this feature is not usec in aTbRef.

Thus if a quoted word ‘funny’ is typed with straight quotes, the ‘smart’ mechanism will change it to ‘funny’. Similarly “funny“ becomes “funny”. The effect is not retro-active. Changing the setting has no effect on pre-existing existing $Text in notes.

This feature can be controlled at Document Level or at note level. Use either of these methods (both are interconnected):

- Document. See Document Settings, Text tab, Smart quotes. Toggling this both sets Smart Quotes for the whole document and the default value of $SmartQuotes.
- Note. Either change $SmartQuotes for the note or use the Substitutions menu either via the Edit menu or the text area's right-click menu. *Warning*: this menu change only lasts until the note is de-selected, on re-selecting the note the default value is re-applied.

Smart Dashes are also controlled via $SmartQuotes. This makes it easier for those with more precise quote/dash control to disable these smart features which are 'on' by default. Note that if using the menu (option #2 above), then Smart Dashes and Smart Quotes must toggled separately via their own discrete menu items.

Support for other app-specific formats

- BibDesk
- Bookends
- Calendar
- Delicious Library
- DEVONthink
- Evernote
- Finder
- FreeMind
- IAWriter
- Marked2
- Microsoft Word
- Notes
- OmniFocus
- OmniOutliner
- QuickCursor
- RIS files
- Scrivener
- Simplenote support
- Storyspace
- TaskPaper
- Tot

BibDesk

Dragging BibDesk files (RIS data) to Tinderbox will create new notes.

Bookends

Tinderbox supports Sonny Software's Bookends reference and citation manager.

To drag a reference into Tinderbox, use Cmd+Opt+drag from the Bookmark window into a Tinderbox view or a Tinderbox text window. N.B. Bookends requires the Cmd+Opt keys to be depressed before clicking on the item to drag; otherwise the import to Tinderbox does not occur correctly.

RIS reference import (and option-drags from Bookends) extract the reference's abstract, if present, and place it in $Abstract.

Drag imports from Bookends are coerced to $TextColor, which improves interoperability in dark mode.

With Bookends import authors are no longer re-sorted in alphabetical order, and $Authors respects the order in which the authors are listed in the source reference.

Using Bookends v11.2.9+

Use a Cmd+Opt+Drag, ensuring the Cmd+Opt are depressed **before** the drag starts (otherwise different data is loaded). The modified drag will result in:

- creating a new note.
- setting the note's $URL to a Bookends reference URI
- set the $Text of the note to formatted Bookends reference. The styling/layout used will depend on the user's settings in Bookends. In most cases the result is the reference source data formatted for citation in that reference format (e.g. ACM, IEEE, etc.). With some references export format choices, like BibTeX, the $text appears to be a form of code but this is the correct export for that format. Notes created by ⌘+⌥-drag from Bookends display the imported citation in the $TextFont, not Helvetica.
  - WARNING: this will replace any existing $Text in the note.
- apply the 'Reference' built-in Prototype (adding the latter to the TBX if necessary). Note this process is linked to the Network feature, so multiple newly added reference notes my appear to set their prototype one at a time, i.e. there may be a short delay before all new notes are prototyped correctly.
- set a variety of reference-related attributes to Bookends-derived values (mainly attributes from the References group).
- if an imported reference from Bookends is not valid unicode, Tinderbox rejects the import rather than risk a corrupt value of $ReferenceRIS.

Bookends v12.8+

Support for dragging multiple items.

Calendar

Tinderbox accepts drags from the Calendar. If the document has an Event prototype, the newly-created note is assigned that prototype. $StartDate and $EndDate will be populated; the name of the note is the event title and the subtitle of the note is the event location.

Delicious Library

Selections copied from Delicious Monster's library application, Delicious Library, are pasted as reference item. Pertinent attributes are automatically filled.

DEVONthink

Tinderbox supports a number of methods to interchange data with and inter-connect to DEVONthink.

- **Import**:
  - '.tsv'file format. Behave like text '.txt' files when dragged to Tinderbox, using the tab-delimited data method.
  - '.opml' file format. This follows normal OPML import methods.
  - Single or multiple notes can be pasted or dragged from DEVONthink Pro whilst retaining their tags (as $Tags). The created note's $URL is set to source item's DEVONthink URI.
  - A Tinderbox note can be set to auto-fetch a DEVONthink note. Set $AutoFetch to `true` and $URL to the DEVONthink URI of the desired note. When Tinderbox routinely fetches Web data, it will also reimport the text and name of the DEVONthink note.
  - Improvements to imported text styling:
    - The source styled text instead of plain text. This does require DEVONthink Pro Office v2.9.11+.
    - Items with only plain source text import plain text.
    - Plain text and markdown items imported from DEVONthink respect the (Tinderbox) note's default $TextFont and $TextFontSize.
  - Using DEVONthink Office Pro 2.9.15+, the note created imports the DEVONthink Creation time as $SourceCreated and the DEVONthink Modification time as $SourceModified.
  - $Tags import from tags in from items watched in DEVONthink folders.
  - Watched DEVONthink items populate $URL with the item's DEVONthink URL (previously it was DEVONthink's Note ID).
  - Dragging image items from DEVONthink to Tinderbox creates a note containing the image.
  - DEVONthink text items that exceed ~10,000 words are not automatically imported.
  - Dragging Freemind (.mm) items from DEVONthink to Tinderbox imports the Freemind maps. Previously, the DEVONthink item, had to be dropped onto the desktop first, in order to create a file.
  - also see Dragging content from DEVONthink to Tinderbox.
- **Watched Group**. A watched group allow for the import of a single group of DEVONthink records. The group's items are generated as child notes of the watched group container in Tinderbox. Multiple watched groups are allowed though be mindful heavy use of this feature may affect overall performance, i.e. *the feature is not designed as a fully-synced system* but rather for importing/updating some DEVONthink data. Notes created via this method are **read-only**, inheriting $ReadOnly state from their 'Imported From DEVONthink' built-in prototype.
- **Export**. Tinderbox notes can be copied/pasted to DEVONthink. Copy the note(s) in Tinderbox. In DEVONthink use Data menu, New ▸ from clipboard (Cmd+N). The following Tinderbox data maps to DEVONthink:
  - $Name: a new RTF item title.
  - $Text: text of the new item.
  - $Tags: the item's tags
  - $NoteURL (source note's Tinderbox pseudo-protocol URL): the item's URL
- The DEVONthink Item ID.
- The DEVONthink URL pseudo-protocol.

DEVONthink items dragged into Tinderbox import their DEVONthink label string into the attribute, $DEVONthinkLabel.

DEVONthink import uses a dedicated built-in prototype (as with watched DEVONthink folders). This allows customisation of Tinderbox's behaviour when importing from DEVONthink v3.

When dragging an item from DEVONthink, Tinderbox will populate the new note's $SourceURL with the URL of the dragged resource.

More on DEVONthink integration

- DEVONthink Item ID
- DEVONthink URL pseudo-protocol
- Mapping Tinderbox and DEVONthink URLs

DEVONthink Item ID

A DEVONthink 'item link', which uses the `x-devonthink-item://` pseudo-protocol, can be used to obtain the source item's ID. The item link looks like this:

`x-devonthink-item://5349A993-377B-4F63-9E43-74317939529B`

The long code string is a DEVONThink unique ID for that asset. Thus the actual item ID is:

`5349A993-377B-4F63-9E43-74317939529B`

This is the ID needed to use in a DEVONThink watched group.

DEVONthink URL pseudo-protocol

DEVONthink's 'item link' use the protocol `x-devonthink-item://` which allows other apps *on the same Mac* to open the item in DEVONthink from a URI held in another app, e.g. Tinderbox note's $URL. It could be any URL-type attribute but $URL should be used if it is intended to auto-fetch the DEVONthink item's content.

The basic link looks like this:

`x-devonthink-item://5349A993-377B-4F63-9E43-74317939529B`

The long code string is a DEVONThink unique ID for the source asset, whether stored within DEVONthink or just indexed by it.

Customised URLs

The item link URL format also allows several optional forms of suffix to these URLs to further customise the data returned.

'page='. `?page=4`. This will open (paginated) documents scrolled to the indicated page. Note however that the index in the URL is zero-based, so page #1 is '0', etc. Thus `?page=4` will open the fifth page of the document (i.e. 0->1->2->3->**4**). Example:

`x-devonthink-item://5349A993-377B-4F63-9E43-74317939529B?page=4`

'search='. `?search=Albert Einstein`. For text documents, DEVONthink will search the document, highlight all the matches and scroll to the first one. Note that the example given may not work as it is not URL Encoded, better would be t use `?search=Albert%20Einstein`. For PDF documents, they must have a text layer, e.g. scans saved as PDFs but not OCR-ed will not work with this method. Example:

`x-devonthink-item://5349A993-377B-4F63-9E43-74317939529B?search=Albert%20Einstein`

'time='. `?time=80.4`. For movies files such as MOV, MP4, this will open the movie at the time code, in seconds, as given. Thus to open a movie at 1 minute 20 seconds, use 80 seconds. Decimals of seconds are allowed. Apparently the method does not work for audio files like MP3, but this may change in due course. Example:

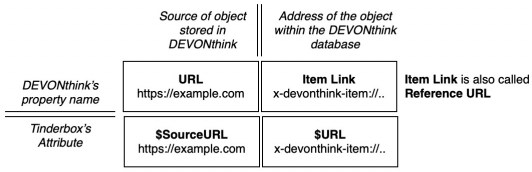
`x-devonthink-item://5349A993-377B-4F63-9E43-74317939529B?time=80.65`

Whilst the Edit menu and right-click in file listing in DEVONthink will copy a 'bare' item link (i.e. with no special suffix as above), right clicking on display of a partially played movie will add a `?time` suffix. Similarly, right clicking the scrolled display of a paginated document will append a `?page` suffix.

Mapping Tinderbox and DEVONthink URLs

Both Tinderbox and DEVONthink can store external URLs, i.e. links to external sources, and offer pseudo-protocols (usable on the same Mac) to call the app's record from another app or process.

The image illustrates the relationship of these URLs and how they map between the two applications.



Evernote

[This support was suspended in v9.0+ due to changes in the way the Evernote app works.]

It is possible to import small amounts of Evernote data via a watched folder looking at a particular notebook.

Finder

Tinderbox can automatically import selected notes from any Finder folder via a Watched Folder, including folders in Dropbox or on other remote servers.

FreeMind

Tinderbox opens FreeMind ".mm" files and will accept FreeMind files dragged into a map.

If a FreeMind item has an associated note, that note is imported as styled text in the Tinderbox text field.

IAWriter

Import from IAWriter is handled correctly.

Marked2

Tinderbox automatically communicates with the streaming preview of the application Marked2. When you select or edit a note using the Markdown prototype in Tinderbox, it will automatically be sent to the Marked2 streaming preview window.

Note it is a $Prototype value of 'Markdown', **not** the prototype's attribute customisations that trigger preview streaming.

Export code operators in the note $Text, e.g. ^value()^, are evaluated in the data passed to Marked 2.

More on Markdown use in Tinderbox.

Microsoft Word

Tinderbox does not support import/export with MS Word's DOC binary format. As RTF is also not supported the best choices are:

- Import
  - Plain text. Export to TXT and add via drag drop.
  - Styled text. Copy/paste between apps.
- Export
  - Plain text.
    - HTML export but with templates set not to add HTML and export file extensions set to '.txt'. More flexibility.
  - Styled text. Export well-styled HTML and open in Word.

Note that some text styles cannot be exported.

Microsoft Word® files (.doc) and Word XML files (.docx) may be dragged into Tinderbox preserving source text and tabular styles in Rich Text form. Documents with complex styling or much embedded content may not import all source information; images embedded in Word documents are not imported. If in doubt, first test with simple text documents.

Notes

Data can be imported from the Apple Notes application (installed on all Macs) via a watched folder. Using iCloud sync on macOS and iOS Notes, this allows for import of data from iOS devices as well as Macs.

OmniFocus

Date imported from OmniFocus includes the notes' URLs (to $URL) and sets the notes prototypes to the built-in 'Task'. The import uses the OmniFocus' task's key dates: Defer Until becomes $StartDate, Due becomes $DueDate, and Completed becomes $EndDate. Any mapped Date attributes unspecified in the imported data are set to a value of "never". If a task was completed before the present time, the note is marked as $Checked. If the task has a note, the text of note is placed in the Tinderbox note's text.

OmniOutliner

Tinderbox supports drag-drop importing of OPML exported from OmniOutliner. Sadly, Omni neglect to document their export format in any detail. A scant description can be found here.

Column-to-Attribute mapping

The first column of an OmniOutliner file (default column title 'Topic') will export as OPML attribute 'text' (regardless of the columns title). An OPML 'outline' tag (i.e. row/note) 'text' attribute always maps on import to the Tinderbox $Name attribute of a note (i.e. its title).

If the OmniOutliner column #1 content contains more than one paragraph, the only first paragraph only is exported as as the 'text' attribute, with the rest of the text exporting as the OPML '_note' attribute. Tinderbox always maps '_note' to a note's $Text attribute. For more deliberate export it is possible to put only the row/note title in column #1 and make a column #2 with the column name '_note' and place the intended Tinderbox note text in that column.

All other columns are exported with a custom OPML attribute that matches the column header name. Thus if targeting import to Tinderbox, these columns should use the *case-sensitive* name of the intended mapped Tinderbox attribute. Thu if aiming to map a column to $SomeAttribute, the OmniOutliner column name should be 'SomeAttribute', and so on.

In summary Tinderbox always maps OPML 'text' to $Name' and '_note' to $Text while any other attribute names (as derived from OmniOutliner column headers) are case-sensitively mapped to an existing Tinderbox attribute or a new user attribute of that name is created.

Regardless of the OmniOutliner source file's column data type, Tinderbox OPML import will always create new Tinderbox attributes as a String type. Thus it is advisable, where possible to define any desired mapped attribute in the TBX file before import, setting the desired.

Pasting an outline from OmniOutliner understands notes associated with OmniOutliner items and saves them in the new note's text.

Data Types

OmniOutliner and Tinderbox data types do not match exactly. The default column type is '**rich text**' which is exported to OPML *without* any styling. The '**number**' type in OmniOutliner allows for various format but is best exported unformat whether targeting a Tinderbox String or Number type attribute.

Use of the '**checkbox**' column type is not recommended as whilst it renders in OmniOutliner like a Tinderbox Boolean attribute the values are exported as 'checked' and 'unchecked' both of which parse into a value of `true` (ticked) if mappe to a Tinderbox Boolean. Thus it is better to use a 'rich text' type column and 'true' and 'false' values as these will pass correctly into a Tinderbox Boolean attribute.

The '**date**' column type displays/exports the short date form 'dd/mm/yyyy' or 'mm/dd/yyyy' the day/month order depending on your OS' locale. Further the delimiter may be other that a '/' depending on your OS' locale. Mapped to a Tinderbo Date attribute this will work fine as long as both OmniOutliner and Tinderbox are running on the same OS locale. If the source OmniOutliner OPML and the Tinderbox app are on different locales (e.g. US vs. non-US) you may need to format th OmniOutliner date column to use the day/month order and delimiter of the recipient OS with the Tinderbox. Note OmniOutliner 'date' columns allow some locale based formatting: see the OmniOutliner manual.

Other OmniOutliner column types are not recommended if creating data intended for export to Tinderbox via OPML.

For multi-value, i.e. list, data values, use a 'rich text' OmniOutliner column and format the text so each value is delimited with a semi-colon, i.e. 'value 1, value 2' should be saved before export as 'value 1;value 2'. On import to a Set or List typ Tinderbox attribute this will separate into discrete list values. If imported to a String attribute with is then changed to a Set or List, again it will parse into discrete values.

With appropriate data values, number/boolean/date/list type data imported to Tinderbox as a new string-type attribute should translate correctly if the new attribute is then changed into the correct Tinderbox data type via the Inspector.

QuickCursor

Tinderbox is compatible with QuickCursor, an inexpensive accessory available in the Mac App Store. Written by Hog Bay Software, QuickCursor lets keyboard-centric users switch easily from Tinderbox text windows to edit text, or selected passages of text, in BBEdit, TextMate, WriteRoom, or a variety of other plain text editors.

RIS files

Files with the extension '.ris' are understood to contain RIS bibliographic data, which can be parsed to appropriate system attributes in the References group. This improves the handling of references to journal articles and conference proceedings imported from Bookends, Zotero, and other sources of RIS data.

Dragged references hold this information in $ReferenceRIS.

The references type (article, journal, book, etc.) is extracted from the 'TY' field and placed in $RefType. See more on the allowed values for RIS code 'TY' .

Author import: $Authors respects the order of the received source list (previously it was re-sorted in alphabetical order,).

TY type codes for RIS

The RIS code 'TY' values can be found in the table at Wikipedia. Note that Tinderbox extracts the TY value but doesn't convert the encoded value to its more human readable version. The allowed values for RIS code 'TY' are as below:

Abbreviation Reference Type

`ABST` Abstract
`ADVS` Audiovisual material
`AGGR` Aggregated Database
`ANCIENT` Ancient Text
`ART` Art Work
`BILL` Bill
`BLOG` Blog
`BOOK` Whole book
`CASE` Case
`CHAP` Book chapter
`CHART` Chart
`CLSWK` Classical Work
`COMP` Computer program
`CONF` Conference proceeding
`CPAPER` Conference paper
`CTLG` Catalog
`DATA` Data file
`DBASE` Online Database
`DICT` Dictionary
`EBOOK` Electronic Book
`ECHAP` Electronic Book Section
`EDBOOK` Edited Book
`EJOUR` Electronic Article
`WEB` Web Page
`ENCYC` Encyclopedia
`EQUA` Equation
`FIGURE` Figure
`GEN` Generic
`GOVDOC` Government Document
`GRANT` Grant
`HEAR` Hearing
`ICOMM` Internet Communication
`INPR` In Press
`JFULL` Journal (full)
`JOUR` Journal
`LEGAL` Legal Rule or Regulation
`MANSCPT` Manuscript
`MAP` Map
`MGZN` Magazine article
`MPCT` Motion picture
`MULTI` Online Multimedia
`MUSIC` Music score
`NEWS` Newspaper
`PAMP` Pamphlet
`PAT` Patent
`PCOMM` Personal communication
`RPRT` Report
`SER` Serial publication
`SLIDE` Slide
`SOUND` Sound recording
`STAND` Standard
`STAT` Statute
`THES` Thesis/Dissertation
`UNBILL` Unenacted Bill
`UNPB` Unpublished work
`VIDEO` Video recording

Scrivener

A pair of built-in (OPML) export templates specifically marked for Scrivener use were added. The templates are effectively the same as those for general OPML export but labelled so as to assist the more tech-averse Tinderbox/Scrivener use looking to export data from one to the other.

A File menu option allows direct import of Scrivener file data to create a new Tinderbox document:

- Each text scrivening and folder becomes a Tinderbox note; v2 Scrivener files only. Text of v3 files is currently not extracted to a Tinderbox note.
- Image, pdf, and Web scrivenings are currently ignored.
- Scrivener keywords are imported to a set attribute $ScrivenerKeywords, and the corresponding label colours become the note $Color.
- Scrivener statuses are imported to $ScrivenerStatus.
- Scrivener freeform corkboard positions are imported to $Xpos and $Ypos. If Scrivener is not using freeform corkboard positions, Tinderbox uses its own layout.
- Scrivener custom metadata is mapped to Tinderbox attributes. Because Scrivener custom metadata labels may not be valid Tinderbox attribute names, Tinderbox uses 'legal' version of Scrivener's internal metadata names; the attributes are found in the Scrivener system attribute group.
- The Scrivener synopsis, if present, becomes the note's $Subtitle.
- Because Scrivener values result in many items' $Color value being set to light colours, the new TBX's Map preference is set for Outlines to have "Darker colors".
- Scrivener internal references are loaded as basic links. The default link type "[InternalLink]" is mapped to ""untitled".
- If an external reference link is found, its URL is placed in the note's $URL attribute. (If several external references are found, only one URL will be imported.

Simplenote support

[NOTE: Simplenote sync support was discontinued from v8.8.0]

Notes below are for legacy purposes only

Synchronising notes with Simplenote, a note-taking app for iPhone/iPad with web synchronisation. Tinderbox synchronises with the web app and the web app with iPhone/iPad. The web app is free and device apps have free (with unobtrusive ads) and pay-for versions. Integration uses the Simplenote v2.0 API giving support for tags and improving reliability.

Simplenote sync, and the account details to use are set in the Document-level Preferences' Simplenote tab.

If Simplenote syncing is enabled for a document, Tinderbox syncs when the document is opened and saved. Notes synched to/from the web are placed in a root-level container called 'Simplenote'. *Do not move this container whilst synching activated* as the synch process will see no notes where the container should be and will delete all Simplenote notes.

Simplenote support is added to a Tinderbox document by opening Document Preferences, Simplenote pane. Tick the 'Check Simplenote' box and enter the Simplenote account name (typically an email address) and password. It is also possible to filter the notes that will sync by nominating one or more Simplenote tags in the preferences.

Tinderbox will download copies of that Simplenote account's notes to a root-level container named 'Simplenote' (case sensitive) whenever the document is reopened or saved; the container is made if not already present. On-demand synchin from TB is achieved only by saving the host TBX.

New notes and notes that have changed on an iPhone or iPad and been web-synched will be added as needed. Notes deleted from an iPhone or iPad and web-synched are removed from Tinderbox's Simplenote container.

Simplenote notes do not have separate titles, as do Tinderbox notes. Tinderbox attempts to propose a useful title from the Simplenote text.

- If the note received from Simplenote is short, TB puts the Simplenote text in both $Text and $Name.
- For longer notes, the full text goes into $Text, and the first sentence or line goes into Title. When syncing, TB syncs (only) the text.
- If a note is made in the TB Simplenote container and it has no Text, Tinderbox uploads the $Name as if it were the note text.
- For notes first created in Tinderbox do not use any punctuation symbols in $Name. Doing so results in a Simplenote using only $Text (if any) or, if no $Text, the $Name up to the first punctuation character.
- Aliases in the Simplenote container are not synched.

Only immediate children within the Tinderbox Simplenote container are treated as potential Simplenotes when synching. In other words, any grandchildren (or deeper) notes in '/Simplenote' are ignored for synching. If a note with children is deleted via synching, the note *and all its descendants* are deleted.

As part of the process, two attributes are used. $SimplenoteModified keeps track of the effective modification date according to the Simplenotes server. $SimplenoteKey is used to store the internal key that identifies a note to the Simplenote server. If a note deleted in Tinderbox is in the 'Simplenote' container and has a $SimplenoteKey, Tinderbox will ask the server to delete the note.

A Simplenote Preference 'Required Tags' (a semi-colon delimited list of tag names) adds an over-riding filter so that only those Simplenote notes with tags included in the list are imported to Tinderbox box and subsequently synced (all other notes are ignored by Tinderbox.

$SimlpenoteTags stores a synched note's Simplenote tags. Two further attributes, $SimplenoteSync and $SimplenoteVersion are added for internal Tinderbox use: users should not edit them.

The Simplenote synchronisation dialog shows the overall number of notes being synched, plus a breakdown of the number to upload, download and delete; a progress bar is also shown.

If a note that is synched with Simplenote is deleted, its Simplenote key is cleared. If the deletion is undone, a new note corresponding to the Tinderbox note will be uploaded, so the deletion is effectively undone.

Aliases. Tinderbox handles the scenario where making an alias of a note inside the Simplenote container could cause both the alias and its original to be deleted from both Simplenote and Tinderbox at the next sync.

Tinderbox is alert to the possibility of Simplenote notes containing control characters (other than Carriage Return, Line Feed and Tab). Previously, syncing such notes to Tinderbox could create files that cannot be reloaded. It is not clear that Simplenote notes should contain control characters, but the app is playing safe.

Added better support for synching notes containing accented characters and other Unicode issues as well as using improved synch code.

Storyspace

The System attribute group ' Storyspace' accommodates interchange use of Storyspace v3 files. For more on the inter-relation of Tinderbox and Tinderbox see Tinderbox's pre-history.

The Storyspace product page; http://www.eastgate.com/storyspace/.

TaskPaper

TaskPaper files may be dragged into Tinderbox. Tinderbox automatically recognises TaskPaper tags and copies them to $Tags. The @done() tag also sets $Checked and $NameStrike; if the @done tag has a completion date, it is copied to $StartDate and $EndDate.

Recognises and maps @due (to $DueDate), @start (to $StartDate) and @done (to $EndDate)

Tot

Tinderbox can watch and automatically import notes from the Tot application, which can installed in macOS and iOS devices.

Watching Tot will import text from Tot, but does not sync *to* Tot.

Text line endings

This topic likely will not affect users unless they are doing multi-line edit scripting or using the Explode feature.

Not all line breaks are equal. 2 different characters get used for line endings:

- Line feed. ASCII character #10. The designator for Explode, regular expressions, etc., is `\n`.
- Carriage return. ASCII character #13. The designator for Explode, regular expressions, etc., is `\r`.

To muddy the waters, different OSs have used different line end encodings:

- Classic Mac OS: carriage return (\r).
- Unix, including macOS: (\n). (So, some longstanding Mac apps may still use \r as originally in Classic Mac OS.
- Windows uses a 2 character method: carriage return + line feed (\r\n).

Text generated in Tinderbox (except perhaps very early versions running pre-macOS) uses the Unix/macOS style line offered style of line end, \n.

When including a line end as part of an Explode delimiter, \r and r generally have the same effect so using \n as a 'default' is probably good working practice.

However, depending on the source of text brought into Tinderbox, e.g. text file dragged in for exploding, the designator used for line end may have to be adjusted.

Time - Displaying and Setting Seconds

In v5+ Tinderbox date/times effectively have, but do not use, a seconds elements:

- If formatted for string use, seconds always show as '00'.*
- Date/time creation operators omit a seconds input or ignore it if used.
- Date/time comparisons cannot be made to seconds' granularity as all date/times have the same (00) seconds value.
- There is no method to return just the seconds of a date/time (QV hours or minutes)
- Displayed Attributes date/time values do not show the seconds element (unless the user has her system set to use a format that includes seconds.

The rest of the article refers to Tinderbox *before* version 5

Do Date attributes have seconds?

All Date (i.e. Date/Time) attributes inherently include a time down to seconds level, even if it is not always displayed. However, confusion arises as Displayed Attributes, i.e. as shown in the top of a note's Text window, cannot be made to display the seconds for a Date attribute; instead they always show the OS short date with hours and minutes only. In addition, some process default to using seconds of '00' whilst others use the seconds as at time of execution.

Which operators do not use seconds in comparisons?

When comparing dates or doing date arithmetic, be aware of the seconds unless using days() or minutes() which compare in whole days or whole minutes respectively. Also a positive or negative equals query (== or !=) will ignore time completely and match on whole days, like the days() operator.

Can I display a Date attribute in Displayed Attributes and see its seconds?

No. At least, no without altering your OS settings: see below.

Which date designators use the second of execution?

The designators *yesterday*, *today*, *now*, and *tomorrow*, plus any dates based on modifications of those designators will set the seconds as measured at time of setting the attribute.

Once a date/time has a non-00 seconds time, any modification to the value will preserve the seconds unless the modification expressly sets the seconds.

What time is set if a date is typed into a Displayed Attribute box?

Unlike the above, the hh:mm are taken from current time but the seconds are set at 00.

Can I type in a date and set seconds?

Yes, you will need to set your OS Time 'short' setting to show hh:mm:ss; otherwise, when you provide a seconds the data is ignored at seconds are set at 00. "20 January 2010", at 17:45 local time will set "20/01/2010 17:45:00." For the same

time "20 January 2010 17:45:25" will set "20/01/2010 17:45:25".

Import

AutoFetch

If a note's $AutoFetch value is `true`, Tinderbox automatically updates the note's $Text to use data drawn from the source stored in the note's $URL. The latter may be any of:

- URLs accessible via the Web.
- locally installed app pseudo-protocol URLs, e.g. "devonthink://…", "bookends://…", etc.
- local URLs, e.g. "file:///…".

The AutoFetch process facilitates automatically updating note $Text from other text sources. As well as the more obvious sources such as web pages or RSS feeds, AutoFetch works with a range of file types, including text, rtf, Microsoft Word®(.doc), pdf, and markdown (.md) files.

Once AutoFetch has run, the note containing the retrieved data also has its $ReadOnly attribute automatically set to `true`. This avoids the user trying to edit text that will get automatically overwritten when AutoFetch next runs.

Including the first setting of the process, when does AutoFetch run on a note and alter its $Text:

- on (re-)setting $AutoFetch to `true`.
- on document load, all existing AutoFetches run once.
- on (HTML-type) Export, all existing AutoFetches run once to refresh the exported content.
- while the document is open, no more than once every 5 minutes as part of the general agent & rules cycle. The time to the next cycle is shown in the circular 'Network' display on the the Agents & Rules tab of the Tinderbox Inspector.

For deeper, more complex, be aware of the experimental attribute $RawData, which holds existing raw data (not post-processed) should the next planned fetch fail.

Post-processing AutoFetched data

It is possible to process imported data, *before* it is added to the note's $Text. This is done by setting valid action code in the same note's $AutoFetchCommand, whose operation is described separately in AutoFetch commands.

AutoFetch commands

To assist in manipulation of AutoFetched data is is possible to set an action to run on the raw output of the data before it is set as the note's $Text. It can be thought of as if it were an 'OnAutoFetch' action. Such actions are set in an Action-type attribute $AutoFetchCommand, and thus use the same syntax as agent actions and rules.

If $AutoFetchCommand is empty (the default), $AutoFetch behaves as normal; the contents of the specified URL are stored in the note's text.

$AutoFetchCommand has code set, that code is run on this note (i.e. the same note holding this code) immediately after a network AutoFetch event has been performed (event triggers are described under AutoFetch).

Historically, these actions have been command line calls but as Tinderbox's range of text-manipulation operators in action code has expanded, pure action code may suffice. For instance, if pulling stock ticker information, it is possible to look for a line like "AAPL: 145.64 USD" and set the price to a Number-type attribute $StockPrice. This could be done like so:

```
$StockPrice = $Text.following("AAPL: ").replace(" USD","");
```

An AutoFetch command might invoke an outside program, perhaps fetching some information from the user's hard disk or network, using the runCommand() operator. For example:

```
$Text = runCommand("echo | ls ~"/Documents);
```

will replace the $Text of the note with a list of all the files currently in the user's Documents folder (the latter syntax is the recommended choice).

```
$Delivered = runCommand("myDatabaseQuery $TrackingID");
```

which will run the shell script 'myDatabaseQuery' in the user's OS home folder, passing it the value of the note's $TrackingID attribute as an argument.

When accessing the command line this way the current working directory is '/', i.e. the root of the current volume (more on the assumed working directory). Thus if calling scripts elsewhere (e.g. in a user account's 'Documents") remember to prefix an appropriate path to the script.

Drag from $Text to Outline or Map to create a note

Dragging a text selection in $Text pane from a text window and dropping it onto an Outline or Map view results in a new note being created:

- The dropped text selection is used as both $Name and $Text of the new note
- Application focus remains on the source text window
- No links are created (unlike when to making footnotes).
- The major view's focus shifts to the newly created note.
- Dropping on a Map view:
  - The note is created at the drop point
  - Dropping onto (into) a map icon viewport results in no note being created (not a note being created on the child map as might be supposed).
- Dropping on an Outline view:
  - Onto the general background, the note is added as first sibling of the current root level notes.
  - If dropped over the right part of a note title the black position arrow will indicated whether the new item will be created before, after or as a child of the item under the cursor, i.e. the same behaviour as drag re-arranging notes in Outline view.

Dragged folder links

Dragging a folder from Finder into Tinderbox creates a container and imports a note for each file in the folder. The $File attribute of the imported note holds the path from which the note was imported. The $LastFetched attribute of the imported note holds the modification date of the imported file.

Dragging a URL from web browser

What selections can be dragged from a web browser?

Whether Tinderbox receives both URL and a link title/anchor text is depends on the *browser*, Tinderbox will use both if passed, otherwise a bare URL is received.

Browsers do not allow dragging of a selection webpage content *unless* it includes at least one link (a visible URL or a link anchor text).

Clicking in the location bar of a browser, where the (partial) URL is shown will select it and allow it to be dragged.

Note that if dragged page content contains HTML such as lists of links, Tinderbox may try to parse these into a container holding two or more discrete notes. However, this article is intended to cover dragging content with a single URL into either the view pane (to make a new note) or into a note's text area.

Dragging a URL onto a view background

Drag a URL from Safari or other web browser onto a view window background makes a new note (in all view types except Attribute Browser, Crosstabs and Hyperbolic). The location of the new note varies by view type:

- **Outline**, **Chart**. By default the new note is the last top-level sibling (i.e. child of root or a hoisted root scope). By hovering the cursor close to a note, the new note can be a previous/next sibling or child of an existing note at any level at root or in a branch.
- **Timeline**. Always the last child of the root of the view. The new note will appear in the 'No Date' sidebar as it will have no $StartDate at this point.
- **Treemap**. By default the new note is a top-level sibling (i.e. child of root or a hoisted root scope). Essentially as for Outline but without the fine positioning option.
- **Map**. Creates a new note at the drop point. If the drop is *onto* an existing note, the new note is created as a child of that note.
- **Attribute Browser**, **Crosstabs**, **Hyperbolic** view. Drop data is not accepted, so nothing happens.

What note content is added for drag-to-view created notes?

If the drag is from the browsers location (URL) bar the dragged data is usually only the (full https://...etc.)

The drop action always sets $URL value, adds $URL as a Displayed Attributes, sets the note's title and its text. There can be some variations depending on the data supplied with :

- $URL is set to the dropped URL.
- The Displayed Attributes ($DisplayedAttributes) of the new note is set to "URL". Note that if the container of the new note has an OnAdd setting Displayed Attributes directly or via an attribute, this setting is overwritten to show $URL. Reset $DisplayedAttributes to restore the inherited value.
- The new note's $Name is set to the value of the source URL, unless a title (e.g. the source page's anchor text) is parsed out in which case the title is used in preference.
- The new note's $Text is set to the value of the source URL, unless a title (e.g. the source page's anchor text) is parsed out in which case the title is added as a weblink, pointing to the URL. in preference. If only the URL is added, no weblink results.

Dragging a link from a webpage onto a note's text

At the drop position, the receiving note's $Text is set to the value of the source URL, unless a title (e.g. the source page's anchor text) is parsed out in which case the title is added as a weblink, pointing to the URL. in preference. If only the URL is added, no weblink results.

The drop also sets window sets the note's $URL value (but will not overwrite any pre-existing $URL value).

See also results of pasting webpage data to $Text .

Dragging a link from a webpage onto a note's Displayed Attributes table

If the URL is dragged onto the Displayed Attributes *and* the drop target is a URL-type attribute, then that attribute *rather than* $URL receives the dropped URL as its value.

Resolving the link title

If the browser does not supply enough data, a drag to the text window results in the just the URL string being added to the note as text.

If the dragged object is a browser bookmark the note name will be taken from the bookmark. If the dragged object is a bare URL, Tinderbox will attempt to fetch the title from the URL and set the note name accordingly.

Dragging images from Safari

Dragging an image from Safari to a note window imports the referenced image as opposed to setting the URL attribute.

Dragging Address info into Tinderbox

Tinderbox accepts drags of vCards (vcf) from Apple Address Book and other vCard-compatible sources. A new note is created.

If a vCard is dropped on an existing note (icon) the data is added to the note and an enactment (of the text 'vCard') is shown. Dropping a vCard onto a Map or Outline view background makes a new note as in previous versions; dropping a new vCard note does not trigger an enactment helping clarify which process is occurring. The process (re-)populates the following attributes in the People attribute group (if suitable data is detected):

- $AIM
- $Address
- $Email
- $FullName
- $Organization
- $Telephone
- $Twitter

Also, the $URL of the (new) note will be (re-)set to the corresponding AddressBook entry. It uses a special URL of the form " `addressbook://AB8F8A3E-B4F2-4DE6-9D7B-C8C511B8CC1A:ABPerson`"; opening the the URL opens Addre Book at that item.

The vCard's raw data is added to $Text. If dropping on an existing note, the raw data is only added if there is no $Text, thus preserving any pre-existing text is not lost. The raw vCard data is also stored in the note's $RawData for that note unless/until AutoFetch is also used by the note (N.B. AutoFetch cannot be used to refresh vCard data).

Dropping a vCard on a note sets the prototype 'Person' if a prototype of that exact (case-sensitive) name exists. If no prototype match occurs, none is applied. When dropping on an existing note, if a "Person" prototype is found, it is set and in doing so will replace any existing $Prototype setting to a different prototype.

The telephone number chosen by Tinderbox will be the source data's *preferred* telephone number.

To drag-drop add vCard data or refresh existing note data, the data must be dropped thus:

- Map. Drop onto a note icon. That is the title/text part, but not onto a data table, or—for containers/agents—the viewport map or plot area.
- Outline. Drop as if placing as a new, or first, child of the receiving note, i.e. so the triangular drop position indicator triangle is over the correct drop-target and points to the right.
- Chart. Drop onto any note (whether a leaf or branch of the chart).
- Timeline. Drop onto a note.
- Treemap. Drop onto a note.

In each case if the drop is correct, you will see an enactment: the view clears and the caption 'vCard' appears and disappears as the view re-draws.

Dragging content from DEVONthink to Tinderbox

From v9.6.0, when dragging an item from DEVONthink, $URL is set to the DEVONthink item URL if it has once. For example, if the item is a web page, $URL is set to the URL of that page. If the item doesn't have a URL, Tinderbox sets $URL to the DEVONthink internal URL to allow prompt reference to the item.

More on interactions with DEVONthink.

Dragging emails to Tinderbox

Dragging an email from an email client such as Apple's Mail.app, MS Entourage or MS Outlook will result in a new note. The email's subject becomes the note's $Name and the body copy becomes the note's $Text. Be aware that message headers, including to/from addresses, etc., are not imported.

Additionally, but *only* when dragging from the Mail.app:

- The new note's $URL attribute is set to the source email's Mail.app pseudo-URL (as ' `message:< … >`'). Note that the URL's characters on the `<` and `>` are actually stored in url-encoded form as `%3C` and `%3E` respectively; thus for example '`message:%3CE7B7A670-A8E3-4EEB-8921-3596825A334C@me.com%3E`'. Using the URL button in the Displayed Attributes table will open the source email in Mail.app.
- The imported $Text is also parsed for any Ziplinks method-style mark-up which is then evaluated and new internal text links are created if valid.

On first use of drag from Mail.app, newer OS versions will require the user to allow inter-app access (a once-only task). This presumably to enable verification of the email's source pseudo-URL.

Dragging HTML files to Tinderbox

When HTML files are dragged into Tinderbox, the $Text of the note created uses the HTML source's style information for formatting as styled text.

Dragging in Apple Calendar events

From v9.5.2, if an event from the Apple Calendar app is dragged or pasted into a Tinderbox view has invitees, the names of the invited people are listed in a system attribute, $Participants.

Whether individual participants are retrieved as the person's name or as their email address relates to the data passed by Calendar and is not due to actions by Tinderbox.

From v9.6.0, imported calendar events are assigned the prototype Event if the prototype exists, but only if the newly-created note wasn't already assigned a prototype by the OnAdd action.

Dragging TextClipping files to View pane

TextClipping-type files may be dragged to the view pane.

Dragging URLs into Tinderbox views

From v9.6.0, when dragging a URL into a Tinderbox view, the value of $DisplayedAttributes is set to 'URL' only if the OnAdd action of the parent container has not already assigned a value. This prevents the default action overwriting a specifi or prototype-inherited Displayed Attributes choice.

Dropping Text files into Tinderbox

This describes use of plain text files (e.g. '.txt.'). Rich Text (RTF) files are not supported for drag/drop

Dragging onto an Outline, Map view

Dragging a file onto a major view window creates a new note whose $Name is the filename of the dropped file. The new file is added at the top of the outline; i.e. $OutlineOrder position '1', or first in sibling order if the view focus is a container. Chart view does not support a general drop, but also see below. When a text file is dropped, Tinderbox will try and find embedded formats: see 'other types of data' below.

Dragging onto a note icon in Outline or Chart view

Dragging a file onto a note icon in a view window creates a new note whose $Name is the filename of the dropped file. Use the normal note-positioning tell-back to see see if a note is added as a sibling or child of the note under the drop. Not that Map view does not support this degree of placement.

In Outline view, dropping onto a note icon is just that: the icon at left and not its badge or title.

Dragging into a text window

Drop is not supported. Copy the note's contents and paste instead.

Other types of text data

Be aware Tinderbox handles drops of other textual data in particular ways:

- Browser URLs (including local pseudo-protocols used by DEVONThink, Bookends, Yojimbo and other apps).
- Address Book info & vcards.
- Emails.
- Tab-delimited or CSV tabular data. ('spreadsheet' import)
- HTML: this is teated as plain text, i.e. the resulting $Text shows the source code and not as rendered HTML.
- OPML files.
- Old version of the app supported drag-drop of tab-indented outlines but this is not supported in current v5 releases.
- Bookends selections.
- FreeMind '.mm' files.
- tabbed outlines (tab per outline level). Produces nested notes as per tabbed outline.
- iCal calendar events.

Text formats not supported

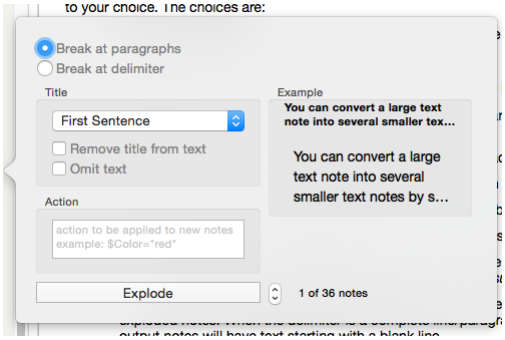
These formats *not supported* for drag/drop:

- RTF (Rich Text Format).
- Word-processing app-specific documents, e.g. for Word, Pages, etc.
- Scrivener project files, although they can be opened directly from the File menu.

Styled text such as RTF, word processor test, rendered HTML can be imported by copy/paste to the $Text of a note's text window; there is no automated method for multiple imports of such data.

Exploding Notes

A single note's text ($Text) can be 'exploded' (split) into a container holding a number of discrete smaller notes by specifying where Tinderbox should divide the current text. The newly created notes contain an appropriate segment of the original note, using part of the same text as the note title. The basic default behaviour is to split the source by paragraph, using the paragraph of text as the note's text and the first sentence of the paragraph as the note's title. However, there are a number of option, described below that control the title and content of the new notes. There will (should!) be no empty notes, such as those that might arise where two hard line returns are used as faux paragraph breaks, as these unwanted 'empty' outputs are automatically deleted as part of the Explode process. Empty notes will not be emitted for sequences of multiple paragraph breaks, i.e. sequences of greater than two breaks.



This process can be used to help with the import of data as well as for existing notes. For imports, drag, a *plain text* format file into an Outline or Map view, so as to create a new note. Note with that, or with a large existing note, selected in the view pane, choose **Explode…** from the Note menu. The option is available regardless of which pane has focus (prior to this, be *aware that the Explode menu option is disabled if the input cursor is in the text pane*).

Calling Note ▸ Explode presents the Explode pop-over. There are essentially three discrete sets of options to set before committing the explode:

- **Defining how/where the source is split.** This is defined by the top two radio buttons:
  - **Break at paragraphs**. Breaks each paragraph into a new note (default option).
  - **Break at delimiter**. Break on a (custom) delimiter. The input box for the delimiter is hidden when this option is not selected. The delimiter is a string of character(s) that is a regular expression that immediately *precedes* the break point. Tinderbox assumes each section of the source note that will form a new note *starts* with the delimiter. If there is source text *before* the first occurrence of the delimiter, Tinderbox behaves as if there were a starting delimiter and makes a new note use the content between source start and the first occurrence of the delimiter. Tinderbox remembers the last-used custom delimiter until the end of the current session. Use of delimiter regular expressions is described in more detail further below.
    - **Delete Delimiter**. If the custom option is chosen, there is a further option to delete the delimiter string, otherwise it is retained as the end of each new note's text (as the *end* of the delimiter string is the per-note break). When importing date for explode and using a deliberate string for delimiter purposes it is usually best to use this sub-option to avoid importing unnecessary data.
- **Selecting source of new note $Name and $Text** . The **Title** panel settings give control of what part of each newly split note's $Text is used to form the note's title ($Name). This is done at sentence or paragraph scope. A 'sentence' is delimited by a terminating period, exclamation mark or question mark. A 'paragraph' is delimited by a line break. By default, an explode-generated note uses the whole exploded section of the source note as its own $Text and that text's first sentence as the note title. Explode removes any leading or trailing whitespace from titles of exploded notes.
  - The scope of a new note's title can be altered:
    - **First Sentence**. Only the first sentence of the note text forms the new title (default). When running under macOS 10.14.x or later, Explode is substantially smarter about recognising sentences. It understands, for example, that "Dr. Perkins paid $10.00 to the U.S. Treasury." is one sentence, not five. However, if the user's current language locale does not support the new sentence-detection neural net, Tinderbox falls back to use the older sentence-end detection logic.
    - **First Two Sentences**. The first *two* sentences form the new title.
    - **First Paragraph**. Only the first paragraph is used for the new title. When importing custom data, such as lists of papers that often contain punctuation marks, this is usually the best option to choose.
  - When Explode constructs a note title, the title may extend up to 512 characters, and will include either the first line or the first $Text sentence of the new note—whichever is the best fit. If still over 512 characters the title is truncated at that point with an ellipsis.
  - Two tick boxes give additional control the $Text of the newly created notes. By default, both are un-ticked:
    - **Remove title from text**. If ticked, the text used for the above choice is deleted from the new note's body text.
    - **Omit text**. If ticked, this results in only a $Name being set for new notes and no $Text. If importing a list of data consisting only of data to form the title of new notes, this box should be ticked.
- **Actions to apply to newly created notes** . The **Action** box allows action code to be inserted that is applied to each newly exploded note by setting an $OnAdd for the 'exploded notes' container created by the process. If it is desired t apply a prototype *other than the default* (see below the section **'Exploded Notes' prototype** for more detail), simply set the action to set the desired prototype's name. Essentially, the default OnAdd action of the 'exploded notes' container is the OnAdd action (if any is set) for the 'Exploded Notes' prototype unless the user adds code to the **Action** box; depending on the inputs, this may result code from **Action** and from the prototype being applied to newly-exploded notes.
- **Example** preview. This panel Gives a preview of the $Name and $Text of the first exploded note. A control below the preview allows cycling through a preview of successive exploded notes.

Finally, the **Explode** button closes the pop-over and starts the Explode process using the choices set above. Focus remains on the exploded note. The Return keypress will act like a button press and start the Explode.

Custom delimiters

This is a regular expression but may be a literal string (i.e. the actual characters to match). For instance:

- `###` A newly exploded note will begin after each occurrence of the string'###'. The text matching is *case-sensitive*.
- `\n` A newly exploded note will begin after new line feed character. This is also the default for this option, it is essentially the same as 'break at paragraphs' overall explode default. [More on line end character codes.]
- `\r` A newly exploded note will begin after each carriage return or hard line break.
- `\t` A newly exploded note will begin after each tab, e.g. for tab-delimited content pasted into a note.
- `\,` A newly exploded note will begin after each comma, e.g. use for CSV content pasted into a note.
- `^\d{1,4}\ ?` A newly exploded note will begin after each line starting with a sequence of between 1 and 4 numerical characters followed by zero or 1 space characters.
  - Delete delimiter. Only available if the 'Delimiter' option is set (default = not ticked). Tick this to remove the specified delimiter from the new exploded notes. When the delimiter is a complete line/paragraph, ensure the string used includes the line return character at the end or some output notes will have text starting with a blank line.

If struggling with understanding regex, assume *any* space or non-alphanumeric character in a custom delimiter needs escaping by preceding it with a backslash (`\)`.

What results from an Explode?

The exploded (source) note itself remains unchanged and a new child container of the selected note is created, called 'exploded notes', and which contains the resulting new notes (i.e. the individual exploded notes are grandchildren of the source note). Each note contains as its text a section of the original text with the notes being titled according to the choices (described above) made before exploding. Using a secondary container for the notes might seem odd until realising this allows for the fact, more so for exploding existing notes than for new import, that the note being exploded might already have children; thus the new notes are separated from existing children of the exploded note.

Also notice that the 'exploded notes' container is added to the outline order after any pre-existing child notes for the exploded note, i.e. it is the last child (and thus easily accessible via the lastChild designator, amongst others).

The 'exploded notes' container does not inherit anything from the source note. However, the **Action** box allows the 'exploded notes' $OnAdd to be set afresh for each explode. Also, the 'Exploded Notes' prototype (see below) is added to th 'exploded notes' child container of the exploded source note. Either the action or the prototype can be used to read values from the parent note or to set values in the newly exploded notes.

'Exploded Notes' prototype

When a note is exploded the 'exploded notes' container has a built-in prototype ' Exploded Notes' applied. Note the different letter case from the 'exploded notes' container; this is deliberate to stop name collision in queries. If the prototype d the /Prototypes container does not yet exist, Tinderbox automatically created the container and adds the prototype to the document before applying it.

Note the case sensitive spelling of the post explode container's title vs. that of the prototype; this is deliberate to avoid name collisions, e.g. he prototype does not want to be matched by a query looking for 'exploded notes' container(s).

By default, the only customisation of this prototype is to add $ChildCount as a Displayed Attribute. The prototype makes it easy to set up things like counts or a specific set of OnAdd actions (rather than set an action in the dialog: this can be useful if there are a number of actions to be set. The ability to customise this prototype becomes useful if doing regular/repeated Explodes.

This prototype can be added manually, via the File ▸ Built-in Prototypes menu, for instance to customise it *before* using an explode. To 're-install' the default version, simply delete the any existing version of the prototype and re-add it via the Built-in Prototypes menu.

Working with exploded notes

Of course, the constant name of the container holding the new notes ('exploded notes') makes it easy for agents to find and work on the new notes' actions. Be aware that unless the 'exploded notes' is cleaned up or renamed after an explode, there may be more than one container of that name. A benison of this is that it is possible to write a generic agent to act on the contents of all explode results using an $AgentQuery:

```
$Name(parent)=="exploded notes"
```

By adding a preceding descendedFrom() query argument to the above query, the scope of action can be reduced to items descended from a given container.

```
descendedFrom(To Do) & ($Name(parent)=="exploded notes")
```

This might be done because of the nature of the resulting agent action to be applied or simply to reduce agent update cycle time.

In summary:

- 'exploded notes' receives all the source note's attribute values except $Rule and $OnAdd. If the 'Exploded Notes' prototype has been altered to locally set attribute values these will be used in preference.
- Individual exploded notes do not inherit attribute values.
- Agents or manual editing of 'Exploded Text' actions should be used to apply attribute values to exploded text.

The explode action is applied *after* the text of the newly-created note is set, allowing the action to modify or depend on the exploded text. Explode also remembers the most recently-used delimiter, which can be convenient when using complicated regular expression delimiters.

Formatting dates for import to Tinderbox

Tinderbox handles import, generally via drag drop onto a major view, in a variety of ways. In some cases, such as tab-delimited text (also called TSV or 'spreadsheet' format), Tinderbox will scan the text and map some data to Tinderbox date attributes.

In order for source text to map correctly to a Date-type attribute, bear in mind that Tinderbox will look for dates in your source data that match **your Mac's System 'short' date and time format**. Be aware that such formats will vary by local and may be further customised by the OS account user. Tinderbox will also happily parse dates input in any of you locale's Short, Medium, Long or Full formats.

If in doubt if in doubt as you your Mac's preferences check 'System Preferences'. Currently it is under 'Language & Region'.

The OS preferences list date and time formats separately. If your import source date includes both date *and* time, ensure it leaves a single space character between the date and the time segments.

When the import source data is passed to a Date attribute, if the result is wrong the resulting Date value will be 'never' or some incorrect date; the later generally occurs if part of the date is in the wrong format and coerces to an unexpected value. There may be no source data still in the TBX file to tell you what text string caused the wrong date to be created. Therefore it makes sense to ensure import data is in the correct format *before* import.

That said, Tinderbox—via undocumented internal logic— *may* detect other common date formats. You can test raw data input and if it works, then that's fine, though you would be wise to check a range of specimen days months and years. However, for more than the occasional import, by such means it will repay you to ensure the import using the formats described above.

A special exception is OPML import. The OPML standard stipulates that dates, used to define the note's creation date/time, should use the RFC 822 format. Tinderbox will make the same assumption for date formats for the 'created' tag attribute. Tinderbox does *not* attempt to parse dates in the note's title or text.

RFC 822 and year format. Note that RFC 822 actually specifies 2-digit years whereas 4-digit is more appropriate (indeed the later RFC 1123 makes it so). Tinderbox's own RFC 822 format string (" **") uses 4 digit years and most OPML capable tools will likely use 4-digit years too. If dates import incorrectly via OPML, check you see if they are using 2 digit years and if necessary correct the source.

Time. If a date is supplied with no time, Tinderbox will append the current system time in hours:minutes:seconds (although the seconds element is usually suppressed in the Tinderbox display). An exception is if the supplied data is in the OS locale's 'Full' format and has no time, in which case the time is set at 12:00:00.

Importing from other file types and applications

Tinderbox supports import of general text formats like plain text, RTF, HTML, etc. Formats specific to particular applications, e.g. citation information, may be supported: see the section on support for other app-specific formats.

Importing Markdown files

From v9.5.2, files with extension .md, .mmd, and .markdown that are dragged into a Tinderbox view are now converted to styled text using Markdown conventions.

To avoid Markdown to RTF style conversion occurring, set the file extension to '.txt'

Inserting images into note $Text

An image (see below for types) can be dragged or pasted into a note's $Text.

To add an image to a note, select the note so it is shown in the text pane. If inserting into existing $Text, create some vertical space between paragraphs so as to make a clear insertion point.

In Finder either:

- copy the image and paste where desired into $Text
- select the image and then drag the selected image to the desired insertion point in $Text.

If an image pasted into the text is wider than available space, it will automatically be scaled.

If dragging from the Desktop, ensure the image is selected *before* starting the drag otherwise the image may not be imported.

To align the image, for instance to centre it, it is necessary to select the 'text' surrounding the image and use the Format, Text menu to set the alignment.

Supported image formats

The best form of image is to use are bitmap formats such as PNG or JPG (JPEG). Images embedded in PDF can be used.

Animated formats such a GIF or MP4 can be inserted; GIFs will auto-play whilst MP4s will show player controls when moused over.

Natural Language Processing

The text ($Text) of notes is scanned to extract information that might be useful for agents. These results include:

- $NLNames: a set of personal names found in the text.
- $NLOrganizations: a set of the names of organisations found in the text.
- $NLPlaces: a set of place names found in the text.
- $NLTags: a set to hold annotations automatically generated using Natural Language Processing. The first such annotation adds the tag 'plan' to notes that Tinderbox believes might represent a planning note, such as "remember to deposit the cheque" or "remind the freshers to begin planning their module essays". At present this process is not enabled.
- $DominantLanguage: Tinderbox's guess at the primary language used in the text of each note; stored as two-letter ISO-639-1 codes such as “en” for English, “de” for German, and “zh” for Chinese.

Note that these result values are extracted automatically and are thus subject to a variety of extraction errors.

Results are extracted from the $Text of:

- New notes.
- Re-edited notes, including pre-existing notes in old TBXs that have be re-edited.

Values are **not** extracted from pre-existing notes, such as in files created, unless they are subsequently re-edited.

This process may store some working files in the application support ' analytics' folder. Such files should **not** be edited by users.

OPML Import

Tinderbox imports OPML files dropped onto Tinderbox views, and also reads OPML outlines. Note that Onmigroups apps, e.g. OmniOutliner, popularised OMPL use but did so using *non-standard* additions. So when using OPML, *do not assume all OPML is the same*: read the OPML specification.

When Tinderbox parses a pasted or drag-dropped OPML, the source OPML filename is used as the name of the container holding the tree of notes parsed from the HTML. The OPML source code is *not retained* in the TBX document as the data is now imported and the OPML code would simply constitute unwanted file bloat. If the user also witness to store the OPML code, that would need to be pasted into a new note's $Text to avoid triggering the parser.

If an OPML file has extended outline attributes, and if the extensions do not conflict with Tinderbox attributes, then Tinderbox will make a user attribute for each of the extended attributes. The default 'text' attribute is mapped to $Name [sic]. The non-standard '_note' attribute popularised by OmniOutliner is, if found, mapped to $Text with '
' being parsed as paragraph breaks.

The user attributes will initially be defined as **Displayed Attributes** as well, allowing for easier inspection of the extended data.

OPML support includes Projekt extensions.

The non-standard but generally used OPML '_status' attribute data is mapped to $Checked which of course can be displayed in outline view as a tick box (good for 'to do' lists).

OmniOutliner exported OPML: see further detail.

Note that the OPML `<expansionState>` element is not supported. By default, any new container notes created from the imported OPML data are all collapsed in Outline view.

Tinderbox will expect dates to be in RFC 822 form in accordance with the OPML spec: see more.

Pasting a child to childless notes

This mainly affects work in Outline and Chart views. When pasting notes, either within or between TBXs, it it may often be the case that the paste location needs to be as the child of existing note the currently has no children. With paste, as opposed to drag/drop, there is no way to indicate via the cursor whether the incoming note(s) are to be added as sibling(s) or child(ren). There are two possible approaches. In both cases select the to-be-parent note before:

- Pasting, to create siblings. Then select pasted notes and hit Tab to indent them within the previous sibling (making it the parent).
- Add a new (temporary) child note, with any name, to the to-be-parent note. Select this new note this new note and do the paste. Delete the temporary note.

There is a subtle difference between the two methods. The second only invokes the $OnAdd of the intended new container. The first method invokes the latter and the $OnAdd of that container's parent as the pasted content is initially pastec in as a sibling of the containers and so gets the $OnAdd action of their mutual parent.

If $OnAdd actions are being used or the code used is trivial, then both methods are equally good. If there are significant location-dependent $OnAdd actions, the second method is the best one to use.

Pasting and Text Margins

When pasting text between Tinderbox documents with different margins, the newly pasted section of text may show a different margin. This difference will remain, even if the selection if re-formatted as 'plain'. Re-formatting is not needed; to re-align everything simply close and re-open the note and the margins will be correctly aligned.

Pasting into Tinderbox and line ends

Tinderbox uses the data on the clipboard. Users are sometimes confused when pasting text from other sources to get fewer or (usually) more line ends in the resulting text than they expected.

The most common reason is that some source formats, PDF being a good example, store lines of text as just that—separate runs of text. This means that although a paragraph of text at source can be selected and copied in a tool like Preview, the data pasted is a set of lines of text not a single run of paragraph. In the same scenario a copy of multiple paragraphs will still be a set of text lines with no obvious paragraph delimiter. This is a 'fault' of the source format, not Tinderbox as some assume: there is no paragraph ending information for the application to use.

When pasting in text, it is generally a good thing to use 'Paste and Match Style' ([Cmd]+[Opt]+[Shift]+V]) instead of a normal paste. This will retain bold/italic mark-up and tabs but strip things like font typeface. It should also normalise line endings to the macOS form (more on line endings in text).

Pasting into views

The result depends on the view type of the current view into which the paste occurs:

- **Outline**, **Chart**. By default the new note is the last top-level sibling (i.e. child of root or a hoisted root scope). By hovering the cursor close to a note, the new note can be a previous/next sibling or child of an existing note at any level at root or in a branch.
- **Timeline**. Always the last child of the root of the view. The new note will appear in the 'No Date' sidebar as it will have no $StartDate at this point.
- **Map**. Creates a new note at the last-clicked point. If an existing note is selected, the new note is created to the right of that note in the first available space.
- **Attribute Browser**, **Crosstabs**, **Hyperbolic, Treemap** view. Drop data is not accepted, so nothing happens.

The result of the paste, where a note is created is that:

- the title ($Name) is the text of the note (or the initial part with terminating ellipsis if very long).
- the text ($Text) is set to the plain text of the source data , i.e. all styling, links, etc. are not retained. If such styling is needed, first make a note, set focus in the text are and paste as this will retain (supported) source styling and HTML links.

Pasting notes to a different TBX: $Created and $Modified

From v9.6.0, when a note is pasted into a document, its $Created and $Modified dates retain their values. Previously, both $Created and $Modified were set to the date on which the note was pasted.

Pasting web browser text into $Text

Copy/Pasting web content into $Text is akin to pasting RTF/rich text styled content into $Text. Weblinks and any any supported style features (text colour, font, weight, emphasis) is retained. If SmartLinks are enabled (the default) any weblinks in the source data should be detected and adopted as functional Tinderbox weblinks.

Using Paste-and-Match-Style (⌘+⌥+⇧+V) plain text is inserted. note that any links are lost as only the anchor text is retained. Actual URLs in text will be retained but will not be active links.

Pasting webpage content into a note also sets the receiving note's $URL to the URL of the source web-age *unless* the note already a $URL value. Thus if pasting from multiple sources, only the source URL of the first page is stored, or none the note already has a $URL value for some other reason.

See also: dragging web content into *view* panes.

RTF (rich text) import

Tinderbox does not support direct import of RTF files.

RTF text can be pasted into note $Text, supported style and weblinks are retained. Use Paste-and-Match-Style (⌘+⌥+⇧+V) to paste a plain text version of the source text.

Styled text can be copied from $Text.

There is no bulk import method, i.e. for one or more discrete files. Note that bulk text import is supported for plain text.

Setting up for Tab-Delimited Import

Originally, data table import was only for tab-delimited data (TSV), but both CSV and TSV are supported. The same rules (below) apply to how the data is handled. CSV formatting is only assumed if the file has a '.csv' extension. For '.txt' or '.tsv' files, tab-delimited format is assumed.

For simple data, the auto-generation of attributes from column heads and the data-types assumed are generally correct but for more complexed or nuanced tasks there are some points to bear in mind.

Tinderbox assumes the data will contain a first row of column headers which it can use to map data to existing attributes or generate new ones.

The first column *always* maps to $Name *unless* a column header value of 'Name' is found.

Headers are assumed, so if none are suppled, Tinderbox will assume the first data row is the header row, which may result in some strange attribute titles!

Mapping source column headers to existing Tinderbox attributes

Mapping is automatic, based on the Header (first) row of the data. Tinderbox will match a column header, *case-sensitively*, to any existing system or user attribute. Special cases:

- If a column other than the first one has the header 'Name' (but not 'name'), that column maps to $Name. If no $Name match is found, the first column is used for $Name, regardless of the header value (and that value is not made into an attribute).
- 'Text' but not 'text' maps to $Text. If $Text is supplied from source it is not set as a Displayed Attribute (more below) but a 'text' (i.e. an attribute called 'text') would be.

Thus header 'my var' will create a new (String-type) '$my_var' following the naming rules below.

Auto-generated (user) attribute names

If a column-header (other than if column #1) does not match an existing system or user attribute name, then a new user attribute will be created. The new attribute will use the exact (*case-sensitive*) insofar as the source value matches. 'Illegal characters for attributes are substituted with a underscore per such character. Thus, source heading 'my / stuff' creates $my___stuff (3 underscores for space+slash+space). Note how the case of legal text is maintained and only each illegal character is substituted by an underscore. Underscores, common in some database tables are thus maintained during import.

Date type coercion on import

These appear to be the 'rules' for coercion, that result is one of only 3 data types being created:

- Boolean. '0', '1', 'true' and 'false' result in a Boolean-type attribute. Any value other than '0' or 'false' passed to an existing Boolean attribute will coerce to a Boolean `true` value.
- Number. Any positive or negative integer or decimal number, barring '0' and '1' will create a Number-type attribute, including '0.0' and '1.0'.
- String. All other data maps default to creating String-type data. Lists, i.e. semi-colon-delimited strings, are still coerced to Strings.

Empty Cells

Empty cells (i.e. content-tab-tab-content) and line breaks in cell content are allowed. If having import problems due to empty cells, try adding a dummy last column of source data with a value in each row and then delete that attribute (and data) once the import is complete. Cells may contain line break characters (e.g. text intended for $Text) if it is enclosed by (straight) double-quotes.

Line Breaks

Tab-delimited. Line breaks are not supported in call values (even for $Text).

CSV. Line breaks are supported, as long as (at least) that cell's value is enclosed with double-quotes.

Quote Characters

Both tab-delimited and CSV ignore single straight quotes and single/double typographic ('curly') quotes. Double-straight quotes:

- Tab-delimited: ignores whether pairs or odd numbers.
- CSV. May only be used as balanced pairs to enclose whole cell values. Do not place further odd or paired double-quotes within cell content, even if the cell is quote enclosed.

Importing Lists

As even Tinderbox-formatted lists import by default as *strings*, to avoid getting the wrong data type, a little extra planning is needed.

Source list data has semi-colon delimited values. Add any needed List or set type attributes (correctly type-configured) to the TBX document *before* import. If the data is semi-colon delimited it will be correctly parsed as multiple list items.

*Source list data does **not** have semi-colon delimited values*. For this you need a two-stage process: ingest the list as a string, then correct the delimiters passing the result to a List or Set attribute. First, ensure the header does not use the attribute name actually desired for your List or Set attribute. Make the latter attributes of the necessary type(s). Add a stamp with 2 actions (i.e. with a semi-colon between them. For example, assume the source list uses a '##' string to delimit values in the 'somelist' column that you want to end up in $SomeSet. The stamp might look like this:

```
$SomeSet = $somelist.replace("##",";");
```

Stamp the ingested notes, check the data. If the list data looks correct, you can delete the source attribute (here $somelist). If using auto-generated Displayed Attributes, you may also need to update those to show the correct (list) attributes.

Forcing a non-default data type mapping

As shown above lists cannot be detected and '0' / '1' may be misconstrued as booleans. There are two ways to avoid this:

- Include a dummy first row of data (i.e. after the header row) with values that will force the correct data type. After import delete the note created for the first (dummy) record. However, note this method does not help for data types other than String, Number and Boolean.
- Pre-create the attributes and set the desired type. There are two ways to do this:
  - Import the first 2 rows of source and correct the data types of the auto-generated attributes (and check mapping to system attributes). Any user attributes with the wrong data type can have these reset via the Document Inspector, User tab.Then delete the imported date container and notes before importing the full data set. If using copy-paste from another app rather than from a stored data file, you might need to import all data and delete, then reimport after the attributes have been corrected. This method will also show up any attribute naming issues that may arise and which may need correcting at source.
  - Manually create the new user attributes from scratch in Tinderbox. If there are a lot of such attributes, or the same process needs to be done many times in new documents, the method above may prove quicker.

Import fixes Displayed Attributes in created notes

The import process sets each new per-row note's $DisplayedAttributes. If you are going to apply a prototype to these notes you may first wish to reset the new notes' Displayed Attributes.

Spreadsheet Import: CSV and Tab-Delimited Text

There is no menu option for initiating tabular data import. For tabular data two formats are supported, tab-delimited (TSV) or comma-separated (CSV):

- Tab-delimited: supported for file drag drop and copy/paste of table data. For files use '.txt' or '.tsv' file extensions.
- CSV: supported for file drag drop (CSV copy/paste is *not supported*). For files use the '.csv' file extension.

Tab-delimited tables are imported intelligently when dragged or pasted into Tinderbox, involving some mapping of table data into existing or new attributes. Mapping is based on a header row which is expected as row 1 of the data.

For example, selecting a table in a spreadsheet and pasting it into Tinderbox will create a useful set of notes:

- The first row is treated as a set of headings, which map to attributes. New user attributes will be created for attributes that do not already exist.
- Any column with the heading 'Name' will map to $Name (early documentation requiring 'Name' to be the first data column can be ignored).
- A new container will be created to hold the table's rows. Unlike early versions of this feature source data is never placed in the import container's $Text. If such data is needed for review or re-import, it should be retained as an external file (or link to it via a File-type attribute).
- Each row of the table becomes a note. The table's fields are Displayed Attributes, and these attributes are populated from the table.
- For date-based data: see date formatting for import.
- If there is a 'Prototype' column in the input data and appropriately (case-sensitively) named prototypes pre-exist in the document, the record(s) will have the relevant prototype set for them (i.e. the app sets each new note's $Prototype). Note the necessary prototypes must exist in the receiving Tinderbox document *before* import occurs: Tinderbox will not create named prototypes found only in the data.

More on attribute creation and data types: mapping to existing attributes, detecting data types, line breaks in cell values, empty cells, etc.

The tabular feature works beyond formal spreadsheets, any tab-delimited text sample should import. This allows other material such as vcf address card data to be dragged into Tinderbox. Tinderbox will attempt auto-detection of dates, non-ASCII characters and to handle clipboard data from Numbers '09.

Also, Tinderbox will generally deal with characters in column headers that are not normally supported in attribute names. For instance, an underscore is often used as a substitute character in field names. Whilst a user cannot deliberately make an attribute named "Some_Field" via the User Attribute Inspector, because it contains an underscore, data import may do so in order to retain fidelity with source.

File Extensions. For Tinderbox to correctly handle tabular data added use and appropriate (case-insensitive) file extensions. For tab-delimited data use '.txt' or '.tsv'. For comma-separated data use '.csv'. Only tab-delimited data can be pasted directly from the clipboard and still be parsed into new notes.

Displayed Attributes. The import process sets the $DisplayedAttributes *locally* for all notes it creates. Thus if you apply a prototype using Displayed Attributes to the newly created notes, the Displayed Attributes will not show in the inheriting notes as the local values unless/until you reset the $DisplayedAttributes for these notes.

Prototypes. Data from a 'Prototypes' column will be used to set $Prototype for the newly imported notes, but only if the prototypes are defined in the document *before* the import occurs. Otherwise even if there is data, $Prototype will remain empty.

Existing notes vs. New notes. The tab-delimited import method only creates new notes; there is no automated data merge for existing notes. Thus, tabular import cannot be used to set attribute values in pre-existing notes. This means a new note with a title ($Name) of an existing note will get created on import *as a duplicate*, rather than changing attributes of the existing note. However, this does offer scope for updating existing notes. Action code can search in a given location,

for notes whose $Name matches a note elsewhere. Having identified the link, further action code could then copy across the necessary attribute values to the older pre-existing note. The newly imported imported note could then be moved to a new container outside the scope of the original query (i.e. so the update action only occurs once).

CSV content detection

CSV import of dropped/pasted data into the view pane is abandoned if more than two fields are missing. The challenge for auto-detecting CSV is that if the first paragraph/line of the received test has commas these might either be CSV markers or normal punctuation.

The revised CSV-detection logic is:

- If there is only one record (i.e. input row or line), this **is not** CSV data.
- Does the first record have at least 2 columns? If not, this **is not** CSV data.
  - Note that successive commas, with or without intervening spaces, are detected as empty (valid) columns
  - Is this the first record? If so, 'EXPECTED' is set as the expected number of columns in other rows.
- If this is not the first record, and this line has more columns than EXPECTED, this is **not** CSV data.
- If this is not the first record, and this line has 2 or more fewer columns than EXPECTED, this **is not** CSV data.
- Repeat for each record.
- If no record has failed CSV detection, the data **is** CSV.

If CSV is not detected, normal paste/drop-to-view behaviour applies. The filename (or first sentence, if not a file) is used as the new note $Name and the entire content is placed in $Text.

vcard import

When importing a vcard by dragging from Contacts or elsewhere, Tinderbox places a copy of the vcard text in $Text to allow you to extract specialised fields unique to your work. In some cases, however, the vcard includes a base-64-encoded image, which is not useful in the text. That data is removed from the imported text.

Watched folders

A set of options in the File ▸ Watch allows the creation of various types of watched folders linked to external applications. The result is a form of AutoFetch operation.

Multiple watched folders of the same source type (e.g. more than one Notes folder) are allowed though be mindful heavy use of this feature may affect overall performance, i.e. he feature is not designed as a fully-synced system but rather for importing/updating *some* external data.

A watched folder *is always added at root level in a document* (and should not be moved elsewhere). This is for necessary design considerations re performance. Likewise the watched destination is envisaged as a single item or small collectio of items and not a large hierarchy of folders/files.

The prototypes of watched notes specify that watched notes are imported with a $TextBackgroundColor of white, even if dark styles, since external documents are most likely compatible with light background colours.

Notes imported from watch folders respect ziplinks-style '[[' markup. To disable ziplinks in a watched folder note, set $Ziplinks to false. To disable ziplinks *throughout* a watch folder, set $Ziplinks for the folder to `false`.

Legacy support note: watched folder support for Evernote was discontinued in v9.0+ due to internal changes in Evernote.

Refresh timing: all watched folders are refreshed when the TBX document is opened. Then, while the document remains open, the folders are refreshed periodically on their own cycle. The frequency of that cycle is not documented and no force refresh mechanism is known.

From v9.5.0:

- When watching a folder in Finder, notes that correspond to deleted (or moved) files are now shown with their names struck through by automatically setting $NameStrike to `true`.
- Watched folders of all kinds are checked more frequently.
- When a file in a watched folder is checked, its $LastFetched is now updated even if the note is unchanged. The note's $NotesModified attribute continues to reflect the modification date of the file.

There are currently 3 forms of watched folder:

- Finder
- Notes
- Tot

Finder

Tinderbox can automatically import selected notes from the any Finder folder, including folders in Dropbox or other remote servers. This is set up via the File menu.

Multiple watched folders are supported, i.e. more than one (root level) watched Finder folder.

To connect Tinderbox to a folder, *create a root level container* (and it can *only* be at root level) to hold the imported notes and set its $WatchFolder attribute to the name of the Finder folder you want to import. The watched file may be only local to the host Mac drives or may be a shared file via iCloud or Dropbox. Tinderbox will automatically import to or update that container (for supported format files) whenever the TBX document is reopened, and periodically thereafter as part of the AutoFetch cycle.

The new container created in Tinderbox uses the same name as the source OS folder in Finder and as, as a Displayed Attribute, $WatchFolder. The latter\s value is the POSix local path to the watched folder. Reseting the $WatchFolder to the default (i.e. no value) will cause the Watch function to cease operating.

The imported notes inherit from a built-in prototype named ' Imported From Finder', making it easy to common Displayed Attributes or visual appearance. Finder tags are imported into $Tags.

New items

On detecting a new source file, Tinderbox will add a note with the complete source filename (including extension) with the above prototype auto-assigned. The new notes Displayed Attributes table shows (via the prototype):

- File holds the OS path of the source file.
- $NotesModified shows when the item was *first* added.
- $LastFetched shows when the source note was last detected as changed (as opposed to when the resource was last checked).
- $ReadOnly, though this can be toggled off by the user.
- $Tags, any imported Finder tags.

If the user chooses, additional Displayed Attributes can b e added via the prototype (or individual note's $DisplayedAttributes) though such displayed attributes cannot be editing in the Displayed Attributes table whilst $ReadOnly is ticked (`true`).

Changed Items

When, during autofetch, a source note is detected as changed (i.e. as a different macOS modification date/time) the affected note is updated:

- the $Text value is replaced by the current source text.
- the $Tags value is replaced by the current source finder tags.
- $LastFetched is updated to the time of this (Tinderbox) edit (not the source file's OS modified date/time).
- if a source file is now missing (e.g. deleted or moved) Tinderbox retains the note, but the user may delete it without the note being later regenerated.

Note that the 'update' overwrites existing $Text and $Tags. In the case of $Tags the inputs are not merged so any tags set in Tinderbox would be lost. Apart from the 3 attributes above, no other change occurs so things like user attributes set for the note will not be affected.

Limitations:

- Changes to the imported notes will **not** be propagated back to the source application (i.e. Finder) or to other devices: the source data is *watched*, **not** two-way synchronised.
- Notes created by watching Finder are read-only by default, inheriting $ReadOnly from their prototype.
- Changing Finder tags, e.g. via File Info, does not necessarily trigger a change to the file's OS modified date/time so Tinderbox does not detect a change. Therefore it may be necessary to alter the files OS modification date/time by othe means if it is desired that Tinderbox detects a change only to tag data.
- there is no way to know when the AutoFetch last ran, though the Tinderbox Inspector's Agents & Rules tab gives some idea (the circular progress meter). As no change happens if no change is detected as source, the only way to be sure the watch has run, e.g. if an expected source change does not appear in Tinderbox, is to close both the TBX *and* the Tinderbox app, then re-open both. Note the point above that simple changes to (Finder) tags in Finder File Info does not mark the file as changed.

Factors to consider when using watched Finder folders

If intending to use menu File ▸ Watch ▸ Folder from Finder, be aware the target folder may contain a mix of documents some of which Tinderbox cannot read/view (i.e. make meaningful use of).

If this is the case consider separating non-compatible

Notes

Tinderbox can automatically import notes from a folder the Notes application, which is installed in all macOS and iOS devices. This is set up via the File menu.

Multiple watched folders are supported, i.e. more than one (root level) watched Notes folder. When watching a new folder from Notes, the menu of folders is sorted alphabetically.

To connect Tinderbox to a Notes folder, *create a root level container* (and it can *only* be at root level) to hold the imported notes and set its $NotesFolder attribute to the name of the Notes folder to import. The Notes application's main 'Note folder cannot be used as an import destination. Tinderbox will automatically import or update that container whenever the TBX document is reopened, and periodically thereafter as part of the AutoFetch cycle.

The imported notes inherit from a built-in prototype named ' Imported From Notes', making it easy to set common Displayed Attributes or visual appearance.

Limitations:

- Changes to the imported notes will **not** be propagated back to the source application (i.e. Notes) or to other devices: the source data is *watched*, **not** two-way synchronised.
- The import process is comparatively time-consuming; it may be preferable to limit import to folders with no more than a few dozen notes.
- Notes created by watching Notes are read-only by default, inheriting $ReadOnly from their prototype.

Tot

Tinderbox can watch and automatically import notes from the Tot application. This is set up via the File menu.

Watching Tot creates a top-level container 'Tot' with 7 numbered child notes, 1–7, one for each Tot note. Watching Tot will import text from Tot, but does not sync *to* Tot. Thus, watched Tot notes are set as read-only as each refresh replace existing $Text with the source Tot note's text.

The watch process also adds the ' Imported From Tot' built-in prototype, to the notes. The prototype sets $Tot and $ReadOnly as Displayed Attributes.

The 'Tot' container sets the $Tot attribute to "1;2;3;4;5;6;7". Removing any list item stops that Tot note being monitored, but the relevant Tinderbox note is not deleted. If a note for an unwatched Tot note is deleted, it is not recreated.

Limitations:

- Changes to the imported notes will **not** be propagated back to the source application (i.e. Tot) or to other devices: the source data is *watched*, **not** two-way synchronised.
- Notes created by watching Tot are read-only by default, inheriting $ReadOnly from their prototype.

Watched Groups

Content can be auto-fetched for groups of records from some other applications. This is **not** a sync mechanism.

Multiple watched groups are allowed though be mindful heavy use of this feature may affect overall performance, i.e. the feature is not designed as a fully-synced system but rather for importing/updating *some* DEVONthink data.

A receiving container for the group's data *is always added at root level in a document* (and should not be moved elsewhere). This is for necessary design considerations re performance.

Refresh timing: all watched folders are refreshed when the TBX document is opened. Then, while the document remains open, the folders are refreshed periodically on their own cycle. The frequency of that cycle is not documented and no force refresh mechanism is known.

There is currently one form of watched group:

- DEVONthink

DEVONthink

Tinderbox can automatically import selected groups of records from DEVONthink Pro (and DEVONthink Pro Office).

To connect Tinderbox to a DEVONthink group, create a *root level container* (and it can *only* be at root level) to hold the imported notes. Set the container's $DEVONthinkGroup attribute to the unique DEVONthink ID of the group to be import (this is the group's Item ID).

Multiple groups are supported, i.e. more than one (root level) watched DEVONThink group, but be mindful that excessive such mapping may affect performance.

The Watch set-up uses the DEVONthink group picker.

NOTE: *the watched group(s) feature is **not** a sync mechanism*, but a method to pull current DEVONthink data into Tinderbox

The DEVONthink reference must be a group and not an individual item (if necessary, place a single item in its own group). Tinderbox will automatically import or update that container whenever the TBX document is reopened, and periodically thereafter as part of the AutoFetch cycle.

The resulting imported notes inherit from a built-in prototype named ' Imported From DEVONthink', making it easy to set common Displayed Attributes or visual appearance.

Limitations:

- Groups nested within the referenced group are not currently imported.
- Changes to the imported notes will not (yet) be propagated back to the to the DEVONthink application or to other devices.
- Notes created by watching DEVONthink are **read-only** by default, inheriting $ReadOnly state from their 'Imported From DEVONthink' built-in prototype.

PDF documents dragged from DEVONthink are imported as styled text if possible. Previously, they were imported as images.

Ziplinks are interpreted when text is pasted in the view pane, or when files or DEVONthink items are imported by dragging them into a view. Ziplinks are not interpreted when $Ziplinks of the imported note is `false`. (Note that the built-in Imported from DEVONthink prototype in previous releases defaults to setting $Ziplinks false; if wanting want to import ziplinks from DEVONthink into older documents, check $Ziplinks is set to true. DEVONthink watch folders should interpret `[[ziplinks]]` found in source (i.e. DEVONthink) text files.

Factors to consider when using watched DEVONthink groups

If intending to use menu File ▸ Watch ▸ Group from DEVONthink, be aware the target DEVONthink group may contain a mix of documents some of which Tinderbox cannot read/view (i.e. make meaningful use of).

However, DEVONthink users should recall that DEVONthink supports 'replicants' (similar, but not exactly the same as, Tinderbox aliases). If a new group is created in the source DEVONthink database, then Tinderbox-viewable document formats such as Markdown (.md) or Rich Text (.rtf) can be replicated into it; this new group can then be the group designated for use with Tinderbox's watch group feature. Tinderbox can then be configured to “watch” that DEVONthink group holding the replicants.

This replicate-to-target group approach can also be used to gather documents from across a DEVONthink database to a single group that Tinderbox watches.

Working with $Text: support for styled text

Styled text, whether just basic bolding/italicising/underlining or more complex RTF formatting is supported only when working via a note's text window. Thus:

- Full RTF styling:
  - Typing into the text window & use of app menus on $Text selections.
  - Copy/paste from RTF source in other notes or outside Tinderbox.
- Un-styled text (no mark up at all):
  - Inspector setting $Text
  - Info view setting $Text
  - Action code setting $Text. This has been changed by some new action code allowing the setting of $Text size and colour.
  - Drag-drop text file import (only supports plain/UTF-8 text)
  - Menu: Edit ▸ Paste and Match Style (only characters, whitespace and line breaks are preserved)

Tip: If pasting styled text, it is often much faster to use Paste and Match Style and then re-apply any needed styles, rather then trying to wrangle RTF paragraphs styles. Rich pastes often bring their own page width, i.e. different from Tinderbo defaults, with fixed widths/margins which can be problematic for hassle free printing. Remember that by default text pane rules are hidden so these rogue paragraph settings may be hard to spot. There is no paste option that preserved some formatting, e.g. bolding, but drops margin and tab settings.

Export

This section covers various forms of export of notes from Tinderbox, both via formatted (e.g. HTML) marked-up text and also using TRF or plain text.

Also see the Export Inspector, export codes, and export-related system attributes.

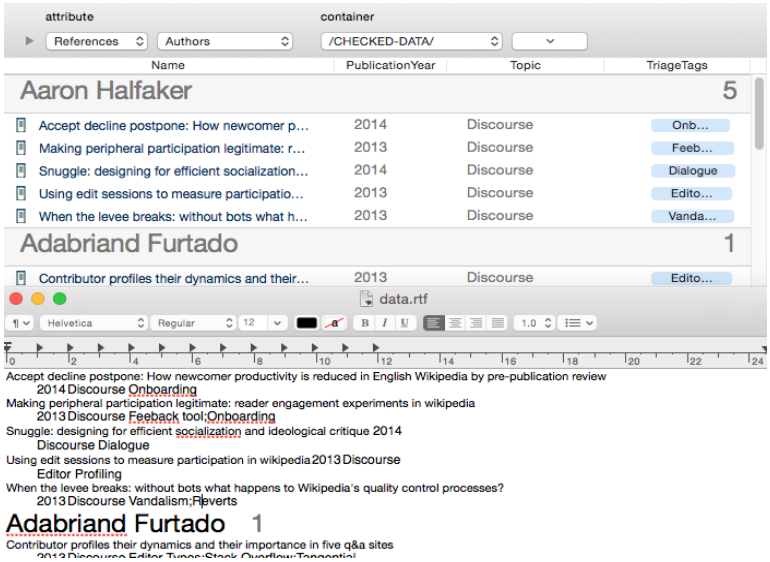
More above export from Tinderbox.

Aliases in HTML Export

When using the HTML Export method, whether for HTML or other formats, aliases are exported as separate pages in the appropriate location within the output. This makes it easier to use web links to alias content that point to the right place. It also helps when web output uses a hierarchical navigation system as with aTbRef.

Aliases share their original's text and web links but can have their own basic links (if none the original's are used on export).

Attribute Browser Export



When the current tab is an Attribute Browser and focus is in the view pane it is possible to Export the view to an RTF document.

Per-attribute value headings (and their summaries if present) are exported as a row.

Each note listed is exported as a row of text, using $DisplayName. If column view is used, column data is included tab-delimited.

Although all discrete entries are tab-delimited, the RTF itself is not in tabular form.

Built-in Export Templates

Tinderbox includes a number of specimen template notes to export using HTML, or other formats such as OPML. These are accessed from the File menu. Any necessary (root level) containers and/or prototypes these notes need are added along with the template. See more on templates.

The built-in template listing also includes templates from the application support ' templates' folder.

Templates that are also prototypes are not shown in the various pop-up template menus (such as in HTML view).

Copying from Column view

When copying selected item(s) from an Outline view that has column view enabled, Tinderbox also copies the displayed columns' data is tab-delimited form. This allows simple sharing by pasting data into spreadsheets and other programs that use tab-delimited data.

Default Export Location

If not already given a location for export, Tinderbox defaults to using the current user's Desktop folder.

The export location is set via the Export Inspector's Export tab.

Default export template

If no export template is defined for a note, a default of ^text^, is used instead. This is not a formal template, but a fall-back to ensure the user sees something even if they have not tried export or do not wish to export data.

Dragging from Tinderbox

A "text receiver" is a program that expects to receive text drags, such as a word processor or a Tinderbox text window.

When dragging *multiple notes* to a text receiver, Tinderbox supplies a list of all the note titles.

When dragging a *single note*, Tinderbox supplies the styled text of the note if the note's text is not empty. If the note's text is empty, Tinderbox supplies the note title.

Email from Tinderbox

The runCommand() action syntax operator allows Tinderbox to access the command line and send email. In most versions of macOS, the user will have to make some adjustments to their system to be able to send email via the command line
As the adjustments vary from OS version to version, the detailed process should be researched in more appropriate fora.

Export code - default formatting for attribute data types

When exporting attribute values using ^value()^, intuition may not match output. A String is likely unchanged, but what about a Boolean, or a Date? This note aims to clarify those possible discrepancies.

Single-value attributes

- String (and string-based: Action, Color, File, Font, URL). There are no effects here. White space in the source value is retained in the output value.
- Number. These are exported as seen. This includes numbers that Tinderbox has stored in exponential notation (see here). The Number.format() operator can be useful in setting maximum decimal places or padding characters so all items values export with the same number of characters.
- Boolean. These export in the form `true` or `false`.
- Date. Dates export in ISO 8601 format (see here), so it is likely a Date .format() transform will be desired.
- Interval. Intervals are exported as a string, the same as seen when editing an interval's value.

Multi-value attributes

- Dictionary, List and Set. These are exported as the literal strings storing the attribute value. Any white space in and between terms is preserved. It is most likely a list .format() transform will need to be applied.

Export - managing source code white space

When ^export^ code is evaluated, it inserts the necessary code into the output—or none—as pertinent. In the case of ^if()^ clauses this can lead to blank lines in the output source code as although the unmatched branch emits nothing, the line return after ^endIf^ is still emitted.

The trick is to place the line return inside the if clause. This is achieved by placing the ^endIf^ on the next line. Originally you might have had this:

```
^value($Name)^
^if(ChildCount)^
There are ^value($ChildCount)^ children
^endIf^
^text^
```

Which would either emit a blank source line before/after the number of children, or a blank lines if there are no children. To resolve this, use:

```
^value($Name)^
^if(ChildCount)^There are ^value($ChildCount)^ children
^endIf^^text^
```

If using an ^else^ branch apply the same principle, i.e. ensure all line breaks belonging to the result of if clause are inside the if statement:

```
^value($Name)^
^if(ChildCount)^There are ^value($ChildCount)^ children
^else^There are no children
^endIf^^text^
```

If source code is indented with tabs or spaces, do not forget the ^endIf^, *precedes* such indentation rather than coming before the first on-screen output.

Export - name collision for created files

When exporting, if two files exported to the same folder location would have the same name, Tinderbox creates a synthetic name to avoid the collision.

For example, if two notes in the same container are both named "1", the first will be exported as "1.html" and the second as "1_1.html". A third note will be exported as "1_2.html".

For this reason, Tinderbox will not return a value if testing $HTMLExportFileName, except if it has a value set explicitly. Even in the case of the latter, the stored name value will be modified according to the above rules should a naming collision occur.

However, a filename can be derived via $HTMLExportPath, which holds the calculated export folder/filename path for the note (read-only). This uses the as-exported folder and file names from export root that would be created if the file were exported at that time. It thus show the names of the folders & file with any characters omitted or substituted as part of the export process.

This can be extremely useful in more complex export scenarios where there is a need to reference in code an exported note's OS filename or path without being forced to hard-set $HTMLExportFileName. If not set, the latter cannot be queried within Tinderbox.

To get just the exported filename from the path:

```
$MyString = $HTMLExportPath.split("/").at(-1)
```

For this note:

$Path is: "/A Tinderbox Reference File/Export/Export - name collision for created files"

$HTMLExportPath is: "/index/Export/Export_-_name_collision_for_created_files.html"

Note the spaces being lost in the latter. The export version will also show $HTMLExportFileNameSpacer changes, if the latter is set.

Export - RSS and Atom feeds

Tinderbox's HTML Export can export all sorts of formats, including Atom and RSS feeds. Indeed, this TBX exports valid Atom and RSS 2.0 feeds. Ideally you should read the format docs for your desired formats as you may wish/need to tweak the templates used by aTbRef to support better you own data needs. The primary references are:

- Atom format: http://www.atomenabled.org/developers/protocol/atom-protocol-spec.php
- RSS format: https://cyber.harvard.edu/rss/rss.html
- Feed validator (for both formats): https://validator.w3.org/feed/

The format specs can be intimidating for the ordinary person, but hopefully the example templates used by aTbRef should meet most everyday needs.

In each case the process uses an agent to gather the notes to be referenced in the feed and then exports using an envelope (agent) & letter (agent's child aliases) technique.

Notes for re-using aTbRefs' method & templates:

- The feed-generating agent goes at the end of the outline. This is deliberate, so as to ensure all other agents have updated in the last cycle so the query results are as up-to-date as possible. This becomes more important in a document like aTbRefs where there may be agents querying other agents, etc.
- aTbRef's feed simply looks for `descendedFrom("A Tinderbox Reference File")&($Modified=="today")`.
  - The first query term ensures non-content notes are ignored. This is also why all the content is descended from a single Root-level note, as it makes it easier to keep back-of-house stuff separate in TBXs where you intend to export the data.
  - The second term looks for items changed today. As creating a note sets both $Created and $Modified, the query finds both new and updated notes. Be aware that $Modified only reflects changes to $Text (possibly $Name also). If this is insufficient for your needs, alter the existing query as required.
- The feed agent should be set to use the 'news_atom' or 'news-rss' template according to the type of feed being generated. You will find the templates needed in the `/Templates` container of the aTbRef TBX document. Each of these templates is set to call a child template for per-item content (newsitem_atom, newsitem_rss).
- All text in orange in the aTbRef versions of the template will need to be reviewed and updated as necessary for use elsewhere, e.g. the feed author/title/source URL/etc.

Export - splitting content and utility notes

As can be seen by looking at the source TBX of aTbRef, it can be useful to place all 'content' i.e. the data you are working on and wish to export, inside one container. This also makes it easy for that container to form the index page of an HTML export or root element for XML export, etc. Thus "A Tinderbox Reference File" is the index.html file for this website. All notes used in the visible web content are descended from this note.

So what else might there be outside that? Looking at aTbRef's outline root you will find:

- /UTILS/. A variety of utility agents and the documents prototypes. Tip, set the prototypes' container's $OnAdd to `$IsPrototype=true;`.
- /Boilerplate/. This holds some code-type notes with sections of code to be included in export templates. This makes the individual templates easier to maintain as boilerplate can be updated once for several different templates. Tinderbox's built-in 'Code' prototype is often used for this type of note as usually it is important that the code's text is not modified by the export process, e.g. inserting HTML paragraph tags, etc., using the prototype saves the user setting lots of attributes to non-default settings in 'code' notes.
- /Templates. All the export templates needed are stored here: HTML, RSS, etc. Tip, set the container's $OnAdd to `$IsTemplate=true;`; in aTbRef's case it is `$Prototype="_template"`, as that sets a prototype for which $IsTemplate is already correctly set.
- /CSS/. All CSS used by the website is exported from the TBX. If needed, the same method can be used for Javascript and the like. Look at HTML view for the CSS container. Note how it does not export but exports its children; also it uses $HTMLExportFileName to change the case (it could be the name) of the exported folder name. View the child's HTML view to see how the CSS filename is constructed. Having this data in a root container means it exports to a sub folder of the exported web-site. Were the CSS notes nested more deeply in the outline, it would create a nest of folders. Thus having a root level container makes sense, especially if a root level content container is also used.
- an HTML site map. An HTML nested listing of the entire content outline. Clearly, placing this inside the content section would not be a good fit, and opens issues of unintended recursion, so it lives outside the content.
- Version checkers. These export small code pages used to allow users who prefer to consume aTbRef in TBX form to check their local document is up to date. See more here and here.
- Atom and RSS feeds. See more.
- Zippers. These use command line code to create separate Zip files of the TBX and the /images/ folder (all images have always lived outside the TBX). The latter is needed if running out an HTML site locally and without web access. See more here and here.

Export and links to page includes

Note that basic and text links to other notes are suppressed on export if the destination note is not exported as a stand-alone page, e.g. the destination is only exported as part of the inline body text of another note.

It might be assumed the 'parent' page of the included content would inherit the inbound link but it does not. The latter might seem a bit of a blunt cut-off of functionality but complex pages using a lot of includes (such as some blogs) make upward inheritance of link destinations less simple than it might appear.

So, if your document is to be exported to a format like HTML and has rich cross linking, this issue re includes can be an important fact to consider when designing the templates and how/where content is stored. To preserve cross-linking you will generally need to use 1 note per exported page and ensure you give good navigation links to the site.

Export selected note or notes

From v9.5.0, the File menu option to export a selected note now allows a multiple selection. This has an implication for how the export location is selected by the user. In addition to the changes that implies, overall there are a number of improvements to this feature:

- **Single or multiple note selection**. Instead of only exporting a single note, this method can now export each note from a selection of multiple notes. Otherwise behaviour is as for exporting a single selected note, as if this were a succession of individual exports.
- **Embedded images**. If a selected note has embedded images these are exported alongside the (HTML) file generated. For each note in the selection the images are exported alongside its source note (i.e. paired images and file, per exported note in the selection).
- **Creation of Destination folder(s)**. If the destination directory does not yet exist inside the export folder, it will be created. For example, if the exported note is at path `/People/Lincoln`, a folder "People" will be created inside the export folder if that folder does not already exist, and Lincoln.html will then be created inside this folder.

To where do notes/images export?

The export folder for the document is set via the Export Inspector. However, the need to support both internal-only presentation preview and pre-export-preview has introduced some complication, especially for existing Tinderbox users, as is explained further below. The export folder may be set internally (for in-app presentation) or externally (for actual export). The default for new v9.5.0+ documents is an internal location. Pre-existing documents with no export folder explicitly set used to default to `~/Desktop` will now default to "none", i.e. the new internal location (see next section).

Therefore the user needs to be much more careful to check the Export Folder location **before** export.

What happens if the OS Export Folder is not defined?

By default, from v9.5.0, a new document defaults to using an OS system internal temporary folder for export, so as to allow zero-configuration preview. If single note export is called in this context, the OS chooser dialog will default to the users Documents folder (`~/Documents`). Note that once a full export has been used in a document, it is likely to have the Export Folder set to an external location.

Changes from previous single-note export behaviour

In older versions, when exporting a note from a document with a previous export folder (and having exported), the export OS chooser dialog opened at the location of the previous file. If the note as new, the OS choose opened to the folder where the note would previously have exported. In other words, on export the user was shown the folder into which the note had/would be exported.

Now, in v9.5.0+, the fact the export can be one or more notes requires a slightly different approach, albeit to the same end. An effect of multiple note selection is that the selected notes may not all be in the same container, yet we still wish them to go where they should go in the overall exported website. Thus the only common folder to which to point Tinderbox's export is the root Export folder, such as is used for full document export. From that Tinderbox works out (and creates if necessary) the folders to the notes actual location … for each note in the selection.

So the outcome is the same as when only a single selected notes was exported, but done a different way.

What does this mean for long term users of this feature:

- **Exporting new pages into an existing set of exported pages** . Select the TBX's export folder and **not** the folder where the actual note/file lives. Doing the latter will create the whole outline-derived OS path all over again. At the beginning of a Tinderbox session the default OS folder offered is the main Export Folder and *for the new behaviour this is the correct folder to use* .
- **Exporting ad hoc items**, e.g. data tables, compiled reports. Select the desired OS folder. Again, the session default is the full export folder.
  - Because the export now always writes the in-doc path, it is effectively impossible to export a selected note directly to the folder specified unless the note being exported is at document root (otherwise the outline path is inserted to the OS folder).
- If using **both the above methods** in one session, when using the re-export of existing files (case #1 above) ensure the document's Export Folder is set to the OS external root folder for whole-document export.

Export: 'envelope and letter' technique

For export of multi-level listings such as section indexes, OPML or full HTML site maps, it is usual to use a pair of templates. The 'Envelope' is the actual full page, in the appropriate export format. This template includes in its body an include call to a second, recursing 'letter' template. A recursing template is one which calls itself. In effect is say "do something and if there are any children include each of them using this template. Thus the children of the original include call their children and so on down to the lowest outline level descendants of every branch of the starting container note. The 'letter' template should be designed so that each included note's output data is a legitimate piece of code within the overall exported page, e.g. in HTML <li> list element or an OPML <outline> item element

This technique is the mechanism used to provide aTbRef's site map of every page in the exported HTML site. The 'envelope' is created and a 'letter' of however many sheets as needed is placed within. For those with aTbRef's source TBX, the source code for the envelope and letter templates may be viewed.

Another issue with recursing template export is that a given note can only export itself once, other than as an include. However, it may be desirable to export date in more than one configuration or format. A note's alias shares the same template attributes, although the alias is exported as a discrete copy (into its outline location). Thus an alias, for reasons cited, cannot be used to help in this context. The solution is to anchor the export from another note.

Again as an example, consider aTbRef's site map. The root note of this TBX is already exported as an HTML page but it is also needed as the seed of the site map page. So, instead, another note is used to export the 'envelope' template. In the case of the site map, the template is used only the one across the site, so in this case the include for the root note is in the envelope template. If the envelope/letter templates were needed for several different exports from the same TBX, the envelope template's include code could be replaced by ^text^ and the include code be moved into the $Text of the note exporting the 'envelope'.

Although perhaps intimidating at first side to those without any coding experience, the principle is very simple once understood and solves a lot of export-related problems.

Exporting code samples

Normally, Tinderbox lets you embed html in your notes, and exports that embedded HTML without change.

Spans of $Text using the note's 'Code' font (as in $CodeFont) are now exported with <code> tags. The markup applied is determined by $HTMLCodeStart and $HTMLCodeEnd. The Style tab of the Export Inspector lets you edit these.

Inline code examples

Sometimes, though, you want to show an example of your HTML (or some other programming language) on a web page. That means you want the HTML tags to appear as you see them in Tinderbox, i.e. you see the character sequence "", not the effect of bolding some text.

Originally, setting $HTMLQuoteHTML to `false` let you export HTML examples in an entire note. But, sometimes, you want the example to be handled differently than the rest of the note so $HTMLQuoteHTML is deprecated in favour of the following usages:

- Tinderbox respects the tags <code> and </code> when they appear within notes. HTML between these tags is always encoded during export so it appears as you see it, instead of being passed unchanged to the browser with the exception that within <code> sections, the opening angle brackets of all tags *must* be encoded (see section below) in the source text.
- Tinderbox also respects the tags <pre> and </pre> when they appear within notes. Inside these tags, Tinderbox adds no paragraph formatting and leaves whitespace unchanged; since the intent of the <pre> tag is to allow manual line breaks, adding paragraph markup in this context defeats its purpose. Generally, text inside <pre> is also further enclosed in <code> tags. The <pre> controlling the line breaks and layout, the <code> the use of a monospace font.
  - If combining use of <code> and <pre>, the <pre> tags should go outside the <code> tags, enclosing them. Either way, enclosed code needs all opening angle brackets encoded (see below).

Setting $HTMLQuoteHTML to `true` will override this feature, so do not mix the two!

Opening angle brackets of tags with in <code> and/or <pre> sections

Although a single left angle bracket < can be typed literally in text, e.g. used as a less than sign, in all other cases it must be entered into the $Text as an encoded entity '`&lt;`'. So, a < followed immediately by anything other than a space or number character must be entered as '`&lt;`' or it may be exported incorrectly. For this reason, nearly all of the many < in the source $Text of this article required the character to be HTML-encoded in the $Text so as to render correctly in the exported text.

Therefore, '`&lt;`' encoding of a tag opening **must** be used even inside sections of text enclosed by <code> and/or <pre> tags. Why should this matter inside code samples? Consider these two examples where the sample includes HTML styling tags and :

- `This is some text.` → here the < of the bold tags were not encoded and bolding is applied
- `This is <b>some</b> text.` → here the < of the bold tags were encoded as '`&lt;`' and so the code is seen as intended

Using the newer $Text 'Code' font option does not remove the need to check < characters inside the code-styled text.

This requirement to encode < characters opening tags holds not just for HTML code samples but any code where < and > are use to indicate inline tags, eg., XML, OPML, etc.

Code samples and paragraph spacing

Note that if you use <code> for a single line sample, you will not emit a wrapping <p> tag as Tinderbox does not emit an 'auto-paragraph' if:

- The line beings with a < character
- The line beings with a < character preceded by whitespace (space, tab, etc.)

So, if you want such a single line code example emitted as a separate paragraph, place an option+space at the end of the line after the </code> tag. Tinderbox will detect this and add the <p> tags Also, if such a line also starts with a tab it will use your TBX's defined indented-paragraph styling.

Whole note code samples

Alternatively, you may have a note consisting entirely of code (HTML, XML, etc.) that needs to be exported verbatim, e.g. as is a boilerplate include to a larger exported page. In this case the best approach is to turn off $HTMLMarkupText (i.e. set to `false`). This should ensure the pages characters are exported untouched. However, characters outside the low ASCII set, such as a • may get transposed to a Unicode-style HTML entity.

Notes using lists

If you want to use $HTMLMarkupText but have paragraphs starting with *, # or • you can just set $AutomaticIndent to `false`. This essentially suppresses Quick List functionality.

Exporting files and OS case-sensitivity

Tinderbox happily supports notes with titles varying in case, e.g. "Some Note" vs. "Some note". It also supports case sensitive export in file naming, e.g. "Some Note.html" vs. "Some note.html".

But, by default, macOS is case-insensitive (i.e. host users' host OS). This creates a problem if Tinderbox exports "Some Note.html" vs. "Some note.html" to the same OS folder. What happens is the filename of second to be exported is coerced to the filename of the case-insensitively matched file and the first-exported file is silently over-written.

Workarounds (if there is a need to export notes) are:

- avoid same-name/different case notes in the same Tinderbox container
- set specific, non-clashing export filenames for one of both/all such notes, e.g. "Some Note1.html" vs. "Some Note2.html".
- use Mac with the OS set to be case-sensitive. This drastic and not suggested, as it involves re-installing the OS and is thus not a trivial task.

Export Folder OS Location

From v9.5.0, the default folder used for exporting files (primarily 'HTML' export) changes:

- New v9.5.0+ documents open with the default export folder set to a folder in OS temporary folders. Preview will use this folder as a temporary export folder, and cleans up when the application is closed.
- Older documents will continue to use the previous default of `~/Desktop`, unless a user-chosen location has been set.

The export folder may be changed by either:

- using the Export inspector's Export Folder button
  - a button in the Export Inspector allows selecting to the temporary folder location (as the OS normally hides this from users).
- choosing File ▸ Export ▸ HTML and selecting an export folder

If the current choice of export folder is the temporary folder, choosing menu File ▸ Export HTML will invite the user to select a new export folder, rather than defaulting to the (hidden and thus confusing) temporary folder. Exporting to the temporary folder is not a useful thing to do, but is a boon when using internal preview/presentation.

A hidden factor here is changes by Apple in internal security required for previewing Web(-type) content. As a result, notes need to be exported before they may be previewed. For a user doing internal preview, exporting files would require clear-up afterwards. By exporting to hidden temporary files, the zero-configuration of internal preview is maintained. Conversely normal (file export is able to work as before.

Exporting Folders

In some circumstances it is desirable to get Tinderbox to emit a folder but no note when exporting a container; for instance aTbRef's CSS file is in a folder /css/ but there is no 'css.html' note.

The method to doing this is simple and relates to use of $HTMLDontExport and $HTMLExportChildren, the two tick boxes top left of the HTML views' Export pane.

To export a folder, un-tick the top box ($HTMLDontExport) and leave the second box ticked ($HTMLExportChildren). As long as there is at least one descendant note that does export, a folder will be created. Indeed, all folders in the path will be created if the only exported note is several outline levels down.

Folder name. The folder created uses the same name as would be used for the note's exported file ($HTMLExportFileName), without the extension ($HTMLExportExtension).

In scenarios needing specific folders for content that names can often be ones that are cryptic is used as the container note's $Name. This too need no be a problem. Simply set $HTMLExportFileName, or type in a name into the 'Filename' box in HTML view. The latter will be the exported name, e.g. "refs" whereas the $Name seen on Tinderbox might be "'refs' folder". If doing this in a context where the TBX may be shared with others, consider using a $DisplayExpression to ad a warning about not editing. For instance this code:

```
$Name+" (DO NOT EDIT/MOVE!)"
```

Thus the user might see a screen caption in Tinderbox like so:

'refs' folder (DO NOT EDIT/MOVE!)

Meanwhile the actual exported folder would simply be "refs".

When using 'folder export' *do not* delete $HTMLExportExtension. During export this is attribute is not used. However, if deleted (i.e. it has no value) then the child/descendant data is exported as siblings of the exported folder.

Exporting images

HTML Export exports images embedded in notes. Note that the images are exported into the *same directory* as the note's HTML file. Prior to v6, notes were exported to a subdirectory of the note's folder. The exported images are JPG, and take the host page's name. If note 'About' has 2 images, export with create 'About.html' alongside 'About.jpg' and 'About1.jpg'. Note the image numbering is essentially zero-based but the zero is not applied, i.e. if a note only has one image, as it most likely, the image will have/need to number.

The number of images embedded in a note is stored in $ImageCount. Any note with an $ImageCount of more than zero will have images embedded in its $Text.

If previewing HTML pages and the document has not been exported, i.e. no export folder is set, and images needed for preview purposes will be generated and placed in the same folder as the TBX. Therefore It is a good idea to set an export folder, at minimum before using HTML preview if using inline images.

Text images are not exported via text export.

Picture adornments are not exportable (but are included if copying an image of the whole map).

Exporting Set-type data

Set-type attributes can hold multi-value data, e.g. a set of tags. Some action operators can create sets, most notably the links() operator. Having created a set it can be useful to make it into an HTML list. This is easily done using the forma operator. For instance, if $MySet has values 'cow', 'dog' and 'eel':

```
format($MySet"<ul>\n","\t<li>","</li>\n","</ul>\n");
```

This gives:

```
<ul>
 <li>cow</li>
 <li>dog</li>
 <li>eel</li>
</ul>
```

It is more convenient is to use the newer .format() command:

```
$MySet.format("<ul>\n","\t<li>","</li>\n","</ul>\n");
```

If the above were a list of notes' $Name values it is also possible to make the list items into links. This is possible using a two-step process. First the existing set's data is copied to a new set with each value enclosed by link export code:

```
$MySetB = $MySet.format("","^linkTo(",")^;","");
```

The format() operator is then called a second time to make an HTML list using a String attribute $FormattedList to hold the output:

```
$FormattedList = $MySetB.format("<ul>\n","\t<li>","</li>\n","</ul>\n");
```

On export, the ^linkTo()^ codes are expanded:

```
<ul>
 <li><a href="Data/cow.html">cow</a></li>
 <li><a href="Data/dog.html">dog</a></li>
 <li><a href="Data/eel.html">eel</a></li>
</ul>
```

The $FormattedList attribute is simply a placeholder to store the fully formatted list ready for use in an export template, thus:

```
^value($FormattedList)^.
```

The overall process can be done without needing a second set attribute if link() can be used to make the source set of values. Note the need to use the eval() operator to capture the link() output, on the fly, in set data form to pass into format():

```
$MySet = eval(links(Data).outbound.example.$Name).format("","^linkTo(",")^;","");
```

The second step remains the same:

```
$FormattedList = $MySet.format("<ul>\n","\t<li>","</li>\n","</ul>\n");
```

Exporting Tabular data as Tab-delim or CSV

Tinderbox is perfectly capable of exporting data tables in either Tab-delimited (TSV) or Commas Separated (CSV formats). Indeed, it is possible to use the same methods to export in formats like JSON It basically boils down to two aspects:

- Tinderbox data structure allowing export of per-note data as output rows.
- suitable templates.

Structure

The best way to achieve this is to export a container using its chid notes as the source of the data rows. Less usually, descendant notes can be used to provide part of the per-row data. Depending on the nature of the task and the layout of the document, it is possible to use either a standard container note, or an agent to provide the data source.

Templates

This process needs two templates. The first, used by the source container, sets the column headings for the table. These headings are generally set as literal strings. Thus for TSV ('[Tab] implies an otherwise invisible tab character, and [Return a line return):

```
Part[Tab]Cost[Tab]Number[Tab]Total[Return]
^children("tsv-item")^
```

or as CSV:

```
"Part","Cost","Number","Total"[Return]
^children("csv-item")^
```

In either case the template exports a set of 4 literal headers and than includes each of it's children in turn using the same template (which is different to this one)

For the children, assume the following. The 'Part' is the title of the note (thus, data from $Name). The 'Cost' will be from a user Number-type attribute ($Cost) and will need formatting as a number with two decimal places. The 'Number' will be taken from a user attribute 'Ordered' (so $Ordered). Lastly the line's 'Total' will be calculated as a number with two decimal places and based on the two preceding values.

First, in TSV form:

```
^value($Name)^[tab]^value($Cost.format(2))^[tab]^value($Ordered)^[tab]^value(($Cost*$Ordered).format(2))^[Return]
```

and in CSV form

```
"^value($Name)^","^value($Cost.format(2))^","^value($Ordered)^","^value(($Cost*$Ordered).format(2))^[Return]
```

Export templates are notes

Export templates are stored as a specialised from of Tinderbox note.

A note's availability as a template is stored in attribute $IsTemplate. An easy way to set this is via the document's Properties Inspector Prototype sub-tab has a check-box, **Template**. If this is checked, $IsTemplate is set to `true`, the note's path appears in the template menu, and the note may be used as an export template. Note that an agent *cannot* also be a template.

Template notes appear in selection menus by their complete path. For example, if the template *Book Template* is a note inside the root-level container named "Templates", it appears in the menu as

```
/Templates/Book Template
```

and would be used in code thus:

```
^include("/Templates/Book Template")^
```

If a note's name (or path/name) is identical to the name of a template file, the note template will be used rather than the file.

Although a note anywhere in the document can be template (and may be in some long-lived TBXs), currently it is strongly advised to use the default location `/Templates`. As the latter is not created by default (as not everyone uses/needs export) the easiest way to make this is to use File menu ▸ Templates and select any of the built-in templates. This creates the `/Templates` folder and other configuration setting (e.g. an OnAdd making all new child notes a template, etc. You

can always delete the built-in template used to create the container if not wanted.

As Tinderbox does not require $Name to be unique, it is possible to have other containers' called 'Templates'. But, if using the default root-level container for storing templates, avoid creating a same-named container that document root. Or, i such a duplicate is needed, ensure the container holding the templates is listed first by outline order ($OutlineOrder). Why? Because when resolving a duplicate $Name or $Path—in action code reference, agent queries/Finds, etc.—Tinderbox *always uses the first match by outline order* .

Note templates can be selected from text pane's HTML tab (only if no template has been initially set) or via the Export Inspector's Export tab.

Template notes (that use the ' HTML' built-in prototype) are configured to use settings like a monospace $TextFont and disabling smart quotes, and more.

Exported files and changes to file naming

From v9.5.0, Tinderbox relaxes historic constraints on exported file names. The default replacement character in export file names, $HTMLExportFileNameSpacer, becomes a space " " character rather than an underscore (_) character.

The default value of $HTMLFileNameMaxLength becomes 100 characters, up from 8 characters in Tinderbox v1 and 24 characters in v3.0.5.

Experiments with supporting more punctuation characters showed that between restrictions for OS filenames, URIs, URLs, web servers, browsers, etc., there is no clear rule for what is punctuation that is is safe to use when exporting a Tinderbox note title as a filename. For instance, a colon is allowed in a URL, but not in a macOS filename. Thus, allowing punctuation characters other than a forward slash period and tilde (`/ . ~`) are allowed in export filenames proved to cause problems at OS and web level.

Punctuation—with the exception of "-" (hyphen) and "_" (underscore)—is replaced with $HTMLExportFileNameSpacer. Runs of punctuation characters are reduced to a single occurrence of $HTMLExportFileNameSpacer. If the filename begins or ends with punctuation, the punctuation is ignored.

Effect of changes on pre-existing Tinderbox documents export

Note that these changes to attribute default do not affect setting is pre-existing documents, where the new values must be set manually is so desired.

HTML Entities

Some source text entities/characters are automatically translated to HTML entities without needing to resort to use of the $HTMLEntities attribute:

- ellipsis (…)
- degree (°)
- French/Swiss style quotes (»Quote« or «Quote»)
- German style quotation marks („Quote")
- mdash (—)
- English typographic quotation marks “ ” and ‘ ’

Characters already already entered in the text as HTML entities are detected and exported as expected.

Since version 6 and full support for Unicode UTF-8 in export, the need for using entities has diminished. Indeed, most of the above examples are exported verbatim.

Tinderbox recognises macros embedded in text. If a paragraph contains only a macro, Tinderbox does not add paragraph mark-up to that paragraph—it assumes that the macro will do this. If you want Tinderbox to add paragraph mark-up, just add some space characters before or after the macro.

Tinderbox recognises any paragraph that contains only export template expressions (notably ^include(…)^ and ^children(…)^). Tinderbox will not add paragraph mark-up to the result of these expressions. Again, adding space characters before or after the mark-up will cause Tinderbox to add paragraph mark-up. If an export code is on a line of its own before the end of the note, do ensure you add closing ^ or the output HTML may not be as you suspect.

Tinderbox ships with specimen templates stored internally within the application package (see built-in export templates). This means you can add a few basic templates in a new TBX to see how code work, without writing your own.

HTML Export & Unicode non-UTF-8 characters

If $HTMLEntities is `true`, non-UTF-8 characters are encoded as numeric HTML entities (e.g. © as '©'). If $HTMLEntities is `false`, non-UTF-8 characters are exported as UTF-8 characters.

HTML Export - exporting folders

Sometimes it may be necessary to export content in sub-folders. For example, CSS files might go in a `/css` folder. In such cases, place the files in container off the root (or nest as many 'dummy' containers as needed).

Source container has no $Text

If the container has no $Text content, then a folder is exported with a name based on $Name and containing any exportable child notes.

A different folder name from $Name

What if the container name is not correct? It may be the case is wrong or the necessary folder name needs to differ from the source container. If so, set $HTMLExportFileName to the appropriate name using the case desired.

$Text present but no export desired

What is there is text in the container note but it is not to be exported, e.g. because it is instructions about what lives in the container? In this case set $HTMLDontExport to `true` or open the container's HTML view and un-tick the top left bo $HTMLDontExport is a little confusing as the value is the opposite of the HTML view tick-box, i.e. ticked equates to `false`, this is the reverse of the way most booleans work.

Do not edit the export extension!

Regardless of the above, even if the container is exported only as a folder **do not** be tempted delete the export extension ($HTMLExportExtension) or you get an unexpected result: the folder is created but child notes are created as its siblings. There is an arcane reason for this logic, but remember, TB wants a file extension for a container even if it is only ever exported as a folder.

Containers children are only boilerplate/includes

A reverse scenario occurs where you want to use notes in a container but not have the notes or the container export, such as when the child notes are only ever used as includes in other exported files. In this case, open the HTML View of the container and un-tick the first two boxed top left, for 'Export' and 'Export Children' (setting $HTMLDontExport and $HTMLExportChildren respectively). Meanwhile, ensure the child notes (i.e. the include(d) content), have their 'Export' box ticked in HTML view. This is slightly counter-intuitive at first meeting, but to be exported as an include, the source of the include content must be an exportable note. By placing the note in a parent container that is set to *not* export its children the correct conditions are met.

HTML Export - page(s) based on multiple notes

There are some additional considerations if exporting a page derived from multiple notes:

- Where source pages also export as discrete pages
- Inbound links
- Planning a many-note-to-single HTML page
- Setting in-page HTML anchors
- Considerations for aliases
- Traversing non-exporting notes

Where source pages also export as discrete pages

Linkage issues. To do.

Inbound links

Tinderbox will not export outbound basic or text links that terminate in a note only used as part of another note's page. There is no facility to create # anchor links.

Planning a many-note-to-single HTML page

To do!

Setting in-page HTML anchors

W3C syntax rules for the destinations of in-page jump links are thus:

*HTML tag **id** and **name** tokens must begin with a letter ([A-Za-z]) and may be followed by any number of letters, digits ([0-9]), hyphens ("-"), underscores ("_"), colons (":"), and periods (".")* . [W3C source reference]

This means that the most obvious built-in solution for a unique ID for an object, that of using $ID, is not valid as it is all digits. Likely most browsers will cope, but it is building on sand.

Another possible source is a user attribute of the Number type using the 'sequential' option, though note such attributes are not intrinsic.

If using either of the above, a simple workaround is to simply put a letter prefix before the numerical value. For instance:

```
id="x^value($ID)^"
```

A separate design issue is to ensure the calling link is able to use the desired destination anchor value. As the user has no control over links generated via ^text^, it can mean explicitly setting the $HTMLExportFileName for all notes so as to seed the `href` path of calling links so these can be post-processed to the desired version.

Considerations for aliases

Be aware that aliases can have their own basic links (inbound and outbound), and export into their equivalent outline location. Aliases always share text links (i.e. those from $Text) with their original.

Traversing non-exporting notes

Occasionally it may be needed to access data from descendant notes that are descended via notes that do not export. The latter might exist only as internal structuring of the internal document outline. Consider this layout:

```
2015 Articles
     January
          Article A
          Article B
     February
          Article C
          Article D
```

You might want the 2015 articles pages to list all articles with month heading but do not need any per-month pages. Thus the root note might export:

```
Traversing non-exporting notes
```

The latter template might have code like:

`Traversing non-exporting notes`

As per-month pages are not needed, it might seem logical to set $HTMLDontExport to `true` for the month containers. However, that means the 'month' template above is never evaluated and so the main note is missing some data. Conversely, leaving the export on generates unwanted pages for the month container notes

The secret here is to let the per-month container notes still export but to use a *blank* template, i.e. template with no code or even HTML comments, i.e. no $Text at all. As Tinderbox will not generate an empty file, no HTML exported file is emitted during export but the above templates can still work.

This scenario is an edge case that shows the flexibility offered by Tinderbox's export methods.

HTML Export - points to consider

If you regularly export a changing TBX to HTML (or other formats) there are some pointers to bear in mind:

- If you change export template code but a note using it is unchanged, the note will not be re-exported. When exporting HTML files, Tinderbox compares the contents of any existing file with the source it is about to export. If both match no export occurs. This means Tinderbox checks for changes in any attribute being exported (not just $Text) and in the template code, e.g. where the note is unchanged but the template has been modified or swapped for a different one Changes occurring in includes will be detected.
- If a note's exported filename changes (e.g. as a result of a change of note tile) the newly exported file does not result in the previous, differently-named file, being deleted. You will need to tidy up the old files in Finder. If hierarchical links are used consider the link from the parent as well as immediate siblings when working out the files affected. Roadmap view helpfully indicates incoming links allowing you to asses files elsewhere in the export that might be affected.
- Also consider the effects of $HTMLDontExport and $HTMLExportChildren; special attention is needed for notes only ever used as includes.
- The degree to which Tinderbox encodes or 'entifies' characters in the text is controlled by three primary attributes: $HTMLEntities, $HTMLMarkupText and $HTMLQuoteHTML. $AutomaticIndent allows suppression of just the mark up of quick lists (both note indenting and on export).
- Consider the effects of Tinderbox quick lists, font size auto-headings, etc. See more…
- If you change all or a significant number of notes' export template(s) it is advisable to do a full export, especially if the changes alter how navigational links set up.
- Notes with no $Text, or if $Text contains only whitespace, are considered empty and no exported file is generated, even if the note is set to export.

It is thus often a good idea to trash an existing exported set of pages before doing a full export. The export process is pretty speedy so, especially if on a local or LAN location, the loss of pages is very short-lived. If the final location is online, i can be useful to do a local export and then use FTP synching to update the online version to reflect the new output.

When exporting a single page via HTML view, also consider:

- If exporting a page that has includes, only the 'host' note's data is updated to the latest change. It is necessary to explicitly click the 'Update' button to re-compile included content from other notes.
- If exporting a single note whose exported filename has changed, see bullet point 2 above.
- If exporting a single note with exportable aliases, the aliases are not exported. A full HTML export is required to achieve this.
- an alias' HTML view exports the alias to the alias' context, including appropriate basic links and template related previous/next HTML links.

Tinderbox will export embedded images (*suspended since v5, but intended to return*). These are exported as JPG or PNG (set via HTML Preferences, 'Link images as'), using the host note's exported filename. If more than one image is in the note, the second image uses the filename suffix '-1' (before the extension), image #3 uses the suffix' '-2', etc. The images export to the the same folder as the host note. If the host note is an include, the images are placed in the including note's exported folder but the image filenames are unchanged. See also.

If the document uses a lot of images it can be useful to store them outside the TBX and link to them via the export template (or possibly) inline code). This can help reduce TBX file size and allow for better quality web images. Do bear in mind that if images are stored externally they cannot be seen directly in Tinderbox, though they could be added as file attribute links (harder to do if there is more than one image per note).

All attribute values allow for non-ASCII Unicode characters.

Aliases

Aliases export as separate files in their own outline-based place in the overall export. As such, they use their own basic links (in/out) and their original's text links (as $Text is the same). If an alias has no outbound basic links, it uses those of its original.

Aliases can be exported via HTML view, as with full HTML export, the aliases location/links are reflected in the output page.

Whilst aliases within Tinderbox cannot have children, aliases assume their original's children during export either for inclusion or separate export at the alias' location. (in effect as implied aliases themselves). This can effect recursing templates as output from an agent's aliases may recurse the original's descendants. In such conditions, it may be necessary to ad an alias detection argument. Originally a template might have the condition:

```
^if($ChildCount)^…
```

But, it may (depending on the user's intent) need to be altered to:

```
^if(!$IsAlias & $ChildCount)^
```

The new clause ensures the inclusion of child notes (and, if recursing, thus all descendants) does not occur if the current item is an alias.

HTML Export - exporting only as an include

It may be that a TBX has notes that need to be exported but whose content is only ever used as an include to another note. How to stop the source note being exported as a note in its own right (and creating a containing folder, etc.)?

To achieve this, the note needs to be inside a container, i.e. not at root level. The container should be set to *not* export its children. So, do either of:

- Set $HTMLExportChildren to `false`
- On the HTML view, un-tick the second box in the top left corner.

If the include note's container is literally that, i.e. it exists simply to hold the includes, then set the container not to export by doing either of:

- Set $HTMLDontExport to `true`
- On the HTML view, un-tick the top box in the top left corner.

As a side issue, do remember that any Tinderbox links *going into* an included note are not exported.

HTML Export: ^^value^^ vs. ^^get^^

Since v4 guidance is to shift to export attribute values using ^value($Attribute)^ rather than ^get(Attribute)^ and ^value($Attribute(Some note))^ rather than ^getFor(Some note, Attribute)^. The previous ^get^ and ^getFor^ are deprecated.

Do be aware that to get the expected value in the exported data some attribute data types, especially Date and Color require the attribute value to be enclosed in a format call:

```
^value($AccentColor)^: dark warm gray dark
^value($AccentColor.format())^: #403a35
^value($Created,"*")^: 2010-01-07T09:46:17Z
^value($Created.format("*"))^: Thu, 7 Jan 2010 09:46:17 +0100
```

An added advantage of the ^value^ method is that the exported value is an evaluated expression so it is possible to manipulate attribute data. If $MyNumber is 5:

```
^value($MyNumber)^: 5
^value($MyNumber*3)^: 15
```

Note that the evaluated result is only seen in the exported code; the source attribute(s) are not affected.

Compared to old methods like ^text^ and ^title^, ^value^ *does not* evaluate any inline ^ export code in $Text or $Name; for instance macros, linkTo and include codes. In such context, trying ^value(eval($Text))^ does not help as eval() evaluates action code rather than export code.

Referencing a template's attributes. Unlike all other notes, in an export template (i.e. $IsTemplate is `true`), referencing an attribute refers to the note being processed and *not* the template. To refer to the template is its path or name, thus $AttrName(template-name) not $AttrName.

HTML Export - alternate mark-up processor

Specifying the path to an HTML mark-up processor in $HTMLPreviewCommand results in the ^text^ command (only) being processed for HTML mark-up using the alternate processor. This allows the user to choose a Markdown or any other markup processor. The output is used both in the text Preview pane and HTML export.

A simple example is to specify the path to an installation of Markdown. This then allows Markdown-Style mark-up to be used in $Text and interpreted as such during evaluation of ^text^ for HTML export. Whereas normally *italic text* in $Text would be exported as `<i>italic text</i>`, a Markdown user would use '_italic text_' in their $Text instead.

See Markdown preview rendering for available values for $HTMLPreviewCommand.

If not empty, ^text^ passes the unprocessed text of the note to the script in $HTMLPreviewCommand instead of processing the text itself.

The 'Markdown' built-in prototype, presets notes to use Tinderbox's built-in copy of the Markdown script for easy configuration. More expert users may chose to edit this to point to their own copies of Markdown, or other such scripts.

Moving Stamps

Stamps may be moved between documents by dragging them out of the Stamps tab of the Document Inspector and dropping them into the new document's view pane. If the document receiving the stamp already has a stamp of the same name, the drag will be ignored. If the stamp refers to user attributes not present in the new document, actions involving those attributes will have no effect.

Stamps may also be dragged from the inspector list to the Finder, where they become files with the extension .tbxstamp. Dragging a .tbxstamp to a new Tinderbox document's view pane will add that stamp to the document, provided that a stamp does not already have that name.

OPML Export

Tinderbox content can be exported in OPML using appropriate templates, taking care to encode for XML standards and to encode paragraph breaks in $Text.

Zero-configuration OPML Export

Use File menu ▸ Export ▸ as Text and on the resulting dialog select the OPML option, offering a choice of scope. If not already present in the document, using this export for the first time will add the necessary built-in templates (and their required prototypes) to the document. If document scope is selected, root level back-of-house containers such as `Prototypes` will not be exported as per their default settings.

IMPORTANT: be aware that many apps offering OPML input/output use *customised variants* of the main standard. Thus it is simply not safe to assume that OMPL will 'just work' in all cases. It may be necessary make adjustments to the OMF templates to generate OPML output that reflects the (usually undocumented) non-standard OMPL aspects of some apps expected OPML input.

See also OPML Import.

Zero-configuration Scrivener (OPML) Export

To assist Scrivener users not confident with export, a similar methods as above is offered. At the Export as Text dialog, select 'Scrivener' and set the scope. As above Template and Prototypes are added as needed.

OPML standard and variations

The formal OPML (v2.0) standard is still quite loose allowing for variations. The following is a list of mappings of TB attributes to standard and common non-standard OPML attributes (non-standard tags in italics):

- $Name → text
- $Text → *_note*
- $Checked → *_status*

Note that the OPML `<expansionState>` element is not supported. Tinderbox's outline branch expand/collapse state is *not* accessible as an attribute to populate this OPML tag. However, for data validation's sake, *do* include the empty ta (`<expansionState>`) in any export templates.

Dates in OPML. If the user wishes to use the optional '`created`' OPML attribute, then RFC 822 formatted dates should be used. For example:

```
created="^value($Created.format("*"))^"
```

outputs (on aTbRef's author's system)

```
created="Thu, 14 Jan 2010 08:58:47 +0100"
```

For considerations relating to import, see OPML Import.

OPML Templates

Tinderbox supplies built-in OPML templates that can be added to a document from the File menu. In addition, the following code operators are provided to assist with OPML export:

- attributeEncode()
- ^opmlEncode()^ (now deprecated in favour of the above)

A pair of wrapper/item templates, similar to those below, were added to the built-in templates. Scrivener users get similar (OPML-based) template for export for ingest Scrivener.

Below are specimen templates for OPML export. Use the 'OPML-wrapper' template with a container note *containing* all the top level note(s) forming the actual OPML items. The 'OPML-wrapper' calls the 'OPML-item' template which then recursively calls itself to build out the OPML outline:

- OPML-wrapper
- OPML-item

Scrivener OPML

The build-in templates for Scrivener are actually exactly the same as those for OMPL. The duplication reflects the fact that many Scrivener users may not be wll versed on the 'code' aspect of export.

Setting the correct file type on export

If exporting OPML, the $HTMLExportExtension should be set to '`.opml`'. If using the built-in templates, this is done automatically via this code:

`^action($HTMLExportExtension=".opml")^` If the the same note/container is normally used for HTML export, consider removing the above code and setting the extension manually on save. Or make a note for OPML export but which uses an ^include()^ call to export the content whose default export is HTML.

OPML-wrapper

Template code. Line layout is to create correct OPML source layout. Note that you may need to adjust the name/location of the OPML-item template where it is referred to in the code; the default assumes a root/level container called 'TEMPLATES'. The ^action()^ code run in the template ensures offered filename for saving the generated data has the necessary '.opml' extension. Copy/paste code in the monospaced font:

```
<?xml version="1.0" encoding="UTF-8"?>
^action($HTMLExportExtension=".opml")^
<opml version="1.0">
<head>
<title>^value(attributeEncode($Name))^</title>
<expansionState>0</expansionState>
</head>
<body>
^children("OPML-item")^
</body>
</opml>
```

OPML-item

Template code. Line layout is to create correct OPML source layout. Note that you may need to adjust the name/location of the OPML-item template where it is referred to in the code; the default assumes a root/level container called 'TEMPLATES'. Copy/paste code in monospaced font:

[code starts]

```
^if($ChildCount)^^indent("\t",$OutlineDepth(parent)-1)^<outline^if($Text)^ text="^value(attributeEncode($Name))^" _note="^value(attributeEncode($Text))^"^else^ text="^value(attribu
^children("OPML-item")^^indent("\t",$OutlineDepth(parent)-1)^
^else^^indent("\t",$OutlineDepth(parent)-1)^<outline^if($Text)^ text="^value(attributeEncode($Name))^" _note="^value(attributeEncode($Text))^"^else^ text="^value(attributeEncode($I
^endIf^
```

[code ends: ensure to copy the line return *after* the last ^endIf^].

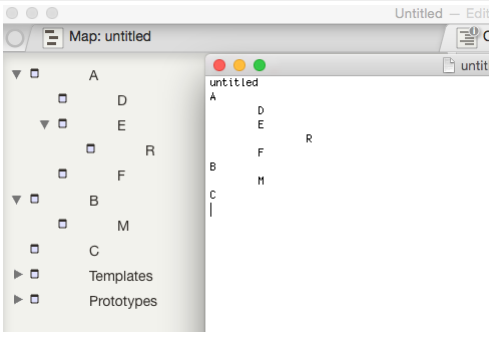
This version does not export $Checked:

```
^if($ChildCount)^^indent("\t",$OutlineDepth(parent)-1)^<outline^if($Text)^ text="^value(attributeEncode($Name))^" _note="^value(attributeEncode($Text))^"^else^ text="^value(attribu
^children("OPML-item")^^indent("\t",$OutlineDepth(parent)-1)^
^else^^indent("\t",$OutlineDepth(parent)-1)^<outline^if($Text)^ text="^value(attributeEncode($Name))^" _note="^value(attributeEncode($Text))^"^else^ text="^value(attributeEncode($I
^endIf^
```

Outline Export

The Export > Outline option exports the whole document's note titles to a plain text document. The title string used is $DisplayName (or $Name if no Display Expression is set). Attributes controlling export are honoured ($HTMLDontExport, $HTMLExportChildren) allowing non-content notes and containers like prototypes to be excluded from export.

The resulting text file has the (display) title of each exporting note indented one tab stop per outline depth level, actually $OutlineDepth-1 tab stops. Thus root level notes are not indented, their children are indented one tab, etc.



Printing from Tinderbox

From v9.6.0, printing from the text pane when multiple notes are selected is temporarily disabled. Print notes one at a time or export the selected notes to a single document and then print that.

RTF (rich text) export

Fully styled RTF $Text can be exported from the File ▸ Export sub-menu. Further options are set via the Export as Text panel. RTF export supports inline use (i.e. in $Text) of a limited range of export codes. When exporting text or RTF, Tinderbox interprets several export markup elements.

- ^include(item)^ is replaced by the styled text of the designated note.
- ^value(expression)^ is replaced by the value of the expression, though most often that is simply citing an attribute reference. For example, ^value($WordCount)^ will be replaced with the value of the word count of the exported note.
- ^if(expression)^*text*^endIf^ and ^if(expression)^*text*^else^*text*^endIf^ permit conditional export. The text between the **if** and **endIf** or the **if** and **else** codes will be exported only if the expression is `true`, otherwise the text between th (optional) **else** and **endIf** codes is exported.

Support for other app-specific formats

See here.

Tabs, Quick Lists, Fonts and Export

Normal paragraph markup is taken from $HTMLParagraphStart and $HTMLParagraphEnd.

However, for the first paragraph, $HTMLFirstParagraphStart and $HTMLFirstParagraphEnd are used. The defaults may be set to the same as above but the mechanism allows for different first paragraph styling. For instance, a house writing style might be to indent all paragraphs save the one after a heading, and so on.

Any paragraph indented with a tab is exported using $HTMLIndentedParagraphStart and $HTMLIndentedParagraphEnd.

The default value for $HTMLIndentedParagraphStart is <code><blockquote></code>.

Quick lists. Adding markers at the start of a paragraph automatically trigger HTML lists (see more detail):

- An asterisk (*) or bullet (•) results in the paragraph exporting a <code>
</code> bulleted list. Each successive paragraph with the same starting character is another list item in the same list.
- A hash symbol (#) results in the paragraph exporting an <code>
</code> numbered list. Each successive paragraph with the same starting character is another list item in the same list.

Auto-headings. Increases in body text size of whole paragraphs causes ^text^ to export those paragraphs as headings. Thus:

- 2-3 points larger is exported to HTML as <code><h3></code>
- 4-5 points gives <code><h2></code>
- 6 or more points above body copy is exported as <code><h1></code>

Generally this is a useful enhancement but it can be problematic if the text size has been enlarged only for clarity within Tinderbox itself. If the intent is for all text in a note to be the same size, but just bigger than the norm, e.g. in a 'readme' note, set the $TextFontSize for the note to the same as the display size in the note window. As the text is now the base size (for that note) auto-headings are not invoked during HTML export.

Controlling auto-markup. All these automatic features are controlled via $HTMLMarkupText with ^text^ used to export $Text. If instead ^value($Text)^ or ^text(plain)^ are used to export $Text, the above effects are by-passed *but there is n automatically generated HTML*. In the latter case, the user would have to manually insert any desired HTML mark-up, e.g. paragraph tags, directly into the $Text.

Text styling that does not export

Since Tinderbox's move to supporting a fuller RTF feature set, a number of styling enhancements were added that cannot be automatically exported via either type of export:

- Text highlighting.
- Superscript
- Subscript

Other general text styling can be HTML exported at per-note scope using CSS. However, changes to the following within a single $Text would require placing actual HTML markup (e.g. span elements) in source $Text:

- Text colour.
- Fonts.
- Font size. There is limited support for marking paragraphs in $Text as heading levels based on variation form the note's current $TextFontSize.
- Paragraph indentation (e.g. insert paragraphs). Partial support via $HTMLParagraphStart & End, but note that fully indented paragraphs receive no such special markup (slightly confusing).

See also Quick Lists.

Tinderbox metadata in pasted Tinderbox data

Developer addition. RTF and RTFD flavours added to the **com.eastgate.tinderbox.metadata** pasteboard flavour.

For Developers: when one or more notes are copied to the pasteboard, Tinderbox adds a new flavour **com.eastgate.tinderbox.metadata** for the convenience of other applications. The new flavour is a list of dictionaries, one for each selecte note, which may have the following keys:

- Name: the note's $DisplayName (NSString)
- Tags: an array of NSStrings, possibly empty, containing the NSString values of each element of the note's $Tags attribute
- Created: the note's creation date $Created (NSDate)
- Text: the note's text $Text in plain-text form(NSString)
- RTF: the note's rtf, if available; this key may be absent if styled text is not available (NSData)
- RTFD: the note's RTFD, if available; this key may be absent if styled text is not available (NSData)

UTF-8 Export

HTML Export generates files in UTF-8 format using Unix-style line endings (line feed character, '\n'). The data excludes a BOM (Byte Order Marker). Although using a BOM is supposedly the correct standards-based approach few apps have adopted it correctly resulting in some processes mishandling the file.

HTML export can export all manner of mark-up (HTML, MMD, XML, JSON, etc.) but for actual HTML it is advisable to add the following for the section of the exported page's source:

```
<meta http-equiv="content-type" content="text/html; charset=utf-8">
```

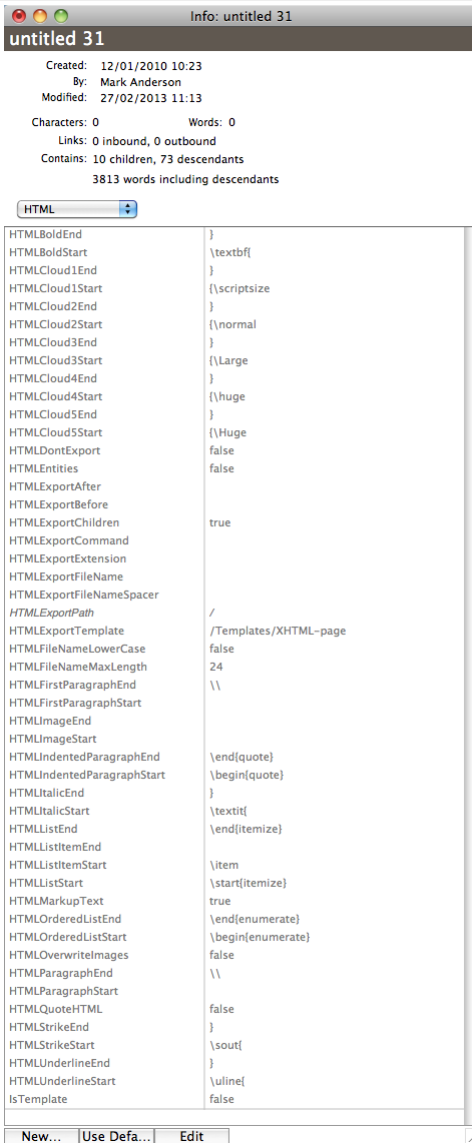
That meta element makes it even easier for ensure a web browser to understand unambiguously the encoding to expect within web page it is opening.

Working with LaTeX

Tinderbox can export, via custom templates and HTML Export, data that includes LaTeX mark-up. The limitations discussed below reflect the fact that alternate mark-up types are are a by-product of the export design rather than a deliberate feature. Tinderbox does not claim, as a standard feature, to export LaTeX.

Obtaining LaTeX output involves consideration of the template used and per-note attribute values:

- Template. This tells Tinderbox what data from the current note (or other transcluded notes) to place into the exported file. It is thus a good place to put LaTeX code such as that needed to mark the beginning or end of a file. Exactly what then gets output from ^title^ and ^text^, etc., is controlled at note level.
- Note-level attributes. By default, when exporting marked-up data, HTML is the presumed mark-up type. To use an alternate mark-up, many note-level system attributes from the HTML group need customisation, as listed below, to generate LaTeX output. It is important to note that not all HTML export features can be customised for LaTeX use.



Given the number of attributes that may require customisation, it may prove useful to employ prototypes to set up for this process. Thus, just as HTML and LaTeX export templates can be swapped over with ease, so too can LaTeX mark-up customisations be swapped by setting a prototype and letting the customisations inherit to the note(s).

Tinderbox's ^text^ export operator can manage the following $Text features as LaTeX mark-up

- bold
- italic
- underline
- strikethrough
- paragraphs (all except 'first paragraph', as explained below)
- indented paragraph (blockquote)
- lists: unordered (bulleted) and ordered (numbered) quicklists

But, do be aware that some $Text features are **not** suitable for LaTeX export. This is because the user does not have full control of the mark-up generation for some features:

- links, both internal and web links
- font size ▸ heading mapping (you cannot control this even for HTML)
- inline images.
- other RTF-like styling such as inline font changes, text colour, highlighting, sub/superscript are completely ignored for HTML/formatted mark-up export purposes.

Do not be tempted to turn off $HTMLMarkup as that then disables *all* $Text mark-up generation.

^text^ auto-markup you can re-set for LaTeX use

LaTeX coding for Underline & Strikethrough require presence of the non-default 'normalelem' package and header declaration: `\usepackage[normalem]{ulem}`. If planning to export underlined or struck-through text, remember to add this declaration in your template(s). Further detail is beyond the scope of aTbRef and should be research in LaTeX documentation.

As the changes below are set in attributes, these need only be altered only for those notes exporting to LaTeX thus allowing it to mix with default HTML export (or other export formats). If the same note may need to export to different format depending on need, it would make sense to set the attribute customisations and switch in the LaTeX values by setting the exporting note(s) to use the prototype and thus inherit its customisations.

Edit the following attributes (the quotes are not part of the code you must enter):

$HTMLBoldStart: "`\textbf{`"
$HTMLBoldEnd: "`}`"
$HTMLItalicStart: "`\textit{`"
$HTMLItalicEnd: "`}`"
$HTMLUnderlineStart: "`\uline{`"
$HTMLUnderlineEnd: "`}`"
$HTMLStrikeStart: "`\sout{`"
$HTMLStrikeEnd: "`}`"
$HTMLEntities: `false` (un-ticked)
$HTMLExportExtension: might best be set to ".tex"

All tags using `\begin`, `\end` and `\item` **must** include *a trailing single space character*. The quotes around values listed below are not part of the value and only indicate where extra white space may occur. This affects these attributes:

$HTMLIndentedParagraphStart: "`\begin{quote}` "
$HTMLIndentedParagraphEnd: "`\end{quote}` "
$HTMLListStart: "`\begin{itemize}` "
$HTMLListEnd: "`\end{itemize}` "
$HTMLOrderedListStart: "`\begin{enumerate}` "
$HTMLOrderedListEnd: "`\end{enumerate}` "
$HTMLListItemStart: "`\item` "

Set to no value (remove any existing value): $HTMLListItemEnd, $HTMLImageStart, $HTMLImageEnd.

Paragraphs. Paragraph initial indenting, and first-vs.-subsequent differences is handled in general LaTeX setup (i.e. in the template) so no paragraph start tag is needed. So set to no value: $HTMLParagraphStart, $HTMLFirstParagraphStar
However, $Text should either use line-spaced paragraphs or both $HTMLParagraphEnd and $HTMLFirstParagraphEnd should use an (invisible) line break character as the closing tag. Do not use "\n", but copy the line return from some $Te
and paste this invisibly character into the appropriate input boxes. An alternative strategy (but with different effects on LaTeX paragraph spacing is to set end paragraphs tags to "`\\`", as illustrated.

Tag clouds. The text size chosen is left to the user but these are suggested values:

$HTMLCloud1Start: "`{\scriptsize`"
$HTMLCloud1End: "`}`"
$HTMLCloud2Start: "`{\normal`"
$HTMLCloud2End: "`}`"
$HTMLCloud3Start: "`{\Large`"
$HTMLCloud3End: "`}`"
$HTMLCloud4Start: "`{\huge`"
$HTMLCloud4End: "`}`"
$HTMLCloud5Start: "`{\Huge`"
$HTMLCloud5End: "`}`"

Keyboard Shortcuts

The following abbreviations are used in this section:

- Cmd. The Command or 'Apple' key (⌘).
- Opt. The Option or Alt key (⌥).
- Ctrl. The Control key (^).

A fuller listing of keyboard names and symbols is listed separately below.

Tinderbox also supports most system-wide shortcuts, some but not all of which are included here.

Keyboard shortcuts: keys and symbols

The following abbreviations are used in this section:

- The **Command** key (**Cmd**) is located either side of the space bar. In menus this is indicated by the 4-looped symbol shown on the keyboard key (⌘). On very early keyboards this key had an Apple logo and the loop symbol and so long term Mac users sometimes still refer to it as the 'Apple', 'cloverleaf' or 'loop' key.
- The **Option** (**Opt**)—or Alt key on PCs—is located to the left of the left-hand Command key (and right-hand on extended keyboards. In menus this key is indicated by the stepped line symbol shown on the 'option' keyboard key (⌥).
- The **Control** key (**Ctrl**), located to the left of the Option key. In menus this is indicated by a caret-type symbol (^). Note, this symbol is not always shown on the keyboard's 'ctrl' key.

Other keys shown by symbol rather than name are:

- **Shift** key (large up-arrow: ⇧).
- **Return** key (horizontal u-shaped arrow: ↩.). This key is the double height key to the right of the top two rows of letter keys and is commonly called the 'Enter' key. Elsewhere, in other apps/references, the Return key may also be indicated by this alternate symbol: ↵. Thus ↩ and ↵ are the same. Neither should be confused with the formal 'Enter' key as described next, below.
- **Enter** key (line over caret symbol: ⌅ or ⍓). Do not confuse this with the separate **Return** key, above, which is often referred to colloquially as the 'Enter' key. This key is found to the bottom right of the number pad on extended keywor and, on old laptops, to the right of the right-hand Command key. Modern laptops or wireless keyboards have no Enter key, you must use Function+Return (**fn**↩).
- **Arrow** keys. Up/down/left/right (↑/↓/←/→).
- **Tab** key (⇥).
- **Escape** key (⎋).
- **Eject** key (⏏).

The format for describing shortcuts is [key1]+[key2]+[etc.]; an exception is the notes on menus where shortcuts are cited using the style shown on the menu. Where letter keys are cited they are usually shown as capitals as in the menus but it is not necessary to use the (shifted) uppercase letter, the lowercase suffices.

Individual Shortcuts

This is a listing of individual shortcuts, listed by function:

- Align Center
- Align Left
- Align Right
- Attribute Browser view type
- Bold
- Browse Links
- Cancel Export
- Cancel Link
- Chart view type
- Check Document Now
- Clear Typeahead buffer
- Close [current TBX file's name]
- Close Current Text Link
- Close Window
- Collapse
- Command Bar
- Copy
- Copy Style
- Copy View As Image
- Create Agent
- Create Child Note—as first sibling
- Create Child Note—as last sibling
- Create Note
- Create Note below selected note (Map)
- Create Note to left of selected note (Map)
- Create Note—as previous sibling
- Crosstabs view shortcut
- Cut
- Cycle Tab selection
- Cycle Tab selection (reverse)
- Dance
- Delete current item (Outline)
- Delete entire word
- Delete to line end
- Deselect All
- Displayed Attributes table - toggle display
- Document Settings
- Drag-as-last-child
- Drag-create note aliases
- Drag-create note copies
- Drill Down (Map)
- Duplicate
- Edit-in-Place: abandon title edit
- Edit-in-Place: enter mode
- Edit-in-Place: exit mode
- Edit-in-Place: exit mode
- Edit-in-Place: split title at cursor
- Expand
- Expand All Descendants
- Expand Horizontally
- Expand Vertically
- Expand View (Map)
- Explode
- Extend Outline selection downwards
- Extend Outline selection upwards
- Find and Replace
- Find Next
- Find Previous
- Find text

- Find using $Text selection
- Focus view (other views)
- Follow text link
- Force Toggle Select tool
- Full Screen mode (toggle)
- Get Info
- Go Back
- Hide others
- Hide Tinderbox
- Horizontal Map Scroll
- Hyperbolic view
- Indent
- Insert Regex Pattern
- Italic
- Jump to Selection
- Link To Selection
- Magnify
- Make Alias
- Make Web Link
- Map view type
- Minimize
- Move main view selection down
- Move main view selection up
- Move Note Down
- Move Note Up
- Move To First (Outline, Chart)
- Move To Front (Map)
- Move To Last (Outline, Chart)
- Navigate (forwards)
- New (child Note)
- New (Note)
- New TBX document
- New Treemap View
- New Window
- Next Document Tab
- Next Tab
- Open (TBX file)
- Open Help menu
- Outline - Collapse or Expand All
- Outline - Demote Selection
- Outline - Hoist
- Outline - Promote Selection
- Outline - Show or Hide siblings
- Outline view type
- Page Setup
- Park Link
- Paste
- Paste and Match Style
- Paste Style
- Previous Document Tab
- Previous Tab
- Print
- Prototype
- Quickstamp
- Quit (Tinderbox) and Keep Windows
- Quit Tinderbox
- Redo (last edit or event)
- Rename
- Resize Window retaining text pane width
- Roadmap
- Save
- Save As…
- Scroll Page Down
- Scroll Page Up
- Scroll to bottom (End)
- Scroll to end of line
- Scroll to start of line
- Scroll to top (Home)
- Select All
- Selection - Add or Remove Item
- Selection - drag select
- Selection - Expand
- Selection of Read-Only Text
- Send Behind - Reversed (Cycle Open Windows)
- Send Behind (Cycle Open Windows)
- Send To Back (Map)
- Set cursor focus in text pane
- Set focus inside a link's anchor text
- Short Date
- Short Date and Time
- Show Dictionary pop-up
- Show Emoji and Symbols
- Show or Hide (text) Ruler
- Show or Hide Displayed Attributes
- Show or Hide Fonts
- Show or Hide Inspector
- Show or Hide Prototypes
- Show or Hide Quickstamp
- Show Original
- Show Original in New Tab
- Show Ruler
- Show Spelling and Grammar
- Shrink
- Special Characters
- Split Note

- Standard Scale
- Standard Size
- Start Dictation
- Stop Creating a Link
- Strikethrough
- Text - set Black ($TextFont) text
- Text - set Blue text
- Text - set Green text
- Text - set normal colour text
- Text - set Red text
- Text - set Warm Gray text
- Text - toggle Yellow-highlighted text
- Text (pane) Only
- Text pane - select next $OutlineOrder item
- Text pane - select previous $OutlineOrder item
- Text pane - show or hide Links pane
- Text Window
- Text window - toggle focus
- Timeline view
- Tinderbox Preferences
- Toggle Displayed Attributes boolean
- Toggle Select Area tool
- Toggle Text Pane sub-tabs
- Toggle window focus
- Toggle zoomed map view
- Underline
- Underline links
- Undo (last edit or event)
- Unindent
- Update now (Agents)
- Use Filter
- Vertical Map Scroll
- View - Collapse All
- View - Expand All
- View (pane) Only
- View and Text (show both panes)

Align Center

[Cmd]+[|]

Align Left

[Cmd]+[{]

Align Right

[Cmd]+[}]

Attribute Browser view type

Cmd]+[Opt]+[A]
Change current view type to Attribute Browser view

Bold

[Cmd]+[B]

Browse Links

[Cmd]+[Opt]+[L]

Cancel Export

[Cmd]+[.]
Use during HTML or text export only.

Cancel Link

[Escape]
Used while link tool (link-drag) is active.

Chart view type

[Cmd]+[Opt]+[R]
Change current view type to Chart view

Check Document Now

[Cmd]+[;]
(semi-colon)
(check now for spelling)

Clear Typeahead buffer

[Esc]

Close [current TBX file's name]

[Cmd]+[Shift]+[W]
(hold Shift key ⇧ to see in menu)

Close Current Text Link

[Opt]+[Space]
The next text typed after this is outside the current link.

Close Window

[Cmd]+[W]

Collapse

[Cmd]+[Opt]+[Left arrow]

Command Bar

[Cmd]+[Shift]+U

Copy

[Cmd]+[C]

Copy Style

[Cmd]+[Opt]+C

Copy View As Image

[Cmd]+[Shift]+[C]

Create Agent

[Cmd]+[Shift]+[A]

Create Child Note—as first sibling

[Shift]+[Opt]+[Return]

Create Child Note—as last sibling

[Shift]+[Return]

Create Note

[Cmd]+[K]
or
[Return]
or
[Double-click] **
** In Map, Outline or Chart view only, and with no note selected. On maps you may double-click on the background or on adornments.
Note is created as next sibling. In Maps it appears to the right of the current selected note.

Create Note below selected note (Map)

[Ctrl]+[Opt]+[Return]

Create Note to left of selected note (Map)

[Ctrl]+[Return]

Create Note—as previous sibling

[Ctrl]+[Return]

Crosstabs view shortcut

[Cmd]+[Opt]+[B]

Cut

[Cmd]+[X]

Cycle Tab selection

[Ctrl]+I
Character i not numeral 1.

Cycle Tab selection (reverse)

[Ctrl]+[Shift]+I
Character i not numeral 1.

Dance

[Cmd]+[Shift]+D
Start/Stop force directed layout of map view.

Delete current item (Outline)

Backspace (Delete) key (⌫)
or
(Forwards) Delete key (⌦) (use [fn]+⌫ on laptop/wireless keyboards)
Note that the key used affects the subsequent selection. In both cases the current item is deleted but:

- Backspace: the previous ($OutlineOrder) item is selected
- Forward delete: the next ($OutlineOrder) item is selected

Delete entire word

[Opt]+[Delete]
or
[Opt]+[Forward delete]
deletes part or whole word back or forwards from the current cursor position, i.e. from cursor to next word boundary in direction of delete action.

Delete to line end

[Cmd]+[Delete]
or
[Cmd]+[Forward delete]
deletes part or whole line back or forwards from the current cursor position, i.e. from cursor to line start or end. In the case of line end it is to the Text windows soft line break and not the next hard line/paragraph break.

Deselect All

[Cmd]+[Opt]+[Ctrl]+[A]
Includes Text pane from v9.5.0

Displayed Attributes table - toggle display

[Cmd]+[Shift]+K
Show/Hide a note's Displayed Attributes table (if any)

Document Settings

[Cmd]+[8]
(number 8)

Drag-as-last-child

[Cmd]+[shift]+[drag]

Drag-create note aliases

[Cmd]+[Opt]+[drag] selected note(s)

Drag-create note copies

[Opt]+[drag] selected note(s)

Drill Down (Map)

[Down arrow]+[Selected note]
[Cmd]+[Opt]+[Down arrow]+[Selected note]
[Double-click]+[viewport part of containers and agents]
Same action as Focus view, and logical opposite of Expand View. Acts like a 'Down Arrow' for Map views. Only works if a note has active focus. There is no menu reference to this shortcut.
All map shortcuts work even if a container has its viewport hidden by expanding the title area.

Duplicate

[Cmd]+[D]
or
[Opt]+drag or click

Edit-in-Place: abandon title edit

[Esc]

Edit-in-Place: enter mode

[Fn]+[Return]
[Ctrl]+C
(with selected note in view pane)

Edit-in-Place: exit mode

[Enter]
or
[Return]

Edit-in-Place: exit mode

[Return]

Edit-in-Place: split title at cursor

[Ctrl]+[Return]

Expand

[Cmd]+[Opt]+[Right arrow]

Expand All Descendants

[Option]+[click] on Outline view expand widget
This expands the container (note) clicked on and does so for any other container notes amongst the clicked note's descendants.

Expand Horizontally

[Cmd]+[Ctrl]+[Right arrow]

Expand Vertically

[Cmd]+[Ctrl]+[Down arrow]

Expand View (Map)

[Up arrow]
[Cmd]+[Up arrow]
For opposing command, see Drill Down.

Explode

[Cmd]+[Shift]+[E]
Explode current note
Original shortcut ([Cmd]+[Opt]+[E]) was re-assigned to text sub-pane toggling

Extend Outline selection downwards

[Shift]+[down arrow]

Extend Outline selection upwards

[Shift]+[up arrow]

Find and Replace

[Cmd]+[Opt]+[F]

Find Next

[Cmd]+[G]

Find Previous

[Cmd]+[Shift]+[G]

Find text

[Cmd]+[F]

Find using $Text selection

[Cmd]+E
Uses $Text selection, sets Find string to current selected text for both view and text pane Find (regardless of whether Find toolbar(s) are currently shown).

Focus view (other views)

[Double-click]+[Outline/chart note icon]
[Double-click]+[Treemap item]
For maps, see Drill Down.
Only works for notes with children. Double-click must occur on the note icon, not the badge or note title.

Follow text link

[click]
(click on link)

Force Toggle Select tool

Press and hold [Cmd] key (⌘).

In maps, charts and outline views, press and hold the command key to force to use the grabby hand cursor, even when pointing to an adornment or a note.

Full Screen mode (toggle)

[Ctrl]+[Cmd]+F
Toggles enter/leave full screen mode

Get Info

[Cmd]+[Opt]+[I]
(letter i not number 1)

Go Back

Cmd+'
(single quote)

Hide others

[Cmd]+[Opt]+[H]

Hide Tinderbox

[Cmd]+[H]

Horizontal Map Scroll

[Mouse wheel]+[Shift]

Hyperbolic view

[Cmd]+[Opt]+[W]

Indent

[Cmd]+[]]
i.e. ']'
In Outline and Chart views this makes the note a child of its previous sibling.
From v6.4.0, if the cursor is in the text pane the current $Text paragraph is indented.

Insert Regex Pattern

[Cmd]+[Opt]+[Ctrl]+[P]
(only when (a) text pane has focus and (b) find toolbar is displayed.

Italic

[Cmd]+[I]
(letter i not figure 1)

Jump to Selection

[Cmd]+[J]

Link To Selection

[Cmd]+[Ctrl]+[L]
Make untyped link from current $Text selection to note with case-sensitive $Name match to selection.

Magnify

[Cmd]+[=]
(equals)

Make Alias

[Cmd]+[L]
or
[Opt]+[Shift]+drag or click

Make Web Link

[Cmd]+[Opt]+[Ctrl]+[L]

Map view type

[Cmd]+[Opt]+[M]
Change current view type to Map view

Minimize

[Cmd]+[M]

Move main view selection down

[Cmd]+[Opt]+[Down arrow]
(focus in text pane)

Move main view selection up

[Cmd]+[Opt]+[Up arrow]
(focus in text pane)

Move Note Down

[Cmd]+[Down arrow]

Move Note Up

[Cmd]+[Up arrow]

Move To First (Outline, Chart)

[Cmd]+[Shift]+[Up arrow]

Move To Front (Map)

[Cmd]+[Shift]+[Up arrow]

Move To Last (Outline, Chart)

[Cmd]+[Shift]+[Down arrow]

Navigate (forwards)

[Cmd]+Return

New (child Note)

[Shift]+[Return]
(the new note created will be a child of the currently selected note)

New (Note)

[Cmd]+[N]
or
[Return]
or
[double-click]

New TBX document

[Cmd]+[N]

New Treemap View

[Cmd]+[Opt]+[Shift]+[T]

New Window

[Cmd]+[Shift]+[N]

Next Document Tab

[Ctrl]+[Tab]
Selects the next (right) document in the document tab bar.

Next Tab

[Cmd]+[Opt]+[]]

Open (TBX file)

[Cmd]+[O]
(letter o not number zero)

Open Help menu

[Cmd]+[Shift]+[/]
(i.e. [Cmd]+[?])
OS-wide Shortcut may vary on non-English language keyboards.

Outline - Collapse or Expand All

[Cmd]+[Opt]+[click]+[disclosure triangle]
The disclosure triangle must be at *root* level for this to work, otherwise the current note and siblings, i.e. the whole current 'branch' is expanded/collapsed.

Outline - Demote Selection

[Tab]
Works with a single or multiple selections.

Outline - Hoist

Hoist == Focus view
[Outline note icon]+[Double-click]

Outline - Promote Selection

[Shift]+[Tab]
Works with a single or multiple selections.

Outline - Show or Hide siblings

[Cmd]+[click]+disclosure triangle
Expands/collapses all siblings one level. Siblings (same level) of other parents are not affected nor are grandchildren exposed.

Outline view type

[Cmd]+[Opt]+[O]
(letter o not number zero)
Change current view type to Outline view

Page Setup

[Cmd]+[Shift]+[P]

Park Link

[Cmd]+[Shift]+[L]

Paste

[Cmd]+[V]

Paste and Match Style

[Cmd]+[Opt]+[Shift]+[V]

Paste Style

[Cmd]+[Opt]+[V]

Previous Document Tab

[Ctrl]+[Shift]+[Tab]
Selects the previous (left) document in the document tab bar.

Previous Tab

[Cmd]+[Opt]+[[]

Print

[Cmd]+[P]

Prototype

[Cmd]+[3]
(Document Inspector, Prototype tab)

Quickstamp

[Cmd]+[2]
(Document Inspector, Quickstamp tab)

Quit (Tinderbox) and Keep Windows

[Cmd]+[Opt]+[Q]
(hold Opt to see in menu)

Quit Tinderbox

[Cmd]+[Q]

Redo (last edit or event)

[Cmd]+[Shift]+[Z]

Rename

[Enter] **
[Fn]+[Return] (newer laptops with no Enter key)
[Cmd]+[Shift]+[Return]
** The Enter key (⌅) *is different from* the Return key (↩) [sic]. On a full Mac keyboard with a right-side numeric keypad, the Enter key is the bottom right corner of the numeric keypad. On older Mac laptops the Enter key is two keys right of the spacebar (i.e. there the right-hand Ctrl key is on a full keyboard).
On current Macs most external keyboards have no numeric keypad and laptops omit the Enter key (⌅). In these contexts either of the latter two shortcuts above can substitute for the Enter key alone. Even with an Enter key available any of the shortcuts may be used.

Resize Window retaining text pane width

[Cmd]+drag window edge

Roadmap

[Cmd]+[Opt]+[Ctrl]+[R]

Save

[Cmd]+[S]

Save As...

[Cmd]+[Opt]+[Shift]+[S]
(hold Opt to see in menu)

Scroll Page Down

[Ctrl]+[Down Arrow]
(text pane)

Scroll Page Up

[Ctrl]+[Up Arrow]
(text pane)

Scroll to bottom (End)

[Ctrl]+[D]

Scroll to end of line

[Ctrl]+[Right Arrow]

Scroll to start of line

[Ctrl]+[Left Arrow]

Scroll to top (Home)

[Ctrl]+[A]

Select All

[Cmd]+[A]

Selection - Add or Remove Item

[Cmd]+[click]
The clicked item is added to the selection. If already selected, the item clicked is removed from the selection. Selecting a container does not select/de-select its contents.

Selection - drag select

[Opt]+[drag]
On maps, a drag select cannot be initiated within an adornment (including pictures) and locked adornments are not selected.
To select text in read-only notes, use [Cmd]+[drag]
To select text in in the output pane of HTML/Text Export views, use [Opt]+[drag]

Selection - Expand

[Shift]+[click]
In Outline view extends selection from existing selection to clicked note and all in between (excluding hidden children/descendants).
In all other views, the click adds to from the current selection (whilst also deselecting the last shift+click added item).

Selection of Read-Only Text

[Cmd]+[drag-select]

Send Behind - Reversed (Cycle Open Windows)

[Cmd]+[Shift]+[`]
(backtick / grave accent key, left of 'Z' key)
Reverses normal order of cycle.
Repeat use cycles through all open windows, back sending the current window to the back of the (Tinderbox) window stack.

Send Behind (Cycle Open Windows)

[Cmd]+[`]
(backtick / grave accent key, left of 'Z' key)
Addition of Shift key (⇧) reverses cycle order.
Repeat use cycles through all open windows, back sending the current window to the back of the (Tinderbox) window stack.

Send To Back (Map)

[Cmd]+[Shift]+[Down arrow]

Set cursor focus in text pane

[Spacebar]
(Main view only)

Set focus inside a link's anchor text

[Opt]+click

Short Date

[Cmd]+[/]
(forward slash)

Short Date and Time

[Cmd]+[Opt]+[/]
(forward slash)

Show Dictionary pop-up

[Cmd]+[Ctrl]+D+selection in $Text
Show OS Dictionary pop-up

Show Emoji and Symbols

[Ctrl]+[Cmd]+Spacebar
Show the OS 'Emoji & Symbols' palette. (Palette is closed manually)

Show or Hide (text) Ruler

[Cmd]+[Ctrl]+R

Show or Hide Displayed Attributes

[Cmd]+[Shift]+[K]
(selected item in main view)

Show or Hide Fonts

[Cmd]+[T]
Toggles OS X Fonts palette.

Show or Hide Inspector

[Cmd]+[1]
(figure 1 not letter i)

Show or Hide Prototypes

[Cmd]+3
(Document Inspector, Prototype tab)

Show or Hide Quickstamp

[Cmd]+2
(Document Inspector, Quickstamp tab; focus goes into attribute search box)

Show Original

[Cmd]+[R]

Show Original in New Tab

[Cmd]+[Shift]+[R]

Show Ruler

[Cmd]+[Ctrl]+[R]

Show Spelling and Grammar

[Cmd]+[:]
(colon)

Shrink

[Cmd]+[-]
(minus)

Special Characters

[Cmd]+[Ctrl]+[spacebar]

Split Note

[Cmd]+[Opt]+[Ctrl]+Return
(focus must be in $text area of text pane)

Standard Scale

[Cmd]+[0]
(zero)

Standard Size

[Cmd]+[Shift]+[T]

Start Dictation

[fn],[fn]
Tap fn key twice

Stop Creating a Link

[Esc]
Used while a link-drag is active. Link creation ceases and link line disappears.

Strikethrough

[Cmd]+[Shift]+[minus]
The hyphen key can be used instead of the number pad minus key, especially on laptop keyboard with no number pad.

Text - set Black ($TextFont) text

[Cmd]+[Ctrl]+[5]
…when cursor is in $Text. Works on *selected* text. If $TextFont is not black, $TextFont colour is applied.

Text - set Blue text

[Cmd]+[Ctrl]+[3]
…when cursor is in $Text. Works on *selected* text.

Text - set Green text

[Cmd]+[Ctrl]+[2]
…when cursor is in $Text. Works on *selected* text.

Text - set normal colour text

[Cmd]+[Ctrl]+[0]
…when cursor is in $Text. Works on *selected* text.

Text - set Red text

[Cmd]+[Ctrl]+[1]
…when cursor is in $Text. Works on *selected* text.

Text - set Warm Gray text

[Cmd]+[Ctrl]+[3]
…when cursor is in $Text. Works on *selected* text.

Text - toggle Yellow-highlighted text

[Cmd]+[Shift]+[Y]
…when cursor is in $Text. Works on selected text. Adds highlight if currently none, removes highlight it is already present.

Text (pane) Only

[Cmd]+[4]
(hide main view pane)

Text pane - select next $OutlineOrder item

[Cmd]+[Opt]+[Down Arrow]
With focus in text pane, select next ($OutlineOrder) item in main view. Active focus remains in $Text area.
(Outline/Chart/Treemap view only)

Text pane - select previous $OutlineOrder item

[Cmd]+[Opt]+[Up Arrow]
With focus in text pane, select previous ($OutlineOrder) item in main view. Active focus remains in $Text area.
(Outline/Chart/Treemap view only)

Text pane - show or hide Links pane

[Cmd]+[7]

Text Window

[Cmd]+[Opt]+[X]

Text window - toggle focus

[Opt]+[Tab]
Toggles focus from the note text pane to the first listed Displayed Attribute (if any). This shortcut toggles between attributes, text and the view pane. To toggle out of the Displayed Attributes, an attribute value box must not be in edit mode.

Timeline view

[Cmd]+[Opt]+[Ctrl]+[T]
(change current tab's main view type)

Tinderbox Preferences

[Cmd]+[comma]

Toggle Displayed Attributes boolean

[spacebar]
selected boolean is toggled to opposite state.

Toggle Select Area tool

[Opt]+drag
(Map view only)

Toggle Text Pane sub-tabs

[Cmd]+[Opt]+[E]
Cycles focus of Text panes(even if sub-tab selector row is hidden)
Previously this opened Explode (now [Cmd]+[Shift]+[E]

Toggle window focus

[Opt]+[Tab]
(cycles $Text → Displayed Attributes (if present) → main view → $Text)

Toggle zoomed map view

[Cmd]+[Opt]+[Ctrl]
(zooms map to show all items)

Underline

[Cmd]+[U]

Underline links

[Cmd]+[Opt]
(focus in text area)

Undo (last edit or event)

[Cmd]+[Z]
Only available for some events, such as text editing.

Unindent

[Cmd]+[[]
i.e. '['
In Outline and Chart views this makes the note the next sibling of its existing parent.

If the cursor is in the text pane the current $Text paragraph is unindented.

Update now (Agents)

[Cmd]+[Ctrl]+[=]
(equals)
Updates agents (and edicts, once)

Use Filter

Toggle view pane filter
[Cmd]+[Opt]+[Shift]+F

Vertical Map Scroll

[Mouse wheel]

View - Collapse All

[^]+[⌘]+[0]
(v9.5.0)

View - Expand All

[^]+[⌘]+[9]
(v9.5.0)

View (pane) Only

[Cmd]+[6]
(hide text pane)

View and Text (show both panes)

[Cmd]+[5]
(show main view and text)

Reverse Look-up Map

This page lists all Tinderbox keyboard shortcuts ordered by the action key used with all variants (Cmd, Opt, etc.) grouped together.
Abbreviations used:

- Cmd: Command (⌘)
- Opt: Option (Alt) (⌥)
- Ctrl: Control (^)
- Esc: Escape (⎋)
- Fn: Function (fn)

Shortcuts - Reverse Listing:

A+[Cmd]: Select All
A+[Ctrl]: Scroll to top of window (Home)
A+[Cmd]+[Shift]: Create Agent
A+[Cmd]+[Opt]: Attribute Browser view (change current tab's main view type)
A+[Cmd]+[Opt]+[Ctrl]: Deselect All. View and text panes (hold down all 3 modifiers to see in Edit menu)

B+[Cmd]: Bold
B+[Cmd]+[Opt]: Crosstabs view (change current tab's main view type)

C+[Cmd]: Copy
C+[Ctrl]: Enter Edit-in-Place (view pane's selected note)
C+[Cmd]+[Opt]: Copy Style
C+[Cmd]+[Shift]: Copy View As Image

D+[Cmd]: Duplicate
D+[Ctrl]: Scroll to bottom of window (End)
D+[Cmd]+Shift]: Dance (map view only)
D+[Cmd]+[Ctrl]+Selection: show Dictionary pop-up (selected $Text)

E+[Cmd]: Use current selection for Find
E+[Cmd]+[Opt]: Cycle Text pane sub-tab selection (Text→Preview→Export→etc.) even if tab selectors are hidden
E+[Cmd]+[Shift]: Explode

F+[Cmd]: Find
F+[Cmd]+[Opt]: Find and Replace
F+[Cmd]+[Ctrl]: Enter/Leave Full Screen mode (toggle)

G+[Cmd]: Find Next
G+[Cmd]+[Shift]: Find Previous

H+[Cmd]: Hide Tinderbox
H+[Cmd]+[Opt]: Hide Others (all other open apps)

I+[Cmd]: Italic
I+[Ctrl]: Cycle Tab selection
I+[Cmd]+[Opt]: Get Info
I+[Ctrl]+[Shift]: Cycle Tab selection (reverse)

J+[Cmd]: Jump to Selection

K+[Cmd]+[Shift]: Show/Hide Displayed Attributes (selected item in main view)

L+[Cmd]: Make Alias
L+[Cmd]+[Opt]: Browse Links
L+[Cmd]+[Shift]: Park Link (from current $Text selection)
L+[Cmd]+[Ctrl]: Link to Selection (untyped link from current $Text selection to note with case-sensitive $Name match)
L+[Cmd]+[Opt]+[Ctrl]: Make Web Link

M+[Cmd]:Minimize
M +[Cmd]+[Opt]: Map view (change current tab's main view type)

N+[Cmd]: New (TBX Document)
N+[Cmd]+[Shift]: New Window (for current document)

O+[Cmd]: Open (TBX Document)
O+[Cmd]+[Opt]: Outline view (change current tab's main view type)

P+[Cmd]: Print
P+[Cmd]+[Shift]: Page Setup
P +[Cmd]+[Opt]+[Ctrl]: Insert Regex Pattern. **NOTE**: Text pane, *only if find toolbar displayed* .

Q+[Cmd]: Quit Tinderbox
Q+[Cmd]+[Opt]: Quit and Keep windows (hold Opt key to see in menu)

R+[Cmd]: Show Original
R+[Cmd]+[Shift]: Show Original in New Tab
R+[Cmd]+[Opt]: Chart view (change current tab's main view type)
R+[Cmd]+[Ctrl]: Show Ruler (if focus in text pane)
R+[Cmd]+[Opt]+[Ctrl]: Roadmap

S+[Cmd]: Save
S+[Cmd]+[Shift]: Duplicate (File menu)
S+[Cmd]+[Opt]+[Shift]: Save As (File menu - hold Opt key to see in menu)

T+[Cmd]: Show/Hide Fonts
T+[Cmd]+[Opt]: Show/Hide Toolbar (for current window)
T+[Cmd]+[Shift]: Standard Size (set selection to default $TextFontSize)
T+[Cmd]+[Opt]+[Ctrl]: Standard Font (set selection to default $TextFont)

U+[Cmd]: Underline
U+[Cmd]+[Shift]: Open Command Bar (use Esc to close)

V+[Cmd]: Paste
V+[Cmd]+[Opt]: Paste Style
V+[Cmd]+[Opt]+[Shift]: Paste and Match Style

W+[Cmd]: Close Window
W+[Cmd]+[Opt]: Hyperbolic View
W+[Cmd]+[Shift]: Close current TBX file (hold Shift key to see in menu)

X+[Cmd]: Cut
X+[Cmd]+[Opt]: Text Window (open current note text pane as tear-off window)

Y+[Cmd]+[Shift]: (when cursor is in $Text). Toggles yellow highlight on selected $Text.

Z+[Cmd]: Undo (last event)
Z+[Cmd]+[Shift]: Redo (last event)

1+[Cmd]: Show/Hide Inspector
1+[Cmd]+[Ctrl]: $Text pane - Set selected text Red

2+[Cmd]: Show/Hide Quickstamp (Document Inspector, Quickstamp tab - focus goes to search box)
2+[Cmd]+[Ctrl]: $Text pane - Set selected text Green

3+[Cmd]: Show/Hide Prototypes (Document Inspector, Prototype tab)
3+[Cmd]+[Ctrl]: $Text pane - Set selected text Blue

4+[Cmd]: Text Only (hide main view pane). Focus switches to text pane.
4+[Cmd]+[Ctrl]: $Text pane - Set selected text Gray

5+[Cmd]: View and Text (show both main view and text panes)
5+[Cmd]+[Ctrl]: $Text pane - Set selected text Black (or $TextFont if not black)

6+[Cmd]: View Only (hide text pane). Focus switches to view pane if it was in the text pane
---
7

8+[Cmd]: Document Settings

9+[Cmd]+[Ctrl]: View ▸ Expand All (v9.5.0)

0+[Cmd]: Standard Scale
0+[Cmd]+[Ctrl]: View ▸ Collapse All (v9.5.0)

\

/+[Cmd]: Insert Short Date (in Note text)
/+[Cmd]+[Opt]: Insert Short Date and Time (in Note text)
/+[Cmd]+[Shift]: Open Help Menu (= [Cmd]+[?]) - main views only
---
:+[Cmd]: Show Spelling and Grammar

;+[Cmd]: Check Document Now (for spelling)

,+[Cmd]: Tinderbox Preferences

`+[Cmd]: Send Behind (Cycle Open Windows)
`+[Cmd]+[Shift]: Send Behind, reverse order (Cycle Open Windows)

' (single quote)
'+[Cmd]: Go Back. Navigate back along last traversed link.
- (use hyphen key or number pad minus key)
-+[Cmd]: Shrink
-+[Cmd]+[Shift]: Strikethrough

.+[Cmd]: Cancel Export (during HTML or text export)

?+[Cmd]: Open Help menu (= [Cmd]+[Shift]+[/])

=+[Cmd]: Magnify
=+[Cmd]+[Ctrl]: Update Now (Agents). Also updates all edicts, once.

[+[Cmd]+[Opt]: Previous Tab

]+[Cmd]+[Opt]: Next Tab

{+[Cmd]: Align Left

}+[Cmd]: Align Right

|+[Cmd]: Align Center

Spacebar - Main view: Set cursor focus in text pane
Spacebar+[Cmd]+[Ctrl]: Show the system Special Characters dialog ('Emoji & symbols')
Spacebar - Displayed Attributes table or Get Info/attributes: Toggle state of boolean attribute
---
Left Arrow: Select Previous Sibling Note - Outline Order (Map)
Left Arrow: Select Parent (Chart).
Left Arrow: Collapses the selected note if expanded. Otherwise, it selects the parent of the selected note if the parent is visible in the view (Outline).
Left Arrow+[Cmd]: Move cursor to start of line (in $Text pane)
Left Arrow+[Cmd]+[Shift]: Extend selection to start of line (in $Text pane)
Left Arrow+[Opt]: Move cursor to start of current word, then start of previous word (in $Text pane)
Left Arrow+[Cmd]+[Opt]: Collapse (Outline & Chart views)

Right Arrow: Select Next Sibling Note - Outline Order (Map)
Right Arrow: Select First Child (Outline & Chart) - or collapse container if no child; expands the selected container note
Right Arrow+[Cmd]: Move cursor to end of line (in $Text pane)
Right Arrow+[Cmd]+[Shift]: Extend selection to end of line (in $Text pane)
Right Arrow+[Cmd]+[Ctrl]: Expand Horizontally
Right Arrow+[Opt]: Move cursor to end of current word, then end of next word (in $Text pane)
Right Arrow+[Cmd]+[Opt]: Expand (Outline & Chart views)

Up Arrow: Expand View (Map)
Up Arrow: Select Previous (Visible) Note (Outline & Chart)
Up Arrow: Select First (Visible) Note (Outline, if no note selected)
Up Arrow+[Cmd]: Move Note Up (Chart/Outline)
Up Arrow+[Cmd]: Expand View (Map)
Up Arrow+[Cmd]+[selection]: Expand View (other view types)
Up Arrow+[Ctrl]: Scroll Page Up
Up Arrow+[Shift]: Extend Outline view selection upwards by one note
Up Arrow+[Cmd]+[Opt]: Text pane focus: select previous $OutlineOrder item in main view (Outline/Chart/Treemap view)
Up Arrow+[Cmd]+[Opt]: Expand view (Map view)
Up Arrow+[Cmd]+[Shift]: Move To Front
Up Arrow+[Cmd]: Move cursor to start of $Text (in $Text pane)
Up Arrow+[Cmd]+[Shift]: Extend selection to start of $Text (in $Text pane)
Up Arrow+[Opt]: Move cursor to beginning of current paragraph, then beginning of previous paragraph (in $Text pane)
Up Arrow+[Fn]: Outline view, scroll one page (screen) up
Up Arrow+[Fn]+[Shift]: Outline view, go to Home

Down Arrow: Drill Down (Map - *with a note icon selected*)
Down Arrow: Select Next (Visible) Note (Outline & Chart)
Down Arrow: Select First (Visible) Note (Outline, if no note selected)
Down Arrow+[Cmd]: Move Note Down (Chart/Outline)
Down Arrow+[Cmd]: Map view + selection: Drill Down
Down Arrow+[Ctrl]: Scroll Page Down
Down Arrow+[Shift]: Extend Outline view selection downwards by one note
Down Arrow+[Cmd]+[Opt]: Text pane focus: select next $OutlineOrder item in main view (Outline/Chart/Treemap view)
Down Arrow+[Cmd]+[Opt]: Drill Down (Map)
Down Arrow+[Cmd]+[Ctrl]: Expand Vertically
Down Arrow+[Cmd]+[Shift]: Send To Back
Down Arrow+[Cmd]: Move cursor to end of $Text (in $Text pane)
Down Arrow+[Cmd]+[Shift]: Extend selection to end of $Text (in $Text pane)
Down Arrow+[Opt]: Move cursor to end of current paragraph, then end of next paragraph (in $Text pane)
Down Arrow+[Fn]: Outline view, scroll one page (screen) down
Down Arrow+[Fn]+[Shift]: Outline view, go to End

Delete Back: Backspace key
Delete Back - Outline: Delete current item and select *next* item
Delete Back+[Cmd]: Text window - Delete to beginning of line (from cursor)
Delete Back+[Opt]: Text window - Delete to beginning of word (from cursor) or delete previous word
---
Delete Forward: if no key, use Delete key (extended keyboard) or [Fn]+Backspace on laptop or wireless keyboards
Delete Forward - Outline: Delete current item and select *next* item
Delete Forward+[Cmd]: Text window - Delete to end of line (from cursor)
Delete Forward+[Opt]: Text window - Delete to end of word (from cursor) or delete next word
---
Return: Create Note (as next sibling)
Return: Edit-in-place mode: Exit mode
Return: Map view: create new note to right of current selection
Return: Roadmap: follow and set focus on selected link
Return+[Cmd]: Navigate (forwards). Follow the first listed link in Browse Links
Return+[Shift]: Create child note as last sibling
Return+[Shift]: Edit-in-place mode: exit and create a new note
Return+[Opt]+[Ctrl]: Create new note below selected note (Map view)
Return+[Ctrl]: Create new note to left of selected note (Map view)
Return+[Ctrl]: Create note as previous sibling (Outline, Chart view)
Return+[Ctrl]: Edit-in-place mode: split title at insertion point (i.e. insert line break in title)
Return+[Cmd]+[Shift]: Rename selected note (enter edit mode)
Return+[Opt]+[Shift]: Create child note as first sibling
Return+[Cmd]+[Opt]+[Ctrl]: Split the current note at the insertion cursor position (focus in $Text area of text pane)
Return+[Fn]: Rename selected note (enter edit mode)

Double-click: Create Note (in Outline/Map/Chart when no note selected)
Double-click: Open Note (in Outline/Map/Chart when a note selected) note a different result if an Outline *icon* is double-clicked

Double-click+[Outline Note icon]: Focus view ('Hoist' in older versions)
Double-click+Roadmap (clicking in in/outbound link lists): shift view focus to destination note
Double-click+[selected Map container icon]: Drill down (double click must be in *viewport* area of icon)

Tab - Outline: Demote current selection
Tab: Roadmap: cycles from first inbound link → first outbound link → link type → etc.
Tab+[Opt]: Toggle focus: $Text → first Displayed Attribute → main view → $Text
Tab+[Shift]: Outline - Promote current selection
Tab+[Ctrl]: Select next document tab
Tab+[Ctrl]+[Shift]: Select previous document tab

Enter: Rename selected note (enter edit mode)
Enter: Enter/Leave Edit-in-place mode (selected note in main view)

Esc: Stop creating a link (when link drag is active)
Esc: Clear typeahead buffer (major/minor views using typeahead)
Esc: Edit-in-place mode: exit edit mode, ignore current edits
Esc+[Opt]: invoke OS text autocomplete (view and text pane)

Home: Scroll $Text to start of note $Text without moving insertion point (in $Text pane)

End: Scroll $Text to end of note $Text without moving insertion point (in $Text pane)

Page Up: Scroll $Text up one page, by $Text pane size, without moving insertion point (in $Text pane)

Page Down: Scroll $Text up down page, by $Text pane size, without moving insertion point (in $Text pane)

[Cmd]: Force toggle Select tool
[Cmd]+drag-select: Select text in read-only notes.
[Cmd]+drag window edge: resize document window retaining text pane width.
[Cmd]+[Opt]: Underline links (Note text windows only)
[Cmd]+[Shift]+drag: add to container as last ($OutlineOrder) sibling(s); default is to be added as first sibling
[Cmd]+[Opt]+[click]: (in link anchor text) follow link, i.e. open destination note
[Cmd]+[Opt]+[Ctrl]: Toggle zoomed map view (show all contents)

[Option]: Toggle Select Area tool
[Option]+[click]: Text pane, click inside $Text link anchor without following link
[Option]+[drag or click]: View pane, duplicate clicked note (Map, Outline, possibly other views)
[Option]+[Shift]+[drag or click]: View pane, make alias of clicked note (Map, Outline, possibly other views)

[Ctrl]+[click]: Open content menu (OS feature)

Outline disclosure triangle+[click]+[Cmd]: Show/hide children of this and all sibling containers *after* this in $OutlineOrder (toggle)
Outline disclosure triangle+[click]+[Cmd]+[Opt]: Expand/Collapse All Containers (toggle)
Outline disclosure triangle+[click]+[Opt]: Expand All Descendants (toggle)

View selection+[click]+[Cmd]: Add/remove item
View selection+[click]+[Shift]: Expand selection

Mouse wheel - Vertical Map Scroll
Mouse wheel+[Shift]: Horizontal Map scroll
Mouse wheel+[Ctrl]: zoom screen (OS feature: System Prefs ‣ Accessibility)

[Outline icon]+[Double-click]: Focus view ('Hoist' in older versions)

Trackpad+[drag]: scroll view

[drag]+[Shift]: Timeline view maintain date (horizontal position) dragging between timebands (Shift must be pressed *before* commencing drag)
[drag]+[Shift]: Map view - constrain to horizontal/vertical movement and disable guides (Shift must be pressed *before* commencing drag)

[Fn] x 2: Start Dictation (tap key twice for OS Dictation function)

[F5]: View pane find bar, show suggested autocompletions

Unicode Codes for Keyboard symbols

Illustrated are the correct Unicode code numbers for a variety of Mac keyboard related symbols.
The easiest way to use these via a keyboard is to open System Preferences' International/Language section (name varies with OS version) and in the Input sources list tick the option for "Unicode Hex input"). Then with that input mode selected, type Opt+[number].
For example: `Opt+2318` gives ⌘ (the Cmd key symbol). Or, to write them a HTML entity codes: ⌘.
When switching input methods, do not forget when done to switch back to your normal language!
In more recent macOS versions, a triangle icon is replacing the arrow in shortcut renderings
Examples:
⌘ Command (Cmd) `U+2318`
⌥ Option (Opt or Alt) `U+2325`
^ Control (Ctrl) `U+2303`
⇧ Shift `U+21E7`
⇪ Caps Lock `U+21EA`
↩ Return `U+21A9`
⌅ Enter `U+2324` (laptop keyboards have no Enter key, use Fn+Return)
⌫ Delete (Backspace) `U+232B`
⌦ Forward Delete `U+2326`
⎋ Escape (Esc) `U+238B`
⏏ Eject `U+23CF`
⏻ Power `U+2333D`
⇥ Tab `U+21E5`
⇞ Page Up `U+21DE`
⇟ Page Down `U+21DF`
↖ Home `U+2196`
↘ End `U+2198`
← Left Arrow `U+2190`
→ Right Arrow `U+2192`
↑ Up Arrow `U+2191`
↓ Down Arrow `U+2193`
‣ Black right pointing small triangle `U+25B8` (used for menu ‣ sub-menu indicators)
fn Function (Fn key): type letters 'fn'
In more recent macOS, the Function and arrow keys are replaced with newer symbols in menus, etc.:

Function (Fn key) `U+1F310` (normally rendered in black and white)
◀ Left Arrow `U+25C0`
▶ Right Arrow `U+25B6`
▲ Up Arrow `U+25C2`
▼ Down Arrow `U+25BC`
Footnote: in most Mac fonts, Opt+Shift+K will insert the Apple logo, e.g. .

⌘ PLACE OF INTEREST SIGN (U+2318)
⌥ OPTION KEY (U+2325)
^ UP ARROWHEAD (U+2303)
⇧ UPWARDS WHITE ARROW (U+21E7)
⇪ UPWARDS WHITE ARROW FROM BAR (U+21EA)
↩ LEFTWARDS ARROW WITH HOOK (U+21A9)
⍓ UP ARROWHEAD BETWEEN TWO HORIZONTAL BARS (U+2324)
⌫ ERASE TO THE LEFT (U+232B)
⌦ ERASE TO THE RIGHT (U+2326)
⍉ BROKEN CIRCLE WITH NORTHWEST ARROW (U+238B)
▸ BLACK RIGHT-POINTING SMALL TRIANGLE (U+25B8)
⏏ EJECT SYMBOL (U+23CF)
⍟ APL FUNCTIONAL SYMBOL CIRCLE STILE (U+233D)
⇥ RIGHTWARDS ARROW TO BAR (U+21E5)
⇞ UPWARDS ARROW WITH DOUBLE STROKE (U+21DE)
⇟ DOWNWARDS ARROW WITH DOUBLE STROKE (U+21DF)
↖ NORTH WEST ARROW (U+2196)
↘ SOUTH EAST ARROW (U+2198)
← LEFTWARDS ARROW (U+2190)
→ RIGHTWARDS ARROW (U+2192)
↑ UPWARDS ARROW (U+2191)
↓ DOWNWARDS ARROW (U+2193)
⇤ LEFTWARDS ARROW TO BAR (U+21E4)
⌧ X IN A RECTANGLE BOX (U+2327)
⌅ PROJECTIVE (U+2305)
⇲ SOUTH EAST ARROW TO CORNER (U+21F2)
⇱ NORTH WEST ARROW TO CORNER (U+21F1)
↖ NORTH WEST ARROW TO LONG BAR (U+21B8)
␣ OPEN BOX (U+2423)
⏎ RETURN SYMBOL (U+23CE)
⇤ LEFTWARDS ARROW TO BAR (U+21E4)
❦ FLORAL HEART (U+2766)
❧ ROTATED FLORAL HEART BULLET (U+2767)
☙ REVERSED ROTATED FLORAL HEART BULLET (U+2619)
✓ CHECK MARK (U+2713)
✗ BALLOT X (U+2717)
☑ BALLOT BOX WITH CHECK (U+2611)
☒ BALLOT BOX WITH X (U+2612)
☐ BALLOT BOX (U+2610)
 <Private Use, Last> (U+F8FF)

Conflicts with other apps

Occasionally, some users find a Tinderbox keyboard shortcut will fail to work. This may be because other apps running already have control of that shortcut. A good diagnostic start is to close all other apps and try again, as if the shortcut nov works then the conflict is with one of those apps. Custom shortcuts can get added to the system so check System Preferences ▸ Keyboard ▸ Shortcuts and scan the list for conflicting assignments.

Harder to diagnose are conflicts with utilities that run in the background. If they have any visible presence it is often just an icon in the main menu bar, so easily overlooked. A good check for what background apps might be running is found vi System Preferences ▸ Users & Groups ▸ [your username] in user list ▸ Login Items.

Known conflicts are with some of the default settings for 1Password's 'mini' applet. Because of how the latter operates it appears to trump Tinderbox's shortcuts for Note ▸ Make Web Link. If both apps are installed, depending on the users needs, either edit the shortcuts for 1Password Mini (in 1Password's Preferences) or use Tinderbox's menu items instead of the shortcut.

Tinderbox Application Support folders

When first run, Tinderbox creates a support folder at `~/Library/Application Support/Tinderbox`. This folder can be opened in Finder from within Tinderbox via the Help menu. Since OS 10.8, Apple has hidden the user's Library folde (`~/Library`) in Finder. To make it easier for the general Mac user to access Tinderbox's support folders, the Help menu's 'Reveal Support Folder in Finder' option will open the folder in a new finder window.

The folder contains a number of sub-folders. If, for any reason, a sub-folder is missing it can be created manually in Finder, noting that the names are case-sensitive. The default sub-folders are:

- badges. This is used for custom badge artwork.
- Colour schemes. This is use for storing custom colour scheme files.
- config. This used for storing custom configuration files.
- favorites. This is used for favourite files, or aliases to them.

The following legacy sub-folder may exist on older systems/installs:

- backup. (Now legacy use). This was used pre-v6 for storing automated back-ups.

The following sub-folder can be added manually:

- fill. This is used for custom fill artwork.

Legacy: In v5, deeper nesting of folders is not supported. For instance, all custom badges must be in /badges and not sub folders within that.

- analytics folder
- backups folder
- badges folder
- color schemes folder
- config folder
- favorites folder
- fill folder
- Markdown folder
- prototypes folder
- templates folder

analytics folder

This folder holds files used by Tinderbox in support of Artificial Intelligence (AI) and Neural-Linguistic Programming (NLP) functions.

The files are not designed for access by the user and this folder should treated as a read-only space for the application's use only.

backups folder

This is not used /created by the current app version.

Users who have had/still have installed older versions of Tinderbox may have this support folder:

`~/Library/Application Support/Tinderbox/backups/`

If not using older versions this folder and its contents may be deleted. It is effectively deprecated. The task is taken over by versioning built into the app (and do not overlook time Machine as another form of recovery back-up).

To make it easier for the general Mac user to access Tinderbox's support folders, the Help menu's 'Reveal Support Folder in Finder' option will open the folder in a new finder window.

badges folder

To make it easier for the general Mac user to access Tinderbox's support folders, the Help menu's 'Reveal Support Folder in Finder' option will open the folder in a new finder window.

- Custom Badge artwork

Custom Badge artwork

The internally packaged image files used by the $Badge attribute for icon badges may be supplemented by adding icon files to the folder:

`~/Library/Application Support/Tinderbox/badges`

This folder is automatically created. In older versions, if the /badges/ directory is not automatically created by Tinderbox on installation, so simply create it manually (note that all the folder names in the path are case-sensitive). Tinderbox polls the '/badges/' folder whenever the Badges menu is called so newly added files will be seen immediately.

To separate out separate sets of artwork, e.g. for different projects, collections of badge files can be placed in discrete sub-folders inside the 'badges' folder. In such cases, the folder shows as a new tab on the badge picker pop-over.

Badges are square PNG format files (transparency is supported). The app's default sets use 32 x 32 pixel artwork which is displayed in most views at 16x16px. $BadgeSize can be used to override this.

The name of a custom badge shown in the Badges menu is name of the custom badge icon file minus the extension, and is case-sensitive. So, the custom badge file 'car.png' will have the badge name 'car'. The naming convention of the built-in badge names is single lowercase words. It appears that if setting $Badge via action code the name string is case-insensitive.

If a custom and built-in badge name clash, the first listed (the custom one) will be used.

color schemes folder

This folder is at:

`~/Library/Application Support/Tinderbox/color schemes`

This folder is used to hold custom colour scheme (TBC) files that are then available for use in any Tinderbox file via the Document Settings' Colors tab. The name of the colour scheme will be the name of the TBC file.

If a TBC is added while a document is open, it is necessary to re-open the app in order that the Document Settings' Colors tab lists the new scheme.

To make it easier for the general Mac user to access Tinderbox's support folders, the Help menu's 'Reveal Support Folder in Finder' option will open the folder in a new finder window.

config folder

Used for holding custom config.xml file (the folder is empty by default) or other custom configuration files.

To make it easier for the general Mac user to access Tinderbox's support folders, the Help menu's 'Reveal Support Folder in Finder' option will open the folder in a new finder window.

favorites folder

Changing from earlier use, the 'favorites' folder is used to provide a stationery file (see below) type of service for any file in that folder.

The 'favorites' folder name is case sensitive and is found at

`~/Library/Application Support/Tinderbox/favorites/`

TBX files or aliases of TBX files placed in the folder are automatically added to the File menu's 'Open Favorites…' sub-menu. Items remain listed in the menu until removed from the folder. This ensures such files are always available whether recently used or not.

Aliases. By placing aliases in this folder rather than the TBX file itself, the TBX can be kept in a more obvious location (such as within the `~/Documents/` folder hierarchy). A Mac alias can even track the file if moved around the same disk.

Files opened from this folder are opened as *unsaved* copies of the source file and given the name 'untitled.tbx'. This simulates the old Mac stationery file behaviour.

Unlike in older versions (pre v6?) of Tinderbox, the 'favorites' folder cannot be used to store (or alias) files you wish to edit directly.

To make it easier for the general Mac user to access Tinderbox's support folders, the Help menu's 'Reveal Support Folder in Finder' option will open the folder in a new finder window.

What are stationery files and why the change in Tinderbox behaviour?

The Mac's notion of a 'stationery file' is one that acts like a stationery pad: for a fresh copy you take the top item, i.e. the file acts as a template. Thus files that are set in the OS as stationery files (via a tick box on the their Finder/Get Info dialog), will open an unsaved copy of the file with an 'untitled' filename.

This OS behaviour exists (as at macOS 10.15.1) when manually opening such a file in Finder. However, the API allowing programatic creation of a copy a stationery file has been deprecated and cannot be relied upon its use.

Thus current Tinderbox behaviour is to treat all files in the folder as if they were stationery files *regardless* of the actual OS setting.

So how is a stationery file edited?

IMPORTANT: do **not** use the File ▸ Favorites menu for this task or you will edit a *copy* of the original!

First check whether the TBX is itself flagged to the OS as a stationery file. Longtime users are more likely to have TBX set as stationery file. To check status:

- use the menu Help ▸ Reveal support Folder In Finder.
- in the window that opens select the 'favorites' sub-folder. This should show all the files you normally see in you File ▸ Favorites menu.
- in the finder window select the file you wish to edit and then Get Info (⌘+⌥+I).
  - if the desired file is an alias, first find the original of the alias in its true Finder location
- in the 'General' section of the Finder's Get Info dialog (expand the section if collapsed), there are two tick-boxes, one titled 'Stationery pad'.
- if the box is ticked, you will need to un-tick it before you can edit.

Once assured the original TBX is not set in the OS as a stationery file—something only likely for longer term users—it can be edited:

- open the TBX in situ *directly from Finder*.
- make any edits, save and close.
- open the same file but via the Favorites menu and ensure the new unsaved file created shows the recent edits.

Is it necessary to re-apply the OS stationery flag if it was originally set (above)? For this 'favorite' file use within Tinderbox, no, but the user may have it so set for other reasons so act accordingly.

fill folder

The 'fill' folder name is case sensitive' and it found at:

`~/Library/Application Support/Tinderbox/fill/`

The folder holds any custom fill files files (see below) used for map view backgrounds or map note icons.

The Fill folder and its subfolders may contain aliases of image files as well as the images themselves.

To make it easier for the general Mac user to access Tinderbox's support folders, the Help menu's 'Reveal Support Folder in Finder' option will open the folder in a new Finder window.

- Custom Fill artwork

Custom Fill artwork

The internally packaged image files used for $Fill textures for map icons and map backgrounds ($MapBackgroundFill) may be supplemented by adding RGB colour bitmap image files in JPG (.jpeg, .jpg) or PNG (.png) format to the fills folde `~/Library/Application Support/Tinderbox/fill` (create the folder if it does not already exist). It is not necessary to re-start Tinderbox to use a newly added fill. The User Fill folder may also contain folders of image files, e.g. a folder per project.

Custom fill files will be listed in the Fills pop-up menu.

Custom fills should be referred to within Tinderbox by their filename minus extension (case-sensitive). Thus a custom fill 'Droplets.jpg' would be set by using a $Fill value of "Droplets" (though the actual name string seems to be case insensitive).

Markdown folder

From v9.5.0, support for legacy markdown stylesheets, stored in this application support sub-folder, has been discontinued. Any existing such preferred style notes should be moved to /Hints/Preview/style (if necessary, first add the built-in Hints container).

Legacy info only

The 'Markdown' folder name is case sensitive' and it found at:

```
~/Library/Application Support/Tinderbox/Markdown/
```

The folder holds a file `style.css` which is used to help rendering the in-app preview of Markdown-based notes. The CSS style can be user-edit to customise the otherwise generic HTML styling used in the text Preview mode.

To make it easier for the general Mac user to access Tinderbox's support folders, the Help menu's 'Reveal Support Folder in Finder' option will open the folder in a new finder window.

prototypes folder

This allows for locally shared prototypes are stored as top-level notes in a Tinderbox document named Prototypes.tbx in subfolder "prototypes" of the Tinderbox support folder.

```
~/Library/Application Support/Tinderbox/prototypes/Prototypes.tbx
```

templates folder

This can be used to hold export templates for use with any open Tinderbox document. Shared templates are stored as (plain) text files in this folder, and may be added to any document by selecting the file from File ▸ Built-In Templates.

If a template with that name already exists, the template's text is replaced with the text of the external template file (without any warning).

Tinderbox File Types

Tinderbox program icon

The current Tinderbox application icon was introduced in v9.0.0.

Data (TBX) Files

These are the normal Tinderbox data files in which a document's data is kept.

Filename Extension: '.tbx'

Icon: White page with the flame from the Tinderbox program icon and 'TBX' beneath it.

It is to be presumed the '.TBX' extension will also be used as the program's data filename extension for the Windows version, when it arrives.

Colour Scheme Files

Colour scheme files allow the user to choose a colour scheme from the Document Setting' Colors tab. Colour scheme files also record the text font and map font preferences. Often, choice of font and font size is connected to the choice of colour scheme. Saving additional information helps makes colour schemes more useful.

Filename Extension: '.tbc'

Icon: a white page with a wheel of colour chips and the title 'tbc'. Previously, it was a white page with the old Tinderbox program symbol and colour chips down the left side.

To use/import a colour scheme file simply drag it onto an open Tinderbox document's view pane.

To save a colour scheme, open the Document Inspector's Colors tab.

Colour schemes are saved with the current text background colour; previously, this was omitted. Colour schemes also save and restore the current text colour, and the default value of $Color.

Colour scheme files are normally intended to be dropped into view windows. If a colour scheme file is double-clicked, Tinderbox displays a dialog suggesting that you drop the file in a view window.

Any TBC files correctly placed within the ' color schemes' folder of Tinderbox application support folder is accessible to open Tinderbox documents via the Document Settings Colors tab. Note TBC added there are not shown for open documents. These must be re-opened to refresh the scheme listing.

Another, more pervasive way to set custom schemes is to customise the colors.xml configuration file. In this case these custom colour definitions becomes the default TBX colour scheme for any subsequent new TBX files; existing TBXs are unaffected but could be updated by dragging on a similarly defined colour scheme file.

Creating and saving colour schemes

Note: It might be good to use a new document for this task

Alter the existing document so the colours are as you wish. Then use the Colors Inspector to create a colour scheme file.

Tinderbox Preferences

These are the special XML files used to help with the default configuration of new Tinderbox files.

Filename Extension: n/a

Icon: A switch alongside the old Tinderbox program icon.

You are unlikely to encounter this file type directly unless you are creating packages for deploying example projects.

The normal location for the Preferences file is:

```
~/Library/Preferences/Tinderbox™ Preferences
```

Users wanting to share Preferences between several users of the same Mac may elect to copy/move their Preferences file to:

```
/Library/Preferences/Tinderbox™ Preferences
```

Tinderbox checks the latter folder for a Preferences file and if found, uses it in preference to any local version.

Stamp files

These are small text files containing one or more discrete action code expressions. The filename is used as the stamp name when importing a file and vice versa for exporting from the Stamps Inspector.

Filename Extension: '.tbxstamp'

Icon: an ink stamp with a yellow handle.

Configuration Files

These are the special XML files used to help with the configuration of the Tinderbox program.

Filename Extension: '.xml'

You are unlikely to encounter this file type directly unless you are trying to alter the program's set-up.

As an alternate to the files packaged in the Tinderbox application, user-edited versions may be placed at:

```
~/Library/Application Support/Tinderbox/
```

If the folder does not exist, just create it. If using custom Badges, add a further subfolder to `/Tinderbox/` called `/badges/`.

This mechanism lets the user override the built-in application configuration settings on a per-user basis. If a configuration file is found in the Application Support file, that version is used in preference to the version stored inside the application bundle. It also saves the user having to make edits inside the application bundle and possibly damaging the application installation as a whole.

Remember: if you do create local copies of these files and upgrade the program to a new version you are advised to check for changes to the defaults and revise your user-set versions accordingly.

Tinderbox currently ships with a number of configuration files:

- colors.xml
- config.xml
- html_helpers.xml
- linkTypes.xml
- menus.xml
- stoplist.txt

colors.xml

This file defines/lists the named colours as seen in a default Tinderbox file. Colour values may be defined, it appears, by any of the supported colour declaration methods: (HTML) name, RGB, hex, HSV. Most of the defaults use hex format declarations. The 'screen' name given to the file is for UI use so, 'red' could be defined as a green!

The (inclusion of and) order of colours seen in UI listings is in bit-wise alphabetical order, i.e. all initial uppercase words come first (i.e. 'Yellow' sorts before 'blue'). One exception to this is the programs' Colors menu, where the inclusion/order is set via the menus.xml file.

The application should be re-started after editing these files to ensure changes are detected. At minimum, the document(s) needing to show the change should be re-opened, should an app re-start be inconvenient at that time.

config.xml

By default, there is no config.xml file, but a user may create a one so as to override:

- a small number of configuration settings currently not included in the document's Document Settings as well a limited range. Most settings relate to weblogs (now moribund) and RSS but two are for general use:
  - **RecentFileLimit**. The number of recently used files to list. Default is 7.
  - **UniqueValueLimit**. The maximum allowable number of discrete values for string-based attribute value lists. The current default is 999. If the limit is exceeded the attribute pop-up list is not shown.
- a limited range of options available in Document Settings (the supported items are not documented).

If created, a config.xml file is stored in the the config sub-folder of the Tinderbox Support folder. Tinderbox should be re-started after editing this file to ensure changes are detected.

When Tinderbox reads a custom config.xml from the application support folder, it first initialises the configuration to the built-in config.xml. Formerly, configurations left unspecified in the custom file used undocumented defaults.

Two further elements may be overridden in config.xml. The default font name sets the default $TextFont for newly-created documents, and 1-9 sets the relative $TextFontSize from 1 (tiny) to 9 (huge). For instance:

```
<TextFont>Helvetica Neue</TextFont>
<TextSize>4</TextSize>
```

To support the wider range of font size selection possible in more recent versions (v7+) a new tag is supported to replace TextSize. NewTextSize holds the desired size a point value. Thus, to set a default 20pt size for $Text, add this to config.xml instead of the old TextSize tag:

```
<NewTextSize>20</NewTextSize>
```

The default content config.xml (as in the app package):

```
<config version="1">
 <RecentPostLimit> 25 </RecentPostLimit>
 <RecentFileLimit> 7 </RecentFileLimit>
 <UniqueValueLimit> 999 </UniqueValueLimit>
 <TechnoratiServer>http://rpc.technorati.com/rpc/ping</TechnoratiServer>
 <TechnoratiMethod>weblogUpdates.ping</TechnoratiMethod>
 <WeblogsComServer>http://rpc.weblogs.com/RPC2</WeblogsComServer>
 <WeblogsComMethod>weblogUpdates.ping</WeblogsComMethod>
 <FeedsterServer>http://api.feedster.com/ping</FeedsterServer>
 <FeedsterMethod>weblogUpdates.ping</FeedsterMethod>
 <GoogleServer>http://blogsearch.google.com/ping/RPC2</GoogleServer>
 <GoogleMethod>weblogUpdates.extendedPing</GoogleMethod>
</config>
```

New to v7, the Outline view darker colour option may be overridden at app level by adding this to the config.xml:

```
<DarkenOutlineColors> 0</DarkenOutlineColors>
```

The old `<TextSize>` which used numbered codes for size is superseded by (and trumped by if both are found) `<NewTextSize>` specified in point size, i.e. '16' for 16-point text.

html_helpers.xml

This file is a list of programs that can be configured in HTML Preferences as the associated applications for editing HTML files output by Tinderbox. The file is a list of application names and their macOS 'creator' strings.

Thus, to add a new HTML Editor/Viewer as a 'helper' you need to know its macOS 'creator' string and decide on the text you want for the on-screen name. The files will look something like this (without the XML comment):

```
<helper>
   <!-- a list of helpers -->
   <helper name="Firefox" signature="MOZB" />
</helpers>
```

The second line above defines 'Firefox' as a helper and this is the name you see in pop-up lists on the program. Under the hood, the creator type of 'MOZB' is passed to the macOS. It shows the syntax of the line you should follow when adding new helpers. In the following example, the Mac Browser Camino is added as a new helper, by inserting a new 'helper name' XML entity anywhere inside the tags. The helper apps are listed in HTML preferences the order used in the file:

```
<helpers>
   <helper name="Firefox" signature="MOZB" />
   <helper name="Camino" signature="MOZC" />
</helpers>
```

The order in which the 'helper name' entities are listed in the XML is the order in which the helpers are listed in Tinderbox's helper pop-ups.

The application should be re-started after editing these files to ensure changes are detected.

linkTypes.xml

This file describes/defines the default set of link types for use in Tinderbox files. A colour may also be defined: the default is black (#000000). Visibility of individual types in map views may not be set here but should be done via the Link Ty pane of the Attributes dialog.

Do not forget you can save your customised files outside the application, rather than overwriting the application's default file inside the package.

The application should be re-started after editing these files to ensure changes are detected.

Hierarchy of inheritance

The link types included in a new TBX file are defined by the 'linkTypes.xml' file. This file resides inside the Tinderbox application package (i.e. most users will never see it). To assist the user Tinderbox also checks this location:

```
~/Library/Application Support/Tinderbox/
```

…and if a copy of the file is found there, then it takes precedence over the application's default version of the file. Note: if the folder does not exist, just make one of the appropriate name. As Tinderbox offers is this 'external' location, more safely accessible to the user, you are advised to use the latter for customisation of link types. This avoids the possibility of a mistaken edit inside the application package that might upset Tinderbox as a whole.

File syntax

This is the default set of link types:

```
<linkTypes version="1">
   <link name="*untitled" label="" color="#000000" required="true" />
   <link name="prototype" color="#993333" required="true" />
   <link name="note" color="#666666" required="true" />
   <link name="note+" color="#444444" required="true" />
   <link name="agree" color="#000000" />
   <link name="disagree" color="#000000" />
   <link name="clarify" color="#000000" />
   <
   <
   <link name="exception" color="#000000" />
   <link name="response" color="#000000" />
</linkTypes>
```

The full allowable <link> syntax is this:

```
<linkType name="*untitled" label="" visible="1" showLabel="0" color="#000000" style="0" required="true" />
```

The 'showLabel' and 'style' arguments were added after the original configuration. So, although the application defaults (above) do not use these additions you may safely use them in custom link files.

Note that 'internal' links such as basic links and text links can have 'target', 'title' and 'class' info set via the create/browse links dialogs just you would with an 'external' (web) link. This is pertinent if the data is to be exported as HTML.

Syntax for linkTypes.xml file:

- linkTypes
- link

linkTypes

<linkTypes> … </linkTypes>

Occurs: Once

Multiple instances: No.

Has attributes: No.

Is container: Yes.

Self-closing: No.

This element acts as a wrapper for all file content.

Example:

```
<linkTypes version="1">
   <linkType name="*untitled" label="" visible="1" showLabel="0" color="#000000" style="0" />
  <link name="note" visible="1" showLabel="1" color="#666666" style="0" />
</linkTypes>
```

link

<link [argument]="[value]" />

Occurs: Inside <linkTypes>

Multiple instances: Yes.

Has attributes: Yes.

Is container: No.

Self-closing: Yes.

This element is defined once per link type defined.

Example:

```
<linkType name="*untitled" label="" visible="1" showLabel="0" color="#000000" style="0" />
```

Attributes:

- name
- label
- visible
- showLabel
- color
- style
- required

name

name="[value]"

This is the name used for the link type within Tinderbox and for link captions in Map view. The first listed item is normally "*untitled". A link type should always have a name. The user can toggle the visibility of a link's label via the showLabel attribute. Some actions like linkedTo() and linkedFrom() use the asterisked "*untitled" string to match 'untitled' links.

The name value can be further customised within the program at file level via the Attributes:Link Types dialog.

Example:

```
<
```

label

label="[value]"

This attribute appears optional. If blank, this has the equivalent effect of causing the visible attribute to be set off. This is the mechanism originally used (pre v3.0.5) to make 'untitled' links have no caption on Map view arrows. Note that this attribute can be used to give a different screen caption than the default of the name value, though in general you should not not need to do this as there is no advantage (at present!).

Example:

```
<link name="*untitled" label="" color="#000000" visible="o" style="0" />
<link name="XYZ" label="something else" color="#000000" visible="1" style="0"/>
<link name="agree" color="#000000" visible="1" style="0"/>
```

visible

visible="[value]"

A value of "0" turns the of the display of the arrow and label this link type and a default of "1" sets it on.

The visible value can be further customised within the program at file level via the Attributes:Link Types dialog.

Example:

```
<linkType name="*untitled" label="" visible="1" showLabel="0" color="#000000" style="0" />
```

showLabel

showLabel="[value]"

The user can toggle the default visibility of any link type's label via this setting. A value of "0" turns the label off and a default of "1" sets the label on (i.e. displayed).

Use the visible attribute to hide both label and arrow in Map view.

The visible value can be further customised within the program at file level via the Attributes:Link Types dialog.

Example:

```
<linkType name="*untitled" label="" visible="1" showLabel="0" color="#000000" style="0" />
<linkType name="note" visible="1" showLabel="1" color="#666666" style="0" />
```

color

color="[value]"

This is the colour to be used to draw the arrow (but not label) in Map view. Colour values should be stated in the form of a 6-digit hexadecimal value (i.e. like web colours) preceded by a '#', e.g. #FFCC99.

The colour value can be further customised within the program at file level via the Attributes:Link Types dialog.

Example:

```
<linkType name="note" visible="1" showLabel="1" color="#666666" style="0" />
```

style

style="[value]"

This sets a default visualisation style for this link type (only), overriding the global/file level preference for straight or curved arrows (in Map view). This possible values are:

- 0: normal. Use the program/file level default for arrows in Map view.
- 1: linear. Use straight lines for arrows in Map view.
- 2: curved. Use curved lines for arrows in Map view.
- 3: bold linear. Use bold straight lines for arrows in Map view.
- 4: bold curved. Use bold curved lines for arrows in Map view.

The style value can be further customised per link within the program via the Create (Web) Link or Browse Links dialog.

Example:

```
<linkType name="note" visible="1" showLabel="1" color="#666666" style="2" />
```

required

required="[Boolean value]"

This optional attribute indicates whether this link type may be deleted by users. There are 4 required types:

- *untitled
- prototype
- note
- note+

This list may vary in subsequent versions.

The default value, also the case if this attribute is omitted, is false.

It is to be assumed users can set existing or user-added types as 'required'. Users should not attempt to alter the 'required' state where this has been set by the program defaults (the above list). This does mean that if you use custom version of the file, when upgrading the program, you should review the version of the file in the application bundle for any changes to use of this argument and adapt your custom file accordingly.

Example:

```
<link name="prototype" color="#993333" required="true" />
```

menus.xml

This file lists and sets the colours, from the list defined in colors.xml, to be shown in the Colors menu. The file allows a custom order to be created. For the menu to operate correctly, listed colours must be properly defined.

The application should be re-started after editing these files to ensure changes are detected.

stoplist.txt

The list of common English words used by the Common Words view for exclusions is found in Contents/Resources/config/stoplist.txt within the Tinderbox app package. It can be overridden at app level by a customised user version in:

```
~/Library/Application Support/Tinderbox/stoplist.txt
```

Note this feature is essentially replaced by the use of the stoplist in the Built-in Hints container, if the latter is used.

The file is a plain text file, one word per line, all lower-case. The reason the word `doesn't` seems to appears twice is to allow for straight and typographic ('curly') variants of the apostrophe character. This is the default list:

about
after
also
because
been
corp

could
does
doesn
doesn't
doesn't
each
even
from
have
here
indent
into
it's
it's
just
last
like
many
might
more
most
much
only
other
over
perhaps
says
seem
seems
should
some
still
such
than
that
their
them
then
there
these
they
those
this
though
through
very
well
were
what
when
where
which
while
will
with
would
year
years
your
of
to
in
it
is
be
as
at
so
we
he
by
or
on
do
if
me
my
up
an
go
no
us
am
the
and
for
are
but
not
you
all
any
can
had
her

was
one
our
out
day
get
has
him
his
how
man
new
now
old
see
two
way
who
boy
did
its
let
put
say
she
too
use
don
ve
re
jpg
gif
htm
html

Other Support Files

Eastgate have made a few other assets such as icon artwork accessible for those users who may wish to change them:

- Lock and sticky icon artwork
- RSS Import Templates

Lock and sticky icon artwork

The $Sticky attribute pushpin and $Lock attribute adornment icons are packaged separately in the application bundle, making it easier to customise them. These files are stored at:

```
/Applications/Tinderbox.app/Contents/Resources/elements/
```

The files are of type '.icns', i.e. macOS icon files:

- locked.icns
- sticky.icns
- unlocked.icns
- unsticky.icns

RSS Import Templates

Two RSS templates are included, one for the overall (channel) feed and one for each item. These are for RSS import purposes and ought not to require user edits, but it is understood they are allowed.
Channel template:-

```
^title^
^link
^subtitle
^description
    <-- blank line here
```

Item template:-

```
^title
^link
^description
```

It appears that user created templates, as placed in the user configuration files folder, are not supported. If editing is required, it is thus necessary to edit the actual templates in the application package.
The mark-up is like HTML export codes but syntax is slightly different. Although the built-in examples do not use closing ^ carets, if editing it is suggested you do so.
From the above example, known RSS import compatible codes are:
^title^
^subtitle^
^link^
^description^
Exactly how these codes map to RSS/Atom feed codes is not documented.

Syntax Library

Note: most users are unlikely to need to read this section. Also, an understanding of XML syntax is assumed.

Tinderbox has several features that may be of interest to those trying to produce for and support other Tinderbox users:

- The XML TBX format
- Applescript

The XML TBX format

```
<?xml version="1.0" encoding="UTF-8" ?>
<tinderbox version="2" revision="13" savedBy="9.0.0 b523"  uuid="[UUID#]" >
    <attrib Name="anything" editable="0" visibleInEditor="1" default="" >
        <attrib Name="System" parent="anything" editable="0" visibleInEditor="1" default="" >
            <attrib Name="[Attr Group name]" parent="System" editable="0" visibleInEditor="1" default="" >
                <attrib Name="[screen name]" parent="[Attr Group name]" editable="1" visibleInEditor="1" [optional: canInherit="0"] default="[default value]" ></attrib>
            </attrib>
            </attrib>
        <attrib Name="User" parent="anything" editable="0" visibleInEditor="1" default="" >
            <attrib Name="[screen name]" parent="User" editable="1" visibleInEditor="1" default="[default value]" ></attrib>
        </attrib>
    </attrib>
    <colors >
        <color name="[screen name]" color="[#hexvalue]"  />
    </colors>
    <menu name="Value" kind="stamps" >
        <stamp name="[stamp screen name]" attribute="" >[stamp action code]</stamp>
    </menu>
    <linkTypes >
        <link name="[screen name]" sourceid="[UID #]" sourcecreator="[string]" sstart="[#]" slen="[#]" style="[#]" destid="[UID #]" [optional: dstart="[#]" dlen="[#]"] destcreator="[string]"  />
    </linkTypes>
    <item ID="[UID #]" Creator="[string]" >
        [any nested <item>s]
        <composites>
            <composite nodes="[UID #(s)] " name="[String]" instantiates="" ></composite>
        </composites>
    </item>
    <macros >
        <macro name="[screen name]" >[TB action code]</macro>
    </macros >
    <preferences >
        <[Pref name]>[default value]</[Pref name]>
    </preferences>
    <windows >
        <window ID="[UID #]" Creator="[string]" type="outline" bounds="[4 x #]" x="0" y="0" scale="5" expanded="[space delim UID #s]" collapsed="[space delim UID #s]"  />
            <tabs >
                <tab [per-tab settings] />
            </tabs >
        </window>
    </windows>
    <utilityWindows>
        <window bounds="Rect[ 4 x #]" ID="[UID #]"  />
    </utilityWindows>
    <searches >
        <search>[search string]</search>
    </searches >
    <filters >
        <filter name="[filter-name]" filter="[query]"  />
    </filters >
    <gallery >
    <gallery >
        <tab [per-tab settings] />
    </gallery >
    <badges limit="7" >[list;of;badge;names]</badges>
    <preview>[Base64 image data]</preview>
</tinderbox>
```

This section describes the XML source data in a TBX file.

WARNING. This is not formal vendor documentation of the format. Always work with back-ups. If doing significant interchange to from TBX files via some other workflow you may well need to contact technical support at some point. However, this section should still give you a good overview of the TBX file as used with v6.x of Tinderbox.

Changes to the XML data written occur unannounced (reflecting new/updated features) so this part of aTbRef is likely to lag the XML seen in new TBX.

A TBX file can be opened and read in any competent text.code editing app. The file's data is valid XML in UTF-8 encoding and using Unix-style (line feed character) line endings. Thus much of the data can be read 'by eye' in source from. The only real exception to this is the RTFD versions of notes' text. An un-styled plain text version of the the $Text is stored so can still be read, albeit without stylistic markup.

However, to keep the data file slim, inherited values are not stored for every note. Rather, notes only store intrinsic attributes and those set locally in that note.

This section is structured to read like the TBX data structure with similar nesting of notes. When reading this, it is useful to also open a TBX in your code editor for reference.

Note that in general, tags write their internal tag attributes in the same order in each case (not the case in early versions of the app), though you *may* still discover otherwise, so treat that assumption as tentative.

The top level objects of the XML (TBX) file:

- XML tag
- tinderbox tag
- System Attributes: 'Internal' group

XML tag

The first object is the XML descriptor:

```
<?xml version="1.0" encoding="UTF-8" ?>
```

The xml tag is a standard tag identifying the XML version and encoding type (UTF-8).

The only other root level tag is the tinderbox tag, which encloses the file data.

tinderbox tag

The 'tinderbox' tag wraps the overall data:

```
<tinderbox version="2" revision="15" savedBy="version 9.6.0 b627" uuid="5FD9193F-D2E3-4849-B7DE-C4D759536C01" >
[data]
</tinderbox>
```

version. The (presumed) structural version of the TBX file: a number.

revision. The (presumed) revision version of file **version**: a number

savedBy. The app version number that created or last saved the current file: a version number and more recently a beta/build number as well (here build number 542); an application version number string

uuid. A unique identifying number for this file to assist with external access/identification: an alphanumeric hash string. Be aware that copying/duplicating a TBX in Finder does not cause a new UI to be generated. Use 'Save As' inside Tinderbox instead.

The 'tinderbox' object contains a number of standard child tags:

- attrib.
- colors.
- menu.
- linkTypes.
- item.
- links.
- macros.
- preferences.
- windows.
- utilityWindows.
- searches.
- filters.
- gallery.
- badges.
- preview.

attrib tag

The `<attrib>` tag is a direct child of the `tinderbox` tag.
It holds a set of nested `<attrib>` tags representing the hierarchy of both system and user (if any) attributes.
`<attrib Name="anything" editable="0" visibleInEditor="1" default="" >` [nested `<attrib>` tags or empty] `</attrib>`
The hierarchy nests thus:

- all attributes
  - System Attributes
    - System Attribute Groups (one or more)
      - Individual System attribute (one or more)
  - User attributes
    - Individual User attribute (one or more, when created by user)

The `<attrib>` tag attributes are fully described here.
This has two nested child tags, representing:

- System attributes.
- User attributes.

attrib - system

A child of the main attrib `<attrib>` tag object.
`<attrib Name="System" parent="anything" editable="0" visibleInEditor="1" default="" ></attrib>`
There are 18 groups visible to the user in the UI. There is one totally hidden group, "Internal", containing 4 attributes.
The object contains a list of all the groups (the Internal group is described separately here).

- attrib - [groupname]

attrib - [groupname]

A child of the System `<attrib>` tag object. For example, for the 'Agent' group:
`<attrib Name="Agent" parent="System" editable="0" visibleInEditor="1" type="4" default="false" ></attrib>`
This object contains a list of attributes defined within the group.

- attrib - attribute

attrib - attribute

Each Tinderbox attribute, both internal-only and those exposed to the user, are defined by a single attribute tag, which is a leaf object with no children. All Tinderbox data objects have all attribute's even if they cannot make use of them, e.g. $AgentQuery can be set to a note but does nothing. Attributes are all defined with a standard set of tag attributes. For example:

```
<attrib Name="OutboundLinkCount" parent="General" editable="0" visibleInEditor="1" type="2" canInherit="0" default="0" >
</attrib>
```

Name. The (screen/action code) name of the attribute. "anything " reserved for the root `<attrib>` tag.
parent. All `<attrib>` tags except the root one have a parent value which is the **Name** of its parent `<attrib>` tag.
editable. Number: zero or one. A '0' implies the attribute is calculated and thus read-only so cannot be set via via UI or action code. A '1' implies the user can edit the attribute's value
visibleInEditor. Number: zero or one. Is this attribute visible in the program's UI? A '0' implies no, a '1' implies yes.
kind. A number. Purpose uncertain but appears to indicate the attribute inherits from a preference; the number is not the order of the preference in source. Note: these mappings have varied over time. Attributes using **kind** are:

- 1 $Creator
- 2 $NameFont
- 3 $MapBackgroundColor
- 4 $TitleFont
- 5 not used
- 6 not used
- 7 $TextColor
- 8 $TextBackgroundColor
- 9 not used
- 10 $TextExportTemplate
- 11 $TitleForegroundColor
- 12 $TitleBackgroundColor
- 13 $PrototypeHighlightColor
- 14 $ParagraphSpacing
- 15 not used
- 16 not used
- 17 $TextSidebar
- 18 $TextFont
- 19 $TextFontSize
- 20 $InteriorScale
- 21 [hidden internal system attribute 'WindowPlace']
- 22 $TextAlign
- 23 not used
- 24 $MapBackgroundAccentColor
- 25 $MapBackgroundPattern
- 26 $SmartQuotes
- 27 $NoSpelling

type. A number: the Tinderbox data type. If the value is zero the attribute is not saved, i.e. if this attribute is missing its value must be assumed to be zero; string is this the default type. These numbers are a 0-based count of the ordering seen in the data-type popup list in the 'User' tab of the Document Inspector. Values:

- 0 string (the XML attribute 'type' is omitted if the value is 0, i.e. the default)
- 1 color
- 2 number
- 3 file
- 4 boolean
- 5 date
- 6 unsigned (only used for unseen internal system attribute 'Alias')
- 7 action (action-type attributes are system attributes only)
- 8 set
- 9 url
- 10 list
- 11 font (font-type attributes are system attributes only)
- 12 interval
- 13 dictionary
- 100 ? (used by internal system attribute 'WindowPlace')

canInherit. Number: zero or one. Only present if value is '0'. If missing, assume a value of '1'. Presumably a zero value implies the attribute is intrinsic. Only used for System attributes.

description. *Only used for User attributes (v6+)*. String: default empty. The optional text description about a user attribute. Entered/edited via the Document Inspector's User tab.

default. A specific default, otherwise that data-type's normal default value.

The Internal attribute group

The **Internal** group defines 4 hidden attributes: $Alias, $EntryScript, $ExitScript, $WindowPlace. None are seen in the UI and the user should not attempt to script them via action code. $Alias is only used in item or agent objects that are aliases. Thus in the XML, the presence of the $Alias attribute in an object indicates the object is an alias rather than a normal note. In such contexts, the value of $Alias will be the $ID of its original note.

attrib - user

A child of the main attrib `<attrib>` tag object.

```
<attrib Name="User" parent="anything" editable="0" visibleInEditor="1" default="" ></attrib>
```

Unlike the System attributes, there are no subgroups, just a list of individual user-created attributes. Non-exhaustive test would suggest, the Tinderbox will allow groups within User attributes but that they can only be created in the TBX source and not via the UI. However, if created, they work, except in the Inspector (quickstamp pane).

- attrib - attribute

attrib - attribute

Each Tinderbox attribute, both internal-only and those exposed to the user, are defined by a single attribute tag, which is a leaf object with no children. All Tinderbox data objects have all attribute's even if they cannot make use of them, e.g. $AgentQuery can be set to a note but does nothing. Attributes are all defined with a standard set of tag attributes. For example:

```
<attrib Name="OutboundLinkCount" parent="General" editable="0" visibleInEditor="1" type="2" canInherit="0" default="0" >
</attrib>
```

Name. The (screen/action code) name of the attribute. "anything " reserved for the root `<attrib>` tag.

parent. All `<attrib>` tags except the root one have a parent value which is the **Name** of its parent `<attrib>` tag.

editable. Number: zero or one. A '0' implies the attribute is calculated and thus read-only so cannot be set via via UI or action code. A '1' implies the user can edit the attribute's value

visibleInEditor. Number: zero or one. Is this attribute visible in the program's UI? A '0' implies no, a '1' implies yes.

kind. A number. Purpose uncertain but appears to indicate the attribute inherits from a preference; the number is not the order of the preference in source. Note: these mappings have varied over time. Attributes using **kind** are:

- 1 $Creator
- 2 $NameFont
- 3 $MapBackgroundColor
- 4 $TitleFont
- 5 not used
- 6 not used
- 7 $TextColor
- 8 $TextBackgroundColor
- 9 not used
- 10 $TextExportTemplate
- 11 $TitleForegroundColor
- 12 $TitleBackgroundColor
- 13 $PrototypeHighlightColor
- 14 $ParagraphSpacing
- 15 not used
- 16 not used
- 17 $TextSidebar
- 18 $TextFont
- 19 $TextFontSize
- 20 $InteriorScale
- 21 [hidden internal system attribute 'WindowPlace']
- 22 $TextAlign
- 23 not used
- 24 $MapBackgroundAccentColor
- 25 $MapBackgroundPattern
- 26 $SmartQuotes
- 27 $NoSpelling

type. A number: the Tinderbox data type. If the value is zero the attribute is not saved, i.e. if this attribute is missing its value must be assumed to be zero; string is this the default type. These numbers are a 0-based count of the ordering seen in the data-type popup list in the 'User' tab of the Document Inspector. Values:

- 0 string (the XML attribute 'type' is omitted if the value is 0, i.e. the default)
- 1 color
- 2 number
- 3 file
- 4 boolean
- 5 date
- 6 unsigned (only used for unseen internal system attribute 'Alias')
- 7 action (action-type attributes are system attributes only)
- 8 set
- 9 url
- 10 list
- 11 font (font-type attributes are system attributes only)
- 12 interval
- 13 dictionary
- 100 ? (used by internal system attribute 'WindowPlace')

canInherit. Number: zero or one. Only present if value is '0'. If missing, assume a value of '1'. Presumably a zero value implies the attribute is intrinsic. Only used for System attributes.

description. *Only used for User attributes (v6+)*. String: default empty. The optional text description about a user attribute. Entered/edited via the Document Inspector's User tab.

default. A specific default, otherwise that data-type's normal default value.

The Internal attribute group

The **Internal** group defines 4 hidden attributes: $Alias, $EntryScript, $ExitScript, $WindowPlace. None are seen in the UI and the user should not attempt to script them via action code. $Alias is only used in item or agent objects that are aliases. Thus in the XML, the presence of the $Alias attribute in an object indicates the object is an alias rather than a normal note. In such contexts, the value of $Alias will be the $ID of its original note.

colors tag

The `colors` tag is a direct child of the `tinderbox` tag.

`<colors>`[nested `<color>` tags or empty]`</colors>`

The tag takes no attributes and acts as a container for `color` tags, i.e. list of defined (named) colours for the current TBX. The list always includes a set of built-in colours. The user may delete or edit these from within the Tinderbox UI as wel as save their own new colours.

- color tag

color tag

All defined colours, whether built-in or user-defined have two attributes:

```
<color name="cool gray" color="#607080" />
```

name. A string. The 'name' of the colour: can be anything.

color. The actual colour used. Stored as a hash-prefixed six-digit hex code.

menu tag

The `menu` tag is a direct child of the `tinderbox` tag.

`<menu>`[nested `<stamp>` tags or empty]`</menu>`

The tag takes no attributes and acts as a container for `stamp` tags, if any. By default there are no stamps.

- stamp tag

stamp tag

A stamp's date consists of a screen **name** and a value representing the stamps action code string, XML-encoded. For example:

```
<stamp name="No KAs" attribute="" >$DisplayedAttributes=;</stamp>
```

linkTypes tag

The `linkTypes` tag is a direct child of the `tinderbox` tag.

`<linkTypes>`[nested `<linkType>` tags or empty]`</linkTypes>`

The tag takes no attributes and acts as a container for `linkType` tags.

- linkType tag

linkType tag

A linkType type tag is included for every built-in link type and any user-created ones. If a link type has a link action, then the action's code is stored in an optional child `<onLink>` tag.

For example:

```
<linkType name="*untitled" label="" visible="1" showLabel="0" color="#000000" colorString="#000000" style="0" />
```

name. The name used for the linkType in code and type listings.

label. The name use when a link label is shown in maps.

visible. Number, zero or one. '1' is visible, '0' means the link is not drawn on screen.

showLabel. Number, zero or one. '1' is visible, '0' means the link is not drawn on screen.

color. The colour used to draw link lines. Stored as a hash-prefixed six-digit hex code.

colorString. A text string, which must be for a defined named `color`. Otherwise the value of 'color' (above) is used.

style. A number. Some form of hash of the state of all the various line style options (not documented).

Child tags:

- onLink tag

onLink tag

If a link type has a link action in it stored in this child tag of the Code><link> tag. For example, here the link type "custom-made" has an action:

```
<linkType name="custom-made" visible="1" showLabel="1" required="0" colorString="#000000" style="0" arrowType="0" >
<onLink >
$Color(destination)="bright green";</onLink>
</linkType>
```

(root) item tag

The `item` tag is a direct child of the `tinderbox` tag.

The top level 'item' tag represents the whole of the user data in the TBX's visible Outline. This 'root' tag has a $Name value of the TBX filename (without the extension). Apart from being the background to root-level major views and thus possibly customised for attributes like $MapBackgroundColour this tag does not behave as source of a normal note. It has no text window.

`<item>`[nested tags or empty]`</item>`

The tag's own attributes and text tag types are listed in the section below, under the item tag. Links—basic, text and web—are stored separately in the `<links>` section. In this part of aTbRef an item object refers to item-scoped data which may relate to a note item, an agent, an adornment or an alias.

Nested in the tag are tags of several types:

- item tag
- adornment tag
- agent tag
- item (alias)

item tag

The item tag is used to contain a single note (or similar object) of data:

`<item ID="3444070141" Creator="John Doe">`[nested outline objects or current item attribute tags] `</item>`

The item tag has two attributes:

- ID. The $ID of the object.
- Creator. The user name set in preferences (and as passed into $Creator).
- proto. Optional: only set if the note uses a prototype. If it does, the prototype's name (i.e. $Prototype value) is stored here, and **not** as a child attribute tag.

Example:

```
<item ID="3444070141" Creator="John Doe" proto="Person" ></item>
```

If the note has $Text is is stored in discrete nested tags in both plain text and encoded RTFD form.

Besides nested outline objects, the item may enclose the following per-item attribute-related tags:

- attribute tag
- composites tag
- text tag
- rtfd tag

Attributes (default and locally-valued) and composites (if any) precede the two $Text-related tags (if any text). Thus nested objects are the last elements in the `item` element before its closing tag.

Attributes, other than the defaults listed below are only written to file if they hold locally set values. Attributes that use preference-default or prototype-inherited values are not stored as these values are calculated on the fly during use. For an item, the following default (intrinsic) attributes are always stored:

- Name
- Created
- Modified
- Xpos
- Ypos

attribute tag

For any locally-set attribute (i.e. not inherited), any item object stores the value in individual attribute tag(s):

```
<attribute name="Created" >2013-02-18T22:09:00+00:00</attribute>
<attribute name="Modified" >2013-02-18T22:09:00+00:00</attribute>
<attribute name="Name" >A grandchild</attribute>
<attribute name="ReadCount" >61</attribute>
<attribute name="SelectionCount" >45</attribute>
```

The example above shows data of several attribute data-types as stored in TBX form. The sole exception to this method of storage is $Text, which is handled in a discrete method of its own.

$Prototype is an exception and if set is not stored here. Instead it is stored in the optional 'proto' tag attribute in the note's `<item>` tag.

An `<attribute >` tag has one internal attribute: name. This is the 'Name' value defined in an ' attrib' tag elsewhere in the document.

The value of the attribute is stored as nested content inside the attribute opening/closing tags.

composites tag

The `composite` tag is a direct child of an `item` tag.

`<composites>`[nested `<composite>` tags or empty]`</composites>`

The tag takes no attributes and acts as a container for tags, if any.

By default there are no composites in a document. These container elements are created/removed as needed.

- composite tag

composite tag

The composite tag is used to contain a single composite object:

```
<composite nodes="1481542875 1481542879 " name="Things" instantiates="" >
```

The tag has three attributes:

- nodes. This is a space delimited list of the IDs of the notes making up the composite.
- name. The (optional) screen name of the composite.
- instantiates. Purpose is unknown. An empty string. If manipulating TBX XML directly, still include this attribute.

Example

```
<composite nodes="1481542875 1481542879 " name="Things" instantiates="" >
```

There is no apparent different between 'master' composites and those formed elsewhere. It appears this it is the fact of being stored in the `/Composites` container is what makes them available for replication.

text tag

If $Text is populated, this tag stores the the plain text (0only) of the overall styled text.

Embedded images in $Text are stored only in the <rtfd> tag's content—image data is **not** included in this tag.

Example:

```
<text >Hello world.</text>
<rtfd >cnRmZAAAAAADAAAAAgAAAAcAAABUWFQucnRmAQAAAC7hAAAAKwAAAAEAAADZAAAAe1xydGYxXGFu
c2lcYW5zaWNwZzEyNTJcY29jb2FydGYxMzQ3XGNvY29hc3VicnRmNTcwCntcZm9udHRibFxmMFxm
bmlsXGZjaGFyc2V0MCBIZWx2ZXRpY2FOZXVlO30Ke1xjb2xvcnRibDtccmVkMjU1XGdyZWVuMjU1
XGJsdWUyNTU7XHJlZDBcZ3JlZW4wXGJsdWUwO30KXHBhcmRcdHgzNjBcc2I4MFxzYTgwXHBhcmRp
cm5hdHVyYWwKClxmMFxmczMyIFxjZjIgSGVsbG8gd29ybGQufQEAAAAjAAAAAQAAAAcAAABUWFQu
cnRmEAAAAIhdgFW2AQAAAAAAAAAAAAA=
</rtfd>
```

Note that the plain text and RTFD versions are always both saved if the note has any $Text.

rtfd tag

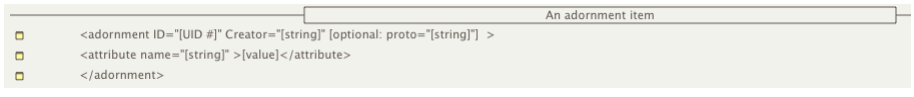
If text has been set via a rich text view, i.e. the text window, the text tag is also stored in RTFD form, even if there is no apparent styling. This example shows a single paragraph of unstyled text "Hello world."(below its plain text stored versi

```
<text >Hello world.</text>
<rtfd >cnRmZAAAAAADAAAAAgAAAAcAAABUWFQucnRmAQAAAC7hAAAAKwAAAAEAAADZAAAAe1xydGYxXGFu
c2lcYW5zaWNwZzEyNTJcY29jb2FydGYxMzQ3XGNvY29hc3VicnRmNTcwCntcZm9udHRibFxmMFxm
bmlsXGZjaGFyc2V0MCBIZWx2ZXRpY2FOZXVlO30Ke1xjb2xvcnRibDtccmVkMjU1XGdyZWVuMjU1
XGJsdWUyNTU7XHJlZDBcZ3JlZW4wXGJsdWUwO30KXHBhcmRcdHgzNjBcc2I4MFxzYTgwXHBhcmRp
cm5hdHVyYWwKClxmMFxmczMyIFxjZjIgSGVsbG8gd29ybGQufQEAAAAjAAAAAQAAAAcAAABUWFQu
cnRmEAAAAIhdgFW2AQAAAAAAAAAAAAA=
</rtfd>
```

Any inline images are included in the RTFD stream, as base64 encoded data, where they occur in the run of text.

The `<rtfd>` tag is always deleted if its associated `<text>` tag is deleted.

adornment tag



The adornment tag is used to contain a single adornment object:

```
<adornment ID="3444053947" Creator="John Doe">[current item attribute tags]</adornment>
```

The adornment tag has two attributes:

- ID. The $ID of the object.
- Creator. The user name set in preferences (and as passed into $Creator).
- proto. Optional: only set if the note uses a prototype. If it does, the value (the prototype's name) is stored here, and **not** as a child attribute tag.

Example:
`<adornment ID="3444053947" Creator="John Doe" >…</adornment>`
Unlike items or agent tags, adornments never nest other outline objects within themselves.
See the item tag description for the general range of attributes used.
Note that adornments cannot display $Text, although they may store a text; the $Text can be read via the text pane as with a normal note.
Attributes, other than the defaults listed below are only written to file if they hold locally set values. These differ from those of the item tag:

- Name
- Created
- Modified
- Xpos
- Ypos
- Height
- Width
- Searchable

Other elements. Besides the and elements, if $Text is used, there is one other optional element included if the adornment is an image adornment:

- image tag

image tag

The `...` element is included in adornments that are image adornments.
The contents of this element is the image data stored as a JPEG, in Base 64 form (regardless of original source format).

agent tag

An agent item

```
<agent ID="[UID #]" Creator="[string]" [optional: proto="[string]"] >
<attribute name="Agent" >true</attribute>
<attribute name="AgentQuery" >[value]</attribute>
<attribute name="HTMLExportChildren" >false</attribute>
<attribute name="IsPrototype" >false</attribute>
<attribute name="[string]" >[value]</attribute>
</agent>
```

The adornment tag is used to contain a single adornment object:
`<agent ID="3444053947" Creator="John Doe">`[current item attribute tags]`</agent>`
The item tag has two attributes:

- ID. The $ID of the object.
- Creator. The user name set in preferences (and as passed into $Creator).
- proto. Optional: only set if the note uses a prototype. If it does, the value (the prototype's name) is stored here, and **not** as a child attribute tag.

Example:
`<agent ID="3444053464" Creator="john Doe" >…</agent>`
Unlike items or agent only nest alias objects within themselves, those of the notes matching their query.
See the item tag description for the general range of attributes used.
Agents use one special (internal, hidden) attribute: Agent. An agent will always include these in its default listing, over and above normal item attributes:
`<attribute name="Agent" >true</attribute>`
`<attribute name="AgentQuery" >[value]</attribute>`
`<attribute name="HTMLExportChildren >false</attribute>`
`<attribute name="IsPrototype" >false</attribute>`
Attributes, other than the defaults listed below are only written to file if they hold locally set values. These differ from those of the item tag:

- Name
- Created
- Modified
- Xpos
- Ypos
- Agent
- AgentQuery (even if empty)
- HTMLExportChildren (uses non-default value: false)
- IsPrototype (uses default: `false`, presumably to override possibly inheriting `true`)

item (alias)

An Alias item

```
<item ID="[UID #]" Creator="[string]" >
<attribute name="Alias" >[original's UID #]</attribute>
<attribute name="Name" ></attribute>
<attribute name="[string]" >[value]</attribute>
</item>
```

Alaises are stored as a normal item tag:
`<item ID="3444054848" Creator="">`[current item attribute tags]`</item>`
The item tag has two attributes:

- ID. The $ID of the object.
- Creator. As $Creator is intrinsic, this is left blank so as not to conflict with the value of the original).

Example:
`<item ID="3444054848" Creator="" >…</item>`
However, unlike normal item or agent tags, aliases never nest other outline objects within themselves. Aliases use, but do not store, their original's prototype.
Alias status is indicated by the presence of one special (internal, hidden) attribute: Alias. An alias will always include these in its default listing, over and above normal item attributes:
`<attribute name="Alias" >3444070141</attribute>`
`<attribute name="Name" ></attribute>`
The 'alias' value is the $ID of the original of the alias. The $Name is left blank as this is intrinsic. Instead, the alias inherits the original's name.
See the item tag description for the general range of attributes used.
Attributes, other than the defaults listed below are only written to file if they hold locally set values, which in the case of aliases will only ever be intrinsic attributes. For an alias item, the following default (intrinsic) attributes are always stored:

- Alias
- Name (set to [no value])
- Created
- Modified
- Xpos
- Ypos

links tag

The `windows` tag is a direct child of the `tinderbox` tag.
`<links>`[nested `<link>` tags or empty]`</links>`
The tag takes no attributes and acts as a container for `link` tags, if any. By default there are no links.

- link tag

link tag

Each link belonging to a note (basic, text or web) is started as discrete 'link' tag. Note that aliases may have their own (intrinsic) basic links, but share text/web links (web links can only be created from $Text unless differently stored as URL-type attributes). Links use a subset of possible tag attributes

A basic link to another note (with a comment):

```
<link name="prototype" sourceid="3176208968" sourcecreator="John Doe" sstart="-1" slen="0" style="0" arrowtype="-1" labelx="0" labely="0" linkWidth="1" destid="3197542267"
destcreator="John Doe" color="normal" destDoc="A35EDCF0-84A5-4C10-9FEC-15D289DA7B15" sourceDoc="" comment="Hello World!" />
```

A text link to a text target:

```
<link name="clarify" sourceid="3175851881" sourcecreator="John Doe" sstart="220" slen="13" dstart="0" dlen="12" style="0" arrowtype="-1" labelx="0" labely="0" linkWidth="1"
destid="3175179052" destcreator="John Doe" color="normal" destDoc="A35EDCF0-84A5-4C10-9FEC-15D289DA7B15" sourceDoc="" />
```

A web link from $Text:

```
<link name="web reference" sourceid="3197539691" sourcecreator="John Doe" sstart="110" slen="7" style="0" arrowtype="-1" labelx="0" labely="0" linkWidth="1" destid="3162983401"
destcreator="John Doe" color="normal" destDoc="A35EDCF0-84A5-4C10-9FEC-15D289DA7B15" sourceDoc=""URL="http://c-command.com/dropdmg/" target="new" />
```

Below are attributes in normal order of occurrence:

name. The linkType name.

sourceid. $ID of the source item.

sourcecreator. $Creator for the source object.

sstart. The zero-based character position of the start of the text anchor in the plain text $Text record, i.e. value of `0` implies the link anchor starts from before character #1 in the $Text. A value of `-1` implies this link has no anchor, such as w basic links.

slen. The length (number of characters) in the link anchor counting from the start **sstart** offset within the source $Text. It is always `0` for basic links.

dstart. [*Optional but included if empty*]. Only used if there is a target anchor. The zero-based character position of the start of the destination note's text anchor in the plain text $Text record, i.e. value of `0` implies the link anchor starts from before character #1 in the $Text.

dlen. [*Optional but included if empty*]. Only used if there is a target anchor. The length (number of characters) in the link anchor counting from the start **dStart** offset within the destination $Text.

style. This is a binary value representing the various options for line style (dot, dash, broad, etc.). By observation, always `0`, zero. Values are additive for options ticked: bold = `128`, linear = `64`, dashed = `16`, dotted = `8`, broad = `256`. Thus dashed+broad = `272`.

arrowtype. Type of arrowhead. Default is `-1` and is the default arrow style. Circle is `1` and Arrow, if manually set, is `0`.

labelx and **labely**. Default is `0`, zero. If the link label is dragged from its auto-placed position. This records the X/Y offsets although the derivation of the values is undocumented. Although not always altered these attributes are always include in a link.

sourcepad and **destpad**. [*Optional but included if empty*]. Only included when either/both source or destination ends of the link are dragged from their auto-positioned default. Top = `2`, right = `4`, bottom = `6`, left = `8`.

linkWidth. Always present. Value is always `1`. Purpose is unknown, possibly a legacy compatibility setting.

destid. $ID of the destination object

destcreator. $Creator for the destination object.

color. The colour of the link. Inherited from the link type. Default value is `normal`.

destDoc. (v9.6.0+) The destination TBX of the link. For in-document links this ID the same as the host document's UUID in the tag. If the link is an inter-document link this holds the UUID of destination document (the link' **destID** will also be a note ID in the target document).

sourceDoc. (v9.6.0+) Purpose not yet clear (currently unused?).

URL. [*Only stored for web links **and** if populated*]. The target URL, and the HTML `href` tag attribute value to use on export.

class. [*Only stored for web links **and** if populated*]. The HTML `class` tag attribute value to use on export.

target. [*Only stored for web links **and** if populated*]. The HTML `target` tag attribute value to use on export.

title. [*Only stored for web links **and** if populated*]. The HTML `title` tag attribute value to use on export.

comment. [*Only stored if populated*]. Per link, ad hoc, comment.

macros tag

The `windows` tag is a direct child of the `tinderbox` tag.

`<macros>`[nested `<macro>` tags or empty]`</macros>`

The tag takes no attributes and acts as a container for `macro` tags, if any. By default there are no macros.

- macro tag

macro tag

By default there are no macros. Each macro is a separate tag.

```
<macro name="Codify" ><code>$1</code></macro>
```

name. The screen name of the macro.

[**value**]. The macro's action code, XML-encoded.

preferences tag

The `windows` tag is a direct child of the `tinderbox` tag.

`<preferences>`[nested preference-name tags or empty]`</preferences>`

The tag takes no attributes and acts as a container for individual preference values.

- [preference name] tag

[preference name] tag

Each preference's name is used as a tag enclosing its value:

```
<ParagraphSpacing > 2
```

In some cases the data is multi-line:

```
<RecentFiles > <files  > <file>Macintosh HD:Users:mwra:TBX:aTbRef24</file> </files></RecentFiles>
```

Preference 'Fontmap' data is also stored in multi-line form.

Colours are stored using 16-bit values (0-65535).

Some tags will be retained for compatibility with older versions of Tinderbox.

windows tag

The `windows` tag is a direct child of the `tinderbox` tag.

`<windows>`[nested one or more `<window>` tags]`</windows>`

The tag takes no attributes and acts as a container for `window` tags, of which there should always be at least one. The tag covers document windows. Standalone text windows are stored under the utilityWindows tag. Non-note secondary windows are not fully recorded as these close at session end, though the last used Inspector and Find window X/Y locations are stored under the preferences tag.

The individual open document windows are listed in front-to-back order, i.e. the first listed is the frontmost in the UI.

- window tag

window tag

The `window` tag is direct child of the `windows` tag. It stores all the tabs currently defined for that window:

Example:

```
< bounds="Rect[ 503 243 1403 843]" ruler="0" toolbar="0" >
<tabs>[one or more <tab> tags]</tabs>
</window>
```

bounds. A space delimited array of 4 numbers being the screen pixel {x,y} co-ordinates of the top-left and bottom-right corners of the window.

ruler. Show/hide state of the ruler in the text pane's $Text area. A value '0' (default) means hidden, '1' means shown.

toolbar. Show/hide state of the toolbar at the top of the window. A value '0' (default) means hidden, '1' means shown.

Note that unlike for utility windows, no ID argument is needed.

- tabs tag

tabs tag

The `tabs` tag is a direct child of the `window` tag.

`<tabs>`[nested one or more `<tab>` tags]`</tabs>`

The tag takes no attributes and acts as a container for `tab` tags, of which there should always be at least one (by default new windows have 2 tabs). The tag holds a set of tags representing each of the tabs in the parent window.

- tab tag

tab tag

The tab tag is used to define a tab used within a (document) window.

Note: this file format is not formally published and the tag attributes are subject to ongoing change. The code sample is taken from an actual tab's data but the order of the attributes cannot be assumed to be constant. As such this does not constitute an official specification of the tag.

```
<tab selected="0" name="" subtitle="" viewType="0" usingRoadmap="0" hideBreadcrumbs="0" scale="32" ID="1639525654" scrollX="448" scrollY="398" selection="1639406348 " expanded=""
textTab="0" splitRatio="0.5" xattribute="Name" xattribute2="Height" xdisplayattribute="DisplayName" xbarattribute="OutlineOrder" xcontainer="" xquery="" xqueryLabel="" xbins="5"
xunlimited="0" xbins2="5" xunlimited2="0" xheatmap="0" xReportStyle="0" spacing="0.35" aspect="2" hyperbolicScale="1" path="" focus="1639510536" crosslinks="0" spread="0.5"
hasBorder="0" isVertical="0" isCentered="1" lineType="0" itemWidth="220" xspace="10" yspace="10" showColumns="0" usingCheckboxes="0"
columnString="_(218);MyNumber(100;2);MyDate(100;l);" treemapExpression="" treemapColorExpression="" treemapBorderColorExpression="$Color" treemapColorStart="#fffef9100"
treemapColorEnd="#ccd5bf100" treemapDepth="0" attribute="" container="" bins="10" sortAttribute="" sortDirections="0" summaryAttribute="" summaryMethod="" query="" queryLabel=""
action="" actionLabel="" fontSize="12" showFilter="0" filterString="" filterName="" />
```

selected. Selection state of the tab. A value of 1 indicated the tab is selected. All other tabs have a value of 0.

name. An optional title string (set/edited via the tab gallery). Default is "", in which case the normal tab label of '[view type]: [root container name]' is used.

subtitle. An optional descriptive string that is set using the tab gallery feature, and can only be seen in the Gallery listing.

viewType. The view pane's current view type:

- 0 = map
- 1 = outline
- 2 = chart
- 3 = attribute browser
- 4 = timeline
- 5 = treemap
- 6 = hyperbolic
- 7 = crosstabs

usingRoadmap. Show/hide state of the text pane links panel. Default value is "0" (hidden)

hideBreadcrumbs. View state of the view pane breadcrumb bar. Values are 0 (shown - default) or 1(hidden).

scale. View pane's zoom state. Default value is 32. Zoomed in (larger text) views have higher values, zoomed out views a smaller value.

ID. The note $ID of the root item for the view.

scrollX. Horizontal scroll position of the view, when last saved. Default is 0. (units of value unknown - pixels?).

scrollY. Vertical scroll position of the view, when last saved. Default is 0 (units of value unknown - pixels).

selection. The $ID number(s) of the view pane's selected item(s) as a space-delimited list.

expanded. Used by Outline *and* Chart. A space-delimited list of $IDs in scope in the view that are expanded.

textTab. The currently selected tab in the text pane. The default is 0 (text). Other values 1 (HTML) and 2 (preview).

splitRatio. The current split ratio of the view and text panes. Default is 0.5, each pane using 50% of the window. A value of 1.0 equates to all text pane, a value of 0 to all View pane.

xattribute, **xattribute2**, **xdisplayattribute**, **xbarattribute**, **xcontainer**, **xquery**, **xqueryLabel**, **xbins**, **xunlimited**, **xbins2**, **xunlimited2**, **xheatmap**, **xReportStyle**. Data relating specifically to the Crosstabs view, see detail:

spacing, **aspect**, **hyperbolicScale**, **path**, **focus**, **crosslinks**, **spread**. Data relating specifically to Hyperbolic view, see detail.

hasBorder, **isVertical**, **isCentered**, **lineType**, **itemWidth**, **xspace**, **yspace**. Data relating specifically to Chart view, see detail.

showColumns, **usingCheckboxes**, **columnString**, **showFilter**. Data relating specifically to Outline view, see detail.

treemapExpression, **treemapColorExpression**, **treemapBorderColorExpression**, **treemapColorStart**, **treemapColorEnd**, **treemapDepth**. Data relating specifically to Treemap view, see detail.

attribute, **container**, **bins**, **sortAttribute**, **sortDirections**, **summaryAttribute**, **summaryMethod**, **query**, **queryLabel**, **action**, **actionLabel**. Data relating specifically to Attribute Browser view, see detail.

The difference in the degree of differing view types' use of XML data reflects how some already store much or all of their view configuration data in attribute values.

Attribute Browser view-specifc data

The following `<tab>` tag attributes refer specifically to Attribute Browser view:

- **attribute**. Selected attribute for review.
- **container**. Path value for the optional scoping container. Default value "".
- **bins**. Number of bins to use for numerical values. Default is 10.
- **sortAttribute**. Selected attribute for sorting the view.
- **sortDirections**. Toggle sort direction. Default is ascending (0)
- **summaryAttribute**. Attribute used to summarise each value.
- **summaryMethod**. Text value of the selected summary type.
- **query**. Optional query code applied by the query. Default value "".
- **queryLabel**. Text label for the query. Default value "".
- **action**. Optional action code applied by the query. Default value "".
- **actionLabel**. Text label for the action. Default value "".
- **fontSize**. View's font size. Default is 12 (pt).

Note that Attribute Browser view shares the Outline's **columnString** attribute, changing the columns in either alters the shared setting at tab level. Thus switching the tab's displayed view type between Outline and Attribute Browser view will retain strings. Outline view (only) can show/hide the columns; unless all extra columns are deleted the tab stores the most recently used settings.

Chart view-specifc data

The following `<tab>` tag attributes refer specifically to Chart view:

- **hasBorder**. Bordered item boolean option. Default is un-ticked/no border (0), 1 is ticked/show border.
- **isVertical**. Vertical layout boolean option. Default is off/horizontal layout (0), 1 is on/vertical.
- **isCentered**. Boolean option for centred positioning on parent item. Default is ticked (1), 0 is off.
- **lineType**. 'Connecting Lines' style. Default is 0 (Orthogonal), also 1 (Straight) or 2 (Curved).
- **itemWidth**. Normal item width. Default value is 220.
- **xspace**. Horizontal spacing value. Default is 10.
- **yspace**. Vertical spacing value. Default is 10.

Crosstabs view-specifc data

The following `<tab>` tag attributes refer specifically to Crosstabs view:

- **xattribute**. Row selected attribute [attribute name].
- **xattribute2**. Column selected attribute [attribute name].
- **xdisplayattribute**. Per-cell label [attribute name].
- **xbarattribute**. Attribute used for bar value within cells [attribute name].
- **xcontainer**. Source container of view [path].
- **xquery**. Optional agent query. Holds query sting, if any.
- **xqueryLabel**. Label string for optional query.
- **xbins**. Rows: maximum number of bins to use. Default is 5.
- **xunlimited**. Rows: boolean to have a bin per value. Default (0) if not enabled, else 1.
- **xbins2**. Columns, maximum number of bins to use. Default is 5.
- **xunlimited2**. Columns, boolean to have a bin per value. Default (0) if not enabled, else 1.
- **xheatmap**. Heatmap show/hide state. Off is 0 (default), 1 is on.
- **xReportStyle**. Not currently used. Default is 0.

Hyperbolic view-specifc data

The following `<tab>` tag attributes refer specifically to Hyperbolic view:

- **spacing**. Item spacing. Default is 0.35.
- **aspect**. Aspect ratio. Default value is 2.
- **hyperbolicScale**. Scale. Default value is 1.
- **path**. Optional highlighted (link type) path. Default is an empty string.
- **focus**. ID of note with current central focus.
- **crosslinks**. Optional cross-links filter. Default is 0 (off), 1 is on.
- **spread**. Spread factor. Default is 0.5.

Outline view-specifc data

The following `<tab>` tag attributes refer specifically to Outline view:

- **showColumns**. Column view state. Default is 0 (no columns), also 1 (columns shown). *Note: this value appears to be discrete to that of Attribute Browser use of column view* .
- **usingCheckboxes**. Checkbox state. Default is 0 (no checkboxes), also 1 (checkboxes shown).
- **columnString**. A string list of displayed column name/width/format. Default is an empty string "". Example: "_(218);MyNumber(100;2);MyDate(100;l);" The three possible parts are:
  - Column name, an attribute title. In this example "_' is column #1 which is always the $Name but which is never titled.
  - Column width (pixels). Stored in parentheses. Default is 100 (pixels?)
  - Optional formatting data string. This is omitted where not pertinent. Where used it is placed after a semicolon after the column width inside the latter's parentheses.
- **showFilter**. Show/hide Outline filter bar. Default is 0 (hidden).

Note that Attribute Browser view shares the Outline's **columnString** attribute, changing the columns in either alters the shared setting at tab level. Thus switching the tab's displayed view type between Outline and Attribute Browser view will retain strings. Outline view (only) can show/hide the columns; unless all extra columns are deleted the tab stores the most recently used settings.

Treemap view-specifc data

The following `<tab>` tag attributes refer specifically to Treemap view:

- **treemapExpression**. Expression used for mapping item area. Default value is "".
- **treemapColorExpression**. Expression used for item fill colour value. Default value is "".
- **treemapBorderColorExpression**. Expression used for item border colour value. Default value is "$Color".
- **treemapColorStart**. Shading start colour. Default value: #fffef9100".
- **treemapColorEnd**. Shading end colour. Default value: "#ccd5bf100".
- **treemapDepth**. Number of descendant levels to display. Default (0) sets no limit.

utilityWindows tag

The `utilityWindows` tag is a direct child of the `tinderbox` tag.

`<utilityWindows>`[nested `<window>` tags or empty]`</utilityWindows>`

This tag holds a listing of stand-alone text windows, using the window tag.

- window tag

window tag

This `window` tag is direct child of the `utilityWindows` tag. The windows are those of stand-alone note text spaces. The individual windows are listed in front-to-back order, i.e. the first listed is the frontmost in the UI. Non-note secondary windows are not recorded as these close at session end.

Stand-alone text windows have no tabs so there are no child tags for these window tags.

Example:

```
<window bounds="Rect[ 184 512 658 866]" ID="3471500281" />
```

bounds. A space-delimited array of 4 numbers being the screen pixel {x,y} co-ordinates of the top-left and bottom-right corners of the window.

ID. The $ID of the source item for the view.

searches tag

The `searches` tag is a direct child of the `tinderbox` tag.

`<searches>`[list of `<search>` tags or empty]`</searches>`

This is a listing of search strings used recent Find queries used in the View pane. Up to 10 recent values are saved using search tags.

- search tag

search tag

This tag stores the string value of an individual recent search term in the View pane's Find toolbar.

`<search>Task</search>`

filters tag

The `filters` tag is a direct child of the `tinderbox` tag.

`<filters>`[list of `<filter>` tags or empty]`</filters>`

This is a listing of any saved filter queries used in the (Outline) View pane. Up to 10 recent values are saved using filter tags.

- filter tag

filter tag

This tag stores the string value of an individual recent search term in the View pane's Find toolbar.

`<filter name="Car" filter="$Name.contains("car")" />`

The filter holds the screen name of filter and the filter's query string.

gallery tag

The `gallery` tag is a direct child of the `tinderbox` tag. Essentially, this stores any tabs saved in the document's saved tabs gallery.

`<gallery>`[list of `<tab>` tags or empty]`</gallery>`

- tab tag

tab tag

The tab tag is used to define a tab used within a (document) window.

Note: this file format is not formally published and the tag attributes are subject to ongoing change. The code sample is taken from an actual tab's data but the order of the attributes cannot be assumed to be constant. As such this does not constitute an official specification of the tag.

```
<tab selected="0" name="" subtitle="" viewType="0" usingRoadmap="0" hideBreadcrumbs="0" scale="32" ID="1639525654" scrollX="448" scrollY="398" selection="1639406348 " expanded="" textTab="0" splitRatio="0.5" xattribute="Name" xattribute2="Height" xdisplayattribute="DisplayName" xbarattribute="OutlineOrder" xcontainer="" xquery="" xqueryLabel="" xbins="5" xunlimited="0" xbins2="5" xunlimited2="0" xheatmap="0" xReportStyle="0" spacing="0.35" aspect="2" hyperbolicScale="1" path="" focus="1639510536" crosslinks="0" spread="0.5" hasBorder="0" isVertical="0" isCentered="1" lineType="0" itemWidth="220" xspace="10" yspace="10" showColumns="0" usingCheckboxes="0" columnString="_(218);MyNumber(100;2);MyDate(100;l);" treemapExpression="" treemapColorExpression="" treemapBorderColorExpression="$Color" treemapColorStart="#fffef9100" treemapColorEnd="#ccd5bf100" treemapDepth="0" attribute="" container="" bins="10" sortAttribute="" sortDirections="0" summaryAttribute="" summaryMethod="" query="" queryLabel="" action="" actionLabel="" fontSize="12" showFilter="0" filterString="" filterName="" />
```

selected. Selection state of the tab. A value of 1 indicated the tab is selected. All other tabs have a value of 0.

name. An optional title string (set/edited via the tab gallery). Default is "", in which case the normal tab label of '[view type]: [root container name]' is used.

subtitle. An optional descriptive string that is set using the tab gallery feature, and can only be seen in the Gallery listing.

viewType. The view pane's current view type:

- 0 = map
- 1 = outline
- 2 = chart
- 3 = attribute browser
- 4 = timeline
- 5 = treemap
- 6 = hyperbolic
- 7 = crosstabs

usingRoadmap. Show/hide state of the text pane links panel. Default value is "0" (hidden)

hideBreadcrumbs. View state of the view pane breadcrumb bar. Values are 0 (shown - default) or 1(hidden).

scale. View pane's zoom state. Default value is 32. Zoomed in (larger text) views have higher values, zoomed out views a smaller value.

ID. The note $ID of the root item for the view.

scrollX. Horizontal scroll position of the view, when last saved. Default is 0. (units of value unknown - pixels?).

scrollY. Vertical scroll position of the view, when last saved. Default is 0 (units of value unknown - pixels).

selection. The $ID number(s) of the view pane's selected item(s) as a space-delimited list.

expanded. Used by Outline *and* Chart. A space-delimited list of $IDs in scope in the view that are expanded.

textTab. The currently selected tab in the text pane. The default is 0 (text). Other values 1 (HTML) and 2 (preview).

splitRatio. The current split ratio of the view and text panes. Default is 0.5, each pane using 50% of the window. A value of 1.0 equates to all text pane, a value of 0 to all View pane.

xattribute, **xattribute2**, **xdisplayattribute**, **xbarattribute**, **xcontainer**, **xquery**, **xqueryLabel**, **xbins**, **xunlimited**, **xbins2**, **xunlimited2**, **xheatmap**, **xReportStyle**. Data relating specifically to the Crosstabs view, see detail:

spacing, **aspect**, **hyperbolicScale**, **path**, **focus**, **crosslinks**, **spread**. Data relating specifically to Hyperbolic view, see detail.

hasBorder, **isVertical**, **isCentered**, **lineType**, **itemWidth**, **xspace**, **yspace**. Data relating specifically to Chart view, see detail.

showColumns, **usingCheckboxes**, **columnString**, **showFilter**. Data relating specifically to Outline view, see detail.

treemapExpression, **treemapColorExpression**, **treemapBorderColorExpression**, **treemapColorStart**, **treemapColorEnd**, **treemapDepth**. Data relating specifically to Treemap view, see detail.

attribute, **container**, **bins**, **sortAttribute**, **sortDirections**, **summaryAttribute**, **summaryMethod**, **query**, **queryLabel**, **action**, **actionLabel**. Data relating specifically to Attribute Browser view, see detail.

The difference in the degree of differing view types' use of XML data reflects how some already store much or all of their view configuration data in attribute values.

badges tag

The `badges` tag is a direct child of the `tinderbox` tag.

`<badges limit="7">`[list of up to 7 badge names] `</gallery>`

The tag stores the names of (up to) the last seven badges applied in the current document as a Tinderbox semicolon-delimited list, e.g. "flag yellow;flag blue;ok". The first value is the oldest use, the last the most recent.

preview tag

The `preview` tag is a direct child of the `tinderbox` tag.

`<preview>`[Base64-encoded string of image]`</preview>`

The image is a thumbnail of the current tab's view, encoded in Base64 so as to be valid XML.

The non-printing character in the stream is a carriage return (#13).

System Attributes: 'Internal' group

```
<attrib Name="Internal" parent="System" editable="0" visibleInEditor="0" default="" >
<attrib Name="Alias" parent="Internal" editable="0" visibleInEditor="0" type="6" default="0" >
</attrib>
<attrib Name="EntryScript" parent="Internal" editable="0" visibleInEditor="0" default="" >
</attrib>
<attrib Name="ExitScript" parent="Internal" editable="0" visibleInEditor="0" default="" >
</attrib>
<attrib Name="GeocodedAddress" parent="Internal" editable="1" visibleInEditor="1" default="" >
</attrib>
<attrib Name="WindowPlace" parent="Internal" editable="0" visibleInEditor="0" kind="21" type="100" canInherit="0" default="Rect[ 50 50 750 1150]" >
</attrib>
</attrib>
```

The System attributes' groups, as defined in a default TBX, include an 'Internal' group. This group of attributes are not exposed via the Tinderbox user interface but appear to be a standard part of the TBX document defaults for internal use vi the app. The exact purpose of all the following is unknown and users should not attempt to use or alter these:

- $Alias.
- $EntryScript.
- $ExistScript.
- $WindowPlace. Likely used in v5 but not since v6.

Applescript

NOTE: *the new support for AppleScript is a complex feature to add to a mature application. It may take time for key features to stabilise and to be able to give canonical code examples (which likely may be done in a resource other than aTbRe Patience is advised.*

NOTE: *scripts can do very bad things to a document; keep good backups* .

Tinderbox offers limited AppleScript support, making it easier to automate workflows with other applications. Explaining general AppleScript functionality is outside the scope of aTbRef but the scripting language is well supported with learning resources both online and in book form.

Below are some sample expressions that Tinderbox AppleScripting supports:

- Notes, including agents and adornments
- Refresh the Tinderbox UI
- Attribute values
- Application properties
- Attributes
- Selections
- Evaluating expressions
- Links
- Link Types

Notes, including agents and adornments

IMPORTANT: case sensitivity when resolving note name matches differs from action code

AppleScript's `named` operator is, unfortunately, *case-insensitive*. Though consistent with Finder it is unlike normal Tinderbox treatment of $Name data. So, be aware that:

```
note "X"
```

is only a short way to write:

```
note named "X"
```

… i.e. *both* examples use `named`, even if it is only explicit in the latter. This means the code acts *case-insensitively* to match the first ($OutlineOrder-based) match to notes named 'X' or 'x' which otherwise would be unique names in Tinderbo (action code). By contrast Tinderbox action code can distinguish note 'X' from note 'x' but if passed two notes called 'x' as a match, it would pick the first by outline order.

If affected by this consider resolving correct identity based on $Path or $IDString data instead and see the `find note in` example further below under referencing notes. If needing to do multiple operations on a note (or in scope of a note) use a more precise method to set a reference to the note and then re-use the reference as needed.

Creating new notes

This creates a new top-level note, and creates an agent in that note:

```
set myNote to make new note in document "Workspace.tbx"
set myAgent to make new agent in myNote
```

When using the **make new** command, note that the returned designator it gives is based on the *current* outline position of the newly created note, and subsequent calls that make or delete notes might render it invalid.

To the $Name of the note [sic] created in the code above:

```
set name of myNote to "inbox"
```

Then set its $Width (see also Attribute values):

```
set value of attribute "Width" of myNote to 5
```

Getting references to notes

To get a list of all the note inside the referenced 'myNote':

```
get notes in myNote
```

Or, the agents in 'myNote':

```
get agents in myNote
```

Or, the adornments in 'myNote':

```
get adornments in myNote
```

To return the name of the third top-level note in the specified document:

```
name of note 3 of document "Workspace.tbx"
```

To return a reference to the designated note:

```
find note in [note or document] with path "/path/to/note"
```

If the target is a document, the path should be an absolute path. If the target is a note, the path can be an absolute path or a relative path with respect to that note.

A way to act on all selected notes (a common operation):

```
repeat with anItem in selection of front document
 set value of attribute "Color" of anItem to "red"
end repeat
```

… see more on selections.

To return a reference to myNote's container, i.e. its 'parent' property (see **Note properties** below),:

```
set theContainer to find note in myNote with path "parent"
```

Moving a note

To move a note to the specified container (as specified in the example above):

```
move note named "X" to theContainer
```

Deleting a note

Use:

```
delete myNote
```

Note Properties

Each note object has a number of AppleScript properties:

- `child` (action code designator 'child' or 'child[0]')
- `color` (i.e. $Color)
- `lastChild` (action code designator 'lastChild')
- `name` (i.e. $Name)
- `nextSibling` (action code designator 'nextSibling')
- `parent` (action code designator parent)
- `previousSibling` (action code designator 'prevSibling')
- `text` (i.e. $Text)

In a few cases the property is an alternate way to access a specific system attribute for the note, but for most these properties are the equivalent of calculated item designators in action code.

Refresh the Tinderbox UI

The expression:

```
refresh theNote
```

… informs Tinderbox that changes have been made to a note and the user interface may require updating.

Thus if, a script alters the note's render in the view pane or its $Text or Displayed Attributes, a 'refresh' call ensures the UI reflects the changes just made via AppleScript.

Attribute values

Attribute values

Set the value of an attribute, here $Width:

```
set value of attribute "Width" of myNote to 5
```

Note the AppleScript limitation of the (implicit use of the) `named` operator and its case-insensitivity matching names.

To get the value of a designated attribute:

```
get value of attribute "Width" of myNote
```

Or, for an offset address:

```
get value of attribute "Width" of (lastChild of MyNote)
```

Attributes with local values can be reported:

```
get the localattributes of TheNote
```

… returns a list of attributes for a note where (both):

- the note has an immediate value for that attribute, rather than inheriting a value from a prototype or the attribute default.
- the attribute is not intrinsic, hidden, or deprecated.

To remove any local value assigned to the designated attribute, restoring the inherited or default value:

```
delete value of attribute "Width" of myNote
```

This is akin to the action `$Width=;`.

The default value of an attribute can be read or set from the note [sic] context:

```
get defaultValue of attribute "Width" of myNote
set defaultValue of attribute "Width" of myNote) to "5"
```

Don't forget to refresh the UI, to ensure the Tinderbox window reflects the changes made via scripting.

Application properties

The `application` object has four read-only string properties:

- `build`: build number of the app, e.g. 'b640'.
- `frontmost`: a boolean that is only `true` when/if the Tinderbox app is the active/frontmost app.
- `name`: the app's name as seen in Finder, e.g. 'Tinderbox 9'.
- `version`: the app's current version, e.g. '9.6.0'.

Attributes

Attributes are an element of the `document` object, though they may hold/inherit a value at the scope of a `note` object.

User Attributes

Scripts can create new user attributes:

```
tell application "Tinderbox 8"
   set doc to document "xtest.tbx"
   set newAttr to make attribute in doc
   set the type of newAttr to "date"
   set the name of newAttr to "myNewAttribute"
end tell
```

Or, to test for an attribute before use, and make it if not found:

```
tell front document
   -- check URL type attribute $HookURL exists or make one
   try
      set hookTest to attribute named "HookURL"
   on error errMsg number errNum
      set newAttr to make attribute
      set the type of newAttr to "url"
      set the name of newAttr to "HookURL"
      set hookTest to attribute named "HookURL"
```

```
   end try
end tell
```

Be aware that when using **make new attribute**, if a user attribute already exists with the designated name, the existing attribute is modified. If a system attribute already exists with the same name, no changes are made and no attribute is created. Therefore it is best to *test for an attribute's existence* before (re-)making it.

user attributes may be accessed as a list:

```
set userList to user attributes of front document
```

User attributes may be renamed. Note that references to attributes specify attributes by name, and so existing references are invalidated after the attribute is renamed. You can get a new reference to the renamed attribute thus:

```
tell front document
   set attrRef to attribute named "OldName"
   set name of attribute "OldName" to "NewName"
   set attrRef to attribute named "NewName"
end tell
```

The `type` of attribute (`kind` pre-v8.0.4) determines the attribute type, and may be any of the following case-sensitive values: action, boolean, color (N.B. US spelling), date, dictionary, email, file, font, interval, list, number, set, string, URL (N.B. case)—i.e. any of the currently defined Tinderbox attribute data types.

Attributes have a read-only property, 'category', that groups related attributes to their attribute Group as seen in Get Info. The category of renamed and deprecated attributes is returned as the category of their replacement.

Selections

Selections are an element of the `document` object.

Setting a selection

To select a named note:

```
set selected note of front document to "Test note"
```

… selects a note. *N.B. presently, selecting more than one note is not supported* .

Working with selections

The current selection:

```
set mySelection to selection of front document
```

… returns a list of a selected note(s). If several notes are selected, to code returns one of those notes, *typically* the first selected note. If no notes are selected, this returns 'missing value'.

To ensure all of the currently selected note(s) are captured:

```
set mySelection to every selection of front document
```

Note that `selection` is an element of a `document` object, so must be used with a specified document as above, or the code will not compile. **But** that form cannot be used if already inside a document tell block (code errors). Instead use `selected note` as below.

To get a reference to a selected note:

```
set mySelection to selected note of front document
```

… returns the selected note. If already inside a document tell use :

```
set mySelection to selected note
```

If several notes are selected, to code returns one of those notes, *typically* the first selected note. If no notes are selected, this returns 'missing value'.

To put a list of names (also works for other attributes) of all selected notes into an AppleScript list (theNames):

```
tell selection of front document to set the theNames to value of attribute "Name"
```

Evaluating expressions

Tinderbox allows action code in the document to be used in two ways: act on and evaluate

evaluation of an expression:

```
evaluate note with expression
```

The first argument note is an AppleScript specifier for the note, which will be bound to this for the evaluation. The evaluate command may be issued to either the document or to a specific note.

To remove any local value assigned to the attribute $Width:

```
delete value of ( attribute of theNote named "Width")
```

This is equivalent to the Tinderbox command `$Width=;`.

Performing actions

To perform an action on the designated note:

```
act on theNote with "…action…"
```

An 'action' is typically one or more assignment or conditional statements (expressions), such as `$Color="red";$Badge="ok";`. Within an 'act on' code, `=` means an 'assign';

Thus an 'act on' is akin to stamping a note (altering the note via the effect of the action code).

Note: the '`act on`' command does not return a value.

Performing evaluation

This returns the result of evaluating an expression:

```
set myResult to evaluate theNote with "…expression…"
```

Within an 'evaluate', `=` means a 'comparison' (although the unambiguous operator == is preferred). An 'evaluate' operation presumes the content of the note is action code whose evaluated result is to be returned.

Links

Notes possess a property, `links`, which is a listing of all outbound basic and text links from that note, i.e. any outbound intra-document links. Links are read-only and have three properties: the source note, the destination note, and the path name.

Note: external links, e.g. web links are not accessible to AppleScript via the `links` property, though they may possible be accessed via the contents of the $Text attribute of the note, as a rich-text feature.

Link Types

Scripts can access link types, which are accessed form the document scope:

To return a link type, by name: `linkType named "agree" in front document`

To return a list of all link types defined in the current document:

```
linkTypes in front document
```

to create a new link type and set some of its properties, other than its name, to non-default values:

```
tell document 1
   make new linkType with properties {name:"name", color:"green",bold:true}
end tell
```

If a link type with this name already exists, no new type is created and the properties are applied to the existing link type.

Link types may also be deleted:

```
delete linkType "experimental" in front document
```

About aTbRef

As a reference file, aTbRef is explicitly and deliberately ***not*** a 'how-to' resource, i.e. it explains how parts of the app work rather than how to employ those app features in the context of a user's personal work tasks. The latter are questions that can be addressed by tutorials or in the user-to-user forums (more on other Tinderbox help resources).

aTbRef9 is baselined on v9.5.0b597 (released 9 December 2022) and includes changes up to and including v9.6.1b638 (15 Aug 2023). This web resource of c.2,500 discrete static HTML webpages is created from 2,829 discrete notes and a further 465 aliases within the source Tinderbox document. The overall number of notes (2,974), agents (151), adornments (4) and aliases (6,574) in the entire document document comprises 9,703 items. These items contain 407,148 words, and use 6,225 internal (i.e. inter-note) links.

Updates to aTbRef

This document is under near constant revision. Generally, only affected pages are altered and changes since the baseline version are annotated with the version in which the change occurred. This means the 'last updated' time on the home page may not always be correct, but it is a pragmatic trade-off in terms of maintenance load.

A full site re-upload is done for all new releases, i.e. during version X, for every X.N or X.N.n release for as long as major version X is current.

Those users who always update to the current version can generally just ignore the text's references to version numbers (apart from checking the most recent changes) whereas users of older versions or without current update support can easily see whether their installed version includes certain improvements or not.

Some legacy syntax is no longer documented. Those needing to use legacy syntax should refer to older versions of aTbRef.

~~To assist TBX-based users watching the version and build checkers, a full local export is done when feasible even if only a few affected pages are uploaded. In such circumstances this will include the RSS/Atom feeds, the zipped TBX and the index page (so the latter's updated time will be correct).~~ Discontinued.

Obtaining the aTbRef source TBX file

NOTE: if you are using the source TBX file version of aTbRef, bear in mind nearly all of the caret (^) symbols will be written doubled in note $Text in order that they do not get interpreted as export codes when outputting the HTML pages. A further benefit from using—or just viewing—the source TBX is that is contains all the export templates needed to produce this site and which can be instructive if learning the HTML export aspects of Tinderbox.

A recent version of this file (including all needed HTML export templates) may be downloaded from the root folder of this HTML website:

https://www.acrobatfaq.com/atbref95/aTbRef95.zip

The ZIP archive will extract to copy of the aTbRef.tbx file. You can move this anywhere you like and open it like any other TBX file.

The images used in the website are not stored in the TBX for a variety of reasons by primarily to reduce the size of the TBX file. For use with HTML export, the images are available as a separate zip:

https://www.acrobatfaq.com/atbref95/images.zip

The above unpacks to a folder "images" that should be placed inside the folder to which you exported your HTML from aTbRef.tbx, i.e. alongside the "index.html".

Using a local HTML version of aTbRef:

- Open the aTbRef.tbx:
- Use the File menu, Export as HTML.
- You will be asked to select a folder for export or make a new one. Tinderbox will remember this location for subsequent re-export of the TBX.
- Once the export process is complete, open the the export location in Finder.
- Drag the 'images' folder unpacked from the separate ZIP download into the HTML export folder.
- In the export folder double-click 'index.html' to open the Home page of the site. If you want you can make an alias for the file and place the alias elsewhere, or make a bookmark in your browser.
- The exported website runs entirely from within its own folder, so if desired it can be moved on your Mac or even to another computer/server on your LAN.
- For those in organisations with official controls on publication on their LAN, a Creative Commons licence is included which should allow you to use this resource locally.
- If you wish to stop your local exports trying to make new zips, before exporting locally, open the notes 'Zipper-images' and ' Zipper' in HTML view and untick the 'Export' box.

To be able to view images used by the HTML site from within the TBX, even if you do not use the HTML export:

- Download and unpack the images ZIP.
- Place the resulting 'images' folder where you need it, either for HTML use (see above) or just for use with the TBX.
- Find that folder in Finder, select the Finder and leave the Finder window open.
- Open the TBX and find the Image Linker agent in the UTILS section.
- Open the agent's text window.
- Switch back to finder and drag the 'images' folder onto the 'File' button in the agent's displayed attributes table.
- This populates the agent's $File.
- The agent's query finds all notes with a $WebImage (i.e. which use a screengrab in the HTML) and combines the agent's $File data with the note's $WebImage to set the note's $File to the correct local path to the image.

The online HTML version of the above TBX's output will be found at:

https://www.acrobatfaq.com/atbref95/index.html

Screengrabs

The original grabs here were originally made using Tinderbox v2.3.4 and v2.4.0 and have been added to as new versions have changed the UI or new features have been added. Grabs are created with Monosnap with post-editing if required ir GraphicConverter, ImageOptim or PNGCompressor. Originally in JPG format, all newer grabs are in PNG format for better quality/compression.

The images are not stored in this TBX but are online at:

https://www.acrobatfaq.com/atbref95/images/

or in the /images/ sub folder of earlier aTbRef versions still online. The image folder can be downloaded for local use with an export of aTbRef. Should you wish to download the images for local use, please do so. I would ask though that, for bandwidth conservation, you do not hot-link to the online images apart from occasional references (e.g. blog items, etc.).

There is no index (as a web page) of the files in the above folder. The intent has to provide images of every dialog menu and window/view in the application and illustrate some of the more visual UI features.

aTbRef's HTML export templates

All the templates used to export this file to HTML are stored in the aTbRef TBX file inside the top level container called "TEMPLATES".

HTML5 Templates

In Feb 2013, the templates were updated top use an HTML5 DOCTYPE and to lose all the old tables. The first method duplicated a number of templates and includes (those with a 5 prefix or suffix). As a result of this it became apparent that the 4 main templates could be merged into one, using more conditional compilation. Thus all the main content individual pages are created by: *5-basic_all*.

A boolean $WebImageLandscape indicates whether the image should be be placed above/before text, or (for most, portrait oriented, images) are floated right in the main text. As the preferences section uses aliases for the Doc-level preference notes, but needs different screen grabs, $WebImageA holds the alternate image and another conditional is used to figure this out.

Old templates are marked with a strike-through in the TBX to help identify those still in use.

Shortly after the move to HTML5, it was discovered that the ^similarTo^ links did not evaluate correctly when placed in an include. The first *5-basic_all* was thus saved as an archive copy (named *5-basic_all (changes for similarTo links)*) and *basic_all* was updated to place the ^similarTo^ code back in a main template. As a result the on/off boolean attribute for the old include was moved into the main template: this allows suppression of the similar-to links and a circa 20-fold increase in the time taken to export (useful when testing!).

As it happens, the need for boilerplate includes becomes less with the move to one main template, although it does segment out the code slightly, making it easier to find the right part for re-editing. So, for now most of the includes remain.

The *5-basic_all* template contains 4 booleans to turn off features that slow export or require web (WAN) access:

- $UseGoogleTracking. If `false`, omits host domain Google Analytics tracking. Needs web access and not pertinent if run locally.
- $UseGoogleTranslate. If `false`, the template omits the Google Translate widget (top right of page) which loads from the web.
- $UseSimilarTo. If `false` export is much faster (as explained above).
- $UseGoogleIndexing. If `false` this omits the Google-based site search which requires web access.

Original HTML4 templates

Originally, the site exported all content via 4 templates:

- *basic*: the default & used for most pages.
- *basic_grabs*: for pages with portrait oriented images.
- *basic_menu_bars*: originally for menu bars and then all pages with landscape oriented images.
- *basic_change*: only used for change log agent pages.

In turn these used/included:

- *basic_child_item*: used for the Basic Comparison Operator listings and some deprecated listings.
- *basic_item_attribute*: used for a Displayed Attribute like table for notes about attributes.
- *basic_item_code*: used for a Displayed Attribute like table for notes about action codes.
- *basic_item_operator*: used for a Displayed Attribute like table for notes about action codes.
- *change_item*: used for change log items.

The HTML sitemap uses:

- *sitemap*: the 'envelope'.
- *sitemap_titles*: the iterating 'letter' template.

The XML sitemap uses:

- *xml_sitemap*: the 'envelope'.
- *xml_sitemap_item*: the iterating 'letter' template.

The Atom feed uses:

- *news_atom*: the 'envelope'.
- *newsitem_atom*: the iterating 'letter' template.

The RSS feed uses:

- *news_rss*: the 'envelope'.
- *newsitem_rss*: the iterating 'letter' template.

The version checker uses: *version-check*.
The build-version check uses: *version-build-check*.
The CSS styles page uses: *body_text_only*.
The image zipper uses: *zipper-images-code*.
The TBX zipper uses: *zipper-code*.
The BOILERPLATE includes all use: *body_text_only*.
A '*redirect*' template is included for later use when the current baseline is superseded, diverting users to what will be the new URL.
A few other basic templates are also included for general test/admin purposes.

What is the 'Zipper'

To make sense of the build checking feature, it is necessary for aTbRef's author to remember to upload a ZIP of the current aTbRef's TBX. Although saving the TBX before full HTML output remains a manual task, the 'Zipper' note uses as its export template a runCommand() action to use a command line to zip the current TBX and place it in the root of the aTbRef export folder ready for FTP upload; nothing is actually exported in terms of HTML for the Zipper note.
As the working directory for CLs called from Tinderbox is `' /'` (i.e. volume root) it is necessary to encode the path to from there to the TBX: it is in $MyString for the Zipper note.
Unless anyone shares the same path on their Mac for storing their local aTbRef TBX this mechanism is unlikely to affect users wishing to make local HTML exports. At worst you will acquire an extra zipped copy of the TBX.

What is the 'Zipper-images'

This process makes a separate zip of the images folder for download by those wanting the image for local use.
You can safely delete the note if exporting locally.

Turning off Google tracking

As of late august 2011, Google analytics tracking code has been added to the pages. The code is only intended to track ages running on the 'acrobatfaq.com' domain, but those using the the TBX to create their own local sites may wish to remove this code: details for this follow further below.
The tracking code is google-supplied JavaScript inserted in the head of the 5 HTML page templates (basic, basic_change, basic_grabs, basic_menu_bars and sitemap). The code is inserted via a conditional include ($UseGoogleTracking) and is stored in the note /Boilerplate/google-tracking.
The conditional include has been used to allow those using their own local export and uses this code:

```
^if($UseGoogleTracking)^^include(google-tracking)^
^endIf^
```

The line break is placed deliberately inside the if statement to avoid a blank line being emitted should the if condition evaluate as `false`. The controlling user attribute is a Boolean, making disabling the include very easy:

- Open the Attributes palette at the User pane.
- Find and select the user attribute *UseGoogleTracking*.
- In the 'Default value' box, delete *true* and replace it with *false*.
- Click the 'Change' button. This step is important as it saves the change made in the last step.
- Done!

Older aTbRef baselines

As new Tinderbox versions can bring significant change, a new sub-sites is created for new major version (since v5). Older versions are retained, but not updated and may be found here:

- v2.4.0 through to v.4.2.5. Still available at: https://www.acrobatfaq.com/tbx/index.html.
- v4.5.0–v4.5.3. Still available at: https://www.acrobatfaq.com/atbref45/index.html.
- v4.6.0–v4.7.1. Still available at: https://www.acrobatfaq.com/atbref46/index.html. New in-version baseline to reflect significant changes to action code).
- v5.0.0–v5.12.3 Still available at: https://www.acrobatfaq.com/atbref5/index.html.
- v6.2.0 (for v6.x)–v6.6.5. Still available at: https://www.acrobatfaq.com/atbref6/index.html.
- v7.0.0–v7.5.6 Still available at: https://www.acrobatfaq.com/atbref7/index.html.
- v8.0.0–v8.9.2. Still available at: https://www.acrobatfaq.com/atbref8/index.html.
- v9.0.0–v9.3.0. Still available at https://www.acrobatfaq.com/atbref9/index.html.

Current baseline:

- v9.5.0–to date. Current site at https://www.acrobatfaq.com/atbref95/index.html.

The previous versions have been left online at their original URLs above both to aid those using old versions and to honour the many inbound links. Web pages in the old versions have a header indicating the location of the (next) most recent site.

Colophon

This file & website were produced using the following tools:

- Tinderbox
- BBEdit
- Safari
- Transmit (FTP)
- DropDMG (DMG creation)
- Monosnap (screen grabs)
- GraphicConverter
- Google Translate script
- Various minor utilities

Searching aTbRef

The HTML pages include a Google search box in the footer. Note that if using a local copy you must be online to use the search; Google will search the online pages—that might possibly be more recent than your own—and the search result links will likewise be to the online pages. If you have downloaded the TBX source, simply use Tinderbox's own Find tool to look for things.
Web engine search may be discontinued as we have no control over search engine crawlers and they are often slow to update small sites like this.
If reading the TBX source file, ignore the UTILS section which contains agents, prototypes, etc., used to create and maintain this file.

Translating aTbRef

Not all Tinderbox users are English speakers. So, to try and help with that all pages include an in-page Google Translate feature (below). The feature offers automatic translation of page content into any of the languages supported by the Translate service. Doubtless the translation it is not fluent but it may help non-English speakers get more out of this resource.

Generating webpages from aTbRef's TBX

EXPORTING aTbRef

aTbRef is a document intended, by design, to be exported to HTML.

The TBX of this file includes the necessary templates to enable that process.

Origin of aTbRef and acknowledgements

This TBX and website was created by Mark Anderson, and is essentially expanded documentation of the Tinderbox app. Originally a private resource, experimentation with HTML Export showed how easy it was to make it a web resource. In the spirit of sharing online, the website is offered as a public resource. As the source TBX holds all the export templates and shows a few interesting code techniques, it too is offered as a downloadable resource.

The current document is baselined on version 9.0.0. As not all users have the latest release, ongoing changes are recorded with the version where they first appear (since the benchmark version), which should aid those who do not upgrade regularly. Any per-release references to changes pre-v8.0.0 have been deleted (with a few deliberate exceptions).

The first version of this document 'aTbRef.tbx' and associated website was written in April 2005 to reflect a baseline of Tinderbox v2.4 features and has been regularly updated to encompass all version changes since. The original baseline and subsequent versions online are listed here. The previous versions have, for the while, been left online at their original URLs above both to aid those using old versions and to honour the many inbound links. Web pages in the old versions have a header indicating the location of the (next) most recent site.

Although there are probably no original notes left, the original document was based on a TBX originally created by Benoit Pointet; that document was called 'aTbRef' and the name lives on herein. The original document I inherited looked very different—see images here and here. From that source (c. February 2005) the document was greatly enlarged before first being published as a website.

Special mention must be made of Eastgate's Mark Bernstein who has been extremely helpful along the way in answering all sorts of obscure questions and has shown a lot of patience in his explanations.

Much of the update data is drawn from the Release Notes kindly provided by Eastgate as part of the Tinderbox downloads. Parts of this document that directly quote copyright materials from Eastgate Systems Inc., e.g. the manual, release notes, etc., and do so with permission from Eastgate Systems Inc.

This document, in its public form is not meant to replace or supplant the manual or the wiki as references. Instead it is simply a way of sharing what I have learned from the kindness of others about Tinderbox.

Thanks/kudos to:

- Mark Bernstein & Eastgate Systems, Inc or answering so many questions, for all the release notes and their permission to use Eastgate materials here.
- Benoit Pointet for the TBX that became the seed of this file and which gave it its name. (no current link, but here, via Internet Archive).
- Matt Neuburg for his article (here, via Web Archive) on writing Mac Online Help files in Tinderbox (for Affrus) which got me started on structured HTML export and specifically the hierarchical site navigation used in the website pages.
- Fellow Tinderbox users. Most recently in the user-to-user forum, but also previously in the old ~~forum~~ and its predecessor the Tinderbox ~~wiki~~ (latter both no longer active) Their questions have contributed to the depth, scope, and heavily interlinked nature of this resource. Their errata have also helped keep things correct. So, 'Yes', errata are genuinely welcomed!

Creative Commons Licence

This Tinderbox and the web pages created from it are written and published as a public resource for users. In case anyone would like to use the work in locations where there may be a requirement to prove permission for use of a third party's work, a Creative Commons licence has been applied to aTbRef:

Attribution-NonCommercial-ShareAlike 4.0 International (CC BY-NC-SA 4.0)

By this licence, you are free to:

- **Share**: to copy, distribute and transmit the work
- **Remix**: to adapt the work

…under the following conditions:

- **Attribution**. You must attribute the work in the manner specified by the author or licensor (but not in any way that suggests that they endorse you or your use of the work). Please use this Attribution.
- **Non-commercial**. You may not use this work for commercial purposes. The following Waivers are already granted.
- **Share Alike**. If you alter, transform, or build upon this work, you may distribute the resulting work only under the same or similar license to this one.

Please note that any of the above conditions can be waived if you get permission from the copyright holder.

Nothing in this license impairs or restricts the author's moral rights.

The full licence may be viewed here: https://creativecommons.org/licenses/by-nc-sa/4.0/

Attribution

Mark Anderson should be acknowledged as the creator of this Tinderbox document (plus associated images, HTML export templates, etc.) and website produced from it.

Parts of this document that directly quote copyright materials from Eastgate Systems Inc., e.g. the manual, release notes, etc., and do so with permission from Eastgate Systems Inc.

The input of the following (as further explained at ' Origin of aTbRef') should be acknowledged: Mark Bernstein & Eastgate Systems Inc., Benoit Pointet, and Matt Neuburg.

Waivers

Eastgate Systems Inc., the creators of Tinderbox, are granted a non-expiring waiver to use the content here in part or whole in any commercial endeavours of Eastgate Systems Inc.

Mark Bernstein, the designer of Tinderbox, is granted a non-expiring waiver to use the content here in part or whole in any personal or commercial endeavours.

Errata and contacting the author

This is effectively a continual work in progress. Please send any errata for this file & website to the author Mark Anderson: mwra@mac.com.

Techniques & demos hidden in the TBX

Although aTbRef originally evolved as a way to share extra information about Tinderbox and to explore what could be achieved using HTML export, the project has continued to explore methods of using features, even if they are not exported. If you have the TBX look at (note that many of this page's links only work in the TBX as their destination notes do not export):

- Splitting content from the back-of-house notes. Visible data of the web site is created by exporting the first root note "A Tinderbox Reference File" and its contents. Everything else—prototypes, templates, boilerplate code, etc.—is stored outside the 'content' area. The two pools of data rarely need to be worked on together via agents so splitting the two out makes for more efficient maintenance and upkeep. There are a number of items at root level as this reflect the needs of export. Be aware that Tinderbox cannot export a note to a relative position other than in its relative (TBX) outline location. Thus if you want all notes to reference a single CSS file it is best to place that at the TBX root level, etc.
- A map view of note colours. All the predefined Tinderbox colours and their shades. Also used to create an online image. The listing shows the hex values for all the defined colours/shades.
- A map view of shapes, borders and patterns. A map showing various map visualisation aspects. Also used to create an online image.
- Agents to restate major listings. Look at each of the listings of action code, attributes and export codes. Notice how there is a main listing and then this is re-started divided up by scope, purpose, etc. The agent in some cases use colour to make the main list (within the TBX) more visually descriptive as to purpose. Look at $AgentPriority setting which have in some cases been reduced to ease loading of the main update cycle time.
- Controlling agents. In the root-level UTILS section are various upkeep agents, mostly set to 'off' when not needed. An 'all agents' agent allows all such agents to be found and controlled from one location, whether in the content or back-of-house.
- Timelines. Although aTbRef does not output timelines, both the Change Log and Previous Versions containers can be viewed as Timelines and will show relevant data. The Previous Versions uses child notes that are not exported.
- Prototypes. These are used mainly for the notes in the 3 big lists, which use a number of special-to-context attributes for their Displayed Attributes. An agent-based prototype is used for the per-version change log entries. Setting just the new version number initialises a number of other features.
- Boilerplate code. The export template code is kept less complex as certain common-to-all-pages code is stored as 'boilerplate' and used in templates via ^include()^ code. In the original HTML$ templates this made more sense as there were 5 primary page templates for content, each reflecting some different layouts.
- Images. Whilst image support has varied over Tinderbox versions, aTbRef has deliberately not used embedded images. This both gives more export control (formatting, compression, naming) and allows for a smaller overall TBX. All images are placed in a root level folder and simply the filename is stored in the note, the template figuring out the path. This also allows for the images themselves to be linked to from elsewhere are reference material. Having only one image per note (if any) was a deliberate design choice originally to facilitate template design and may be reviewed in later versions as image support is improved.
- Image Linking. An agent 'Image Linker' has its $File set to the local location of the 'images' folder used to store the images used by webpages (you can adjust it if using your own local TBX copy of aTbRef). All notes using a web image store the image filename in $WebImage. The agent finds these notes and sets their $File to the agent's $File + a / + the note's $WebImage. This correctly sets $File for the note allowing the linked image to be viewed (via Finder) from the TBX. This assumes you have downloaded the images. One small exception is where the Doc-level preferences that repeat App-level ones are alias notes but use a different screen grab (stored in a separate attribute). The latter notes will open their original's image but in the circumstances that does not really matter.
- Images and templates. Originally there was a template for 'wide' images, which needed to sit above $Text as opposed to most that could mix with text. In the HTML5 versions this is removed as image inclusion is based on the image attribute and a Boolean landscape/portrait attribute.
- HTML export. All templates used are stored in the TBX in the Templates container. Originally the export was in HTML4 and used tables. This has been replaced by HTML5, and a resulting reduction in the overall number of page templates. The older set are retained, for comparison, and can still be used for local export. Template and boilerplate code notes with a '5' in their title are for HTML5 use. Compare the old and new versions to see how improvements have been made.
- Conditional includes. The ^similarTo^ code involves more computation for export: with such links included it takes 20 times longer to export the whole site. Thus the boilerplate note used to generate this output has a Boolean to enable/disable the include. Try exporting in each condition and note the difference. The Switch for this has moved to the calling template.
- HTML sitemap. This is a simple envelope/letter technique to map the whole site as a single HTML outline. Far too large an HTML page for real use, this is really just a demo.
- XML sitemap. Use to help Google spider your site more efficiently. The XML is very verbose and creates an 8+MB file. For this reason export of the XML sitemap is turned off by default (use HTML view on that item to enable it). The note calling the export has $HTMLDontExport shown as a Displayed Attribute as toggling the 'Export' box on HTML causes the view to re-preview the code which can be slow. This export alone generates a c.8 MB XML page, so do not turn it on unless you really want to see it at work!
- RSS & Atom feeds. These are basically two different flavours of the same form of syndication, and especially useful for those who use newsreaders. Both feeds use an envelope/letter method of export. As the agents making the feeds look for changed items, these are deliberately placed at the end of the root so they process at the end of each agent update cycle (which proceeds in outline order across the file).
- CSS. The file uses a Code prototype for the note holding CSS code to avoid things like hash characters being misinterpreted as quicklist items. The is no specific benefit to maintaining the CSS this way. At the time it was just another test of what was possible. It also helps keep everything needed in one place. Indeed, except the images the whole site uses only data stored in the one TBX.
- The Zipper-images and Zipper notes are used to make up-to-date Zips of the images folder (as this is not part of the TBX) and a Zip of the TBX itself. Although Tinderbox runs all agents before starting a full export, there is no easy means to trigger a save, so if using this technique it is best to run a forced update ('Update now', via the File menu) and save manually (again via the File menu) before running a full HTML export.
- See more.

Tinderbox Documentation And Other Resources

The primary form of official documentation for Tinderbox is the Tinderbox Manual (the app's macOS HTML Help). This also includes the Release Notes (pre-v6 these were a separate TBX file). The Manual is accessed from the Help menu.

Eastgate offers *user-to-user* help via the Tinderbox forum (https://forum.eastgate.com). This resource, 'aTbRef' also available online at https://www.acrobatfaq.com/atbref95/index.html, although it should be noted it is not an official Eastgate resource.

Other Eastgate resources:7

- Tinderbox website (https://www.eastgate.com/Tinderbox/)
- Tinderbox *user-to-user* forum (https://forum.eastgate.com/). NOTE: this is not Tinderbox's official tech support (see below), which should be a first port of call for urgent technical issues.
- Tinderbox's official tech support (info@eastgate.com). This is generally a faster way of accessing technical helps then using the forum.
- Old forum, lo longer online: was http://www.eastgate.com/Tinderbox/forum/.
- Tinderbox Cookbook, for action/export code (https://www.eastgate.com/Tinderbox/cookbook/).
- Tinderbox wiki, no longer online: was at: https://www.eastgate.com/wiki2/wiki.cgi?TinderboxWiki.
- The Tinderbox Way: a book by Mark Bernstein (https://www.eastgate.com/Tinderbox/TinderboxWay.html). Several editions:
  - Ed.1 c.2006 (see) alongside Tinderbox v.3.x (prior to publication there is mention as far back as Sep 2004)
  - Ed.2 May 2012 (see), 375 for Tinderbox v5.11.0.
  - Ed.3 27 Oct 2017 (see) for Tinderbox v7.3.0.
- The Tinderbox File Exchange (https://www.eastgate.com/Tinderbox/Exchange.html)
- Tinderbox tech support email (tinderbox@eastgate.com).
- Tinderbox Tutorial Series. A CD or downloadable DNG:
  - Volume 1.
  - Volume 2.

Third-party resources:

- aTbRef: this site (https://www.acrobatfaq.com/atbref). Several discrete versions are still online, baselined on earlier Tinderbox versions: see here.

If it is necessary to obtain downloads of Tinderbox versions other than the current one, e.g. because the user has not extended their licence, then use https://www.eastgate.com/download/ and select the DMG file for the appropriate version.

Tinderbox Tutorials: aTbRef's author has a Tinderbox tutorial web page linking to a number of tutorials.

- The Tinderbox Manual
- Release Notes
- Shoantel's Tinderbox Tutorials
- Some Tinderbox pre-history

The Tinderbox Manual

The Tinderbox Manual is provided in Mac HTML Help format and is accessed via the programs Help menu.

Also contains the Release Notes.

From time to time, a PDF generated from the app Help is made available from Eastgate's Tinderbox download webpage.

Release Notes

The Release Notes are provided as part of Tinderbox's built-in Help. Additional notes about new features, enhancements and fixes are added for each public release.

Open the Release Notes from the Help menu, select 'Tinderbox Help'. When the Help window opens, choose 'Release Notes' from the listing on the opening page.

Shoantel's Tinderbox Tutorials

aTbRef's author has created a number of Tinderbox tutorial webpages which are listed here: https://www.acrobatfaq.com/tb_clarify/index.html.

Some Tinderbox pre-history

This article it may help some users get a better understanding of some of the ideas and experience that fed into Tinderbox's design.

By current standards, Tinderbox is a long-lived app. Work began on a project called 'Ceres' after the launch of Storyspace v2.0 on 24 January 2001 http://www.eastgate.com/Development/40.html and https://web.archive.org/web/20010202192100/http://www.eastgate.com/Storyspace2.html and continued through 2001 with active use in making several blogs, including Mark Bernstein's own blog, e.g. https://www.markbernstein.org/November01.html. Indeed, in his blog, the day after release, Bernstein notes https://www.markbernstein.org/Feb0201.html#note_6210 that Tinderbox v1 and Storyspace v2 share a common underlying framework on which work started 15 August 1999. Meanwhile, work on Ceres started on April 20, 2001. By 14 May 2002 work on a version for the newer Mac OS X was already underway, with v1.1.0 supporting OS X being released on 1 June 2002.

In late 2002, the 'development peekhole' site mentioned work on Tinderbox for Windows, but that project was eventually abandoned. Probably usefully so as maintaining close integration with two very different OSs whilst trying to keep featur parity would need a bigger team. Mentioned 10 Feb 2004 (https://web.archive.org/web/20040607115854/http://www.eastgate.com/Development/).

In July 2003, the Tinderbox wiki opened at http://www.eastgate.com/bin/wiki.cgi (now dark) and was the first user forum.

Ceres was used to publish its first weblog on 1 June 2001 https://web.archive.org/web/20010603192437/http://www.eastgate.com/Development/. Also see https://www.eastgate.com/Ceres/index.html.

Shortly before launch, the program's name was changed from Ceres to Tinderbox in January 2002 (https://web.archive.org/web/20020201200841/http://www.eastgate.com/Development/ note for 23 January 2002), though the rationale for th change was not given. Tinderbox launched on 18 February 2002 after around a year of development under the working name 'Ceres'. At launch, system requirements for Tinderbox stated :

"*Tinderbox runs superbly on all modern Macintosh computers. We recommend a G3 or G4 and MacOS 8.5 or later. Tinderbox runs beautifully on iMacs and iBooks. Tinderbox takes up about 5M on your hard disk (less if you don't need the manual). It likes to have 16M of memory, but can cope with much less if you're short of space.*" (https://web.archive.org/web/20020212070749/http://www.eastgate.com/Tinderbox/download.html).

The Ceres/Tinderbox Development Peekhole which blogged ongoing development last posted in 28 April 2007 (https://web.archive.org/web/20070527181023/http://www.eastgate.com/Development/).

During 2013-14, the Tinderbox codebase was completely overhauled and re-written on current Apple development frameworks in Xcode, launching as v6.0.0 in May 2014. At the same time the UI was completely changed, reflecting changed user expectation and the change of underlying frameworks. Lost in the new frameworks was the old ability to drag links between windows. Indeed, the zeitgeist as the time was for single window apps that such a model proved not a good fit for all Tinderbox users and multiple document windows and a small number of independent report windows soon returns to the app.

An upside of the move to Xcode has been coherent Unicode support throughout the app, noting that not all users are using English or languages using roman script. The latter feeds through into support for locale based sorting, time and date formatting, etc. Recently, Tinderbox has also begun to experiment with some of the AI aspects in newer frameworks (thus OS requirements above baseline) to detect major language and possible key terms.

Storyspace

As noted, Tinderbox reflects some features and concepts in its older sibling program, Storyspace (http://www.eastgate.com/storyspace/). The latter was developed as a tool for writing—and reading—'literary' hypertextual works, although Storyspace users found other uses for the application. Storyspace's design reaches back to initial work by Bolter & Joyce in the mid 1980s, written in C and available on Mac And Windows. Eastgate took on the 'publishing' of the Storyspace app from its originators in December 1990 and has maintained the system since, re-coding it twice in the process. Firstly it was done in Pascal, upon taking over in the code base (for Mac and Windows). Latterly, for v3 in 2015 using Apple Xcode (macOS only) and unifying the code base for Storyspace and Tinderbox which now share many UI elements and a common XML data file format.

Storyspace is notable for being about the (only?) early hypertext system still in active development and use and is an import milestone in the area of literary hypertexts. Much of the canon of the latter was/is still published by Eastgate using Storyspace: http://www.eastgate.com/catalog/Fiction.html. More on Storyspace's place in the hypertext story is documented in Chapter of Belinda Barnett's book *Memory Machines* (https://www.anthempress.com/memory-machines-pb).

Eastgate was also involved in early hypertext systems, creating the Hypergate app in the late 1980s and Web Squirrel (for making link collections in the early days of the Web) in the 1990s. Hypergate is notable in terms of being the first use of a 'breadcrumb' notion of showing where you are, such as seen at the top of aTbref webpages or the in-app view breadcrumb bar. Interestingly, the original concept was a means to tell the user they had visited a page before, noting that in those days, hypertext navigation was new and confusing to most people.

Academic Record

Both Tinderbox and Storyspace have been regularly cited (since 1987) and used in papers published in a number of journals and proceeding, most notably the ACM Hypertext Conference (https://dl.acm.org/conference/ht) where Mark Bernstein is the more prolific contributor and author or the most in-conference cited paper *Patterns of Hypertext* (https://dl.acm.org/doi/10.1145/276627.276630). A good number of Tinderbox users employ Tinderbox in their research so Tinderbox also has an (un)credited part on a good number of other academic papers and PhD theses (all the research analysis for aTbRef author's PhD was done using Tinderbox as the primary tool).

Some Tinderbox-related Eastgate products

- **Wizards**. In the old v1–v5 app design it was possible to use 'wizards' to distribute Tinderbox project files or demo general functionality. More information can be found on the old aTbRef v5 baseline at https://www.acrobatfaq.com/atbref5/index/SyntaxLibraryTutorialsWi/TinderboxWizards.html.
- **Flint** [No longer available]. During the time of Tinderbox v4.x there was still interest in (static webpage) blogging. Flint was essentially a very rich wizard file (see above). See also: http://www.eastgate.com/Tinderbox/Flint/ and https://www.acrobatfaq.com/atbref46/index/ImportExportFormatting/FlintWeblogAssistantWiza.html.
- **Twig** [No longer available]. Twig might best be thought of as a more limited version of Tinderbox. It can be read about in the v5 baseline at https://www.acrobatfaq.com/atbref5/index/ObjectsConcepts/TinderboxvsTwig.html. See also http://www.eastgate.com/Twig/Screencasts.html.

Change Log

This section logs changes since v9.5.0b597 release (9 December 2022) which was the baseline for this updated issue of 'aTbRef'. Tinderbox Help also contains a listing recent changes that includes bug fixes which are only of direct relevance to some users.

The links below summarise changes in each version and link through to things that have been added or changed in that version. There is also a separate list of release dates of older versions.

Versions from—and including—the v9.5.0b597 baseline:

v9.6.1b638 (15 Aug 2023)

Released 15 Aug 2023. Build number is 638.

Minor changes not necessarily warranting explicit mention in an aTbRef note:

- **Action Code**:
  - The **fetch()** action is now evaluated on the agent queue, rather than on the network queue, to avoid errors when the fetch action was run during an agent update.
  - The characters ? and / are now percent-encoded by urlEncode(). Even though these are now permitted in the query portion of URLs, NSURL balks at their presence.
  - Fixed **create()** to allow for $Name value containing parentheses.
- **Miscellaneous**:
  - Corrected an unusual crash associated with the create action.
  - Corrected some possible concurrency issues relating to fetch() actions.
  - Applying a stamp that deletes links (or changes the note's prototypes) could crash, because the current editing session was ended after the stamp was applied and might reference links that no longer exist.
  - Programmatically creating a note with a title containing paired square brackets could lure Tinderbox into recursively evaluating the proposed title as an apparent expression.
  - Improve opening of exceptionally large documents.
  - Addressed a deadlock at startup by avoiding reference to $Path inside Lynx::InHintsFolder()
  - Slightly relaxed the protections on processing […] in paths, in order to continue to support offset references such as `nextSibling($MyDictionary['robin'])`.
- **Windows**:
  - See v9.6.1 Release Notes for more detail of minor points.

v9.6.0b632 (1 Aug 2023)

Released 1 Aug 2023. Build number is 632.

Minor changes not necessarily warranting explicit mention in an aTbRef note:

- **Action Code:**
  - The result of [String].size is now the number of unicode characters in the string. Previously, it was the size of the UTF-8 string in bytes, which differed in strings containing multibyte characters.
  - Tinderbox no longer forbids expressions that interrogate properties of the root note—the parent of top-level notes. For example, a top-level note can now get the value of $MapBackgroundColor(parent) if it wants to know the background colour of the top-level map.
  - A comment followed by a var statement caused some statements following the var statement to be treated as comments.
  - The operators **inside(path)** and **descendedFrom(path)** once again properly recognise designators as the **path** argument.
  - **collect(scope,attribute)** and its relatives failed when **scope** was a **find(…)** expression. Now fixed.
  - **links.[inbound/outbound].path.attributeName** expressions failed if the attribute was a URL attribute, because the system was misinterpreting the scheme (such as https:) as a dictionary key.
  - Syntax highlighting now recognises keywords such as function when they appear at the start of a note.
  - Syntax highlighting of quoted strings now includes the closing quote.
  - Revised parsing of action( ), which sometimes saw syntax errors in valid expressions.
  - Indexed references to a list of lists, e.g. `var:list myMatrix = "[0;1;2];[3;4;5][6;7;8][9;10;11]";` or `$MyList=[0;1;2];[3;4;5][6;7;8][9;10;11];` failed because the parser stripped brackets incorrectly. The parser has been revised to handle nested lists more consistently. A separate note will describe the state of list and dictionary syntax as I understand it.
  - **%matches** is now treated by TbxCodeField as a keyword.
  - **.format()** now uses "duck typing" to assess the type of the object being formatted if the type cannot be determined. For example, if `$MyList is [ [1;2] ; [3;4] ]`, then `$MyList[0].format(",")` inspects the result, `[1;2]`", decides that it looks like a list, and formats it as a list.
  - Comments begin with // and continue to the end of the line. Formerly, comments terminated with a second //, but URLs are not unheard of in comments and this caused confusion.
  - Fixed: the eachLink(){} action crashed if the action deleted the targeted link.
  - Corrected handling of lists in dictionaries.
- **AppleScript/OSAscript**
  - The command `make new linkType with properties {name:"tester6", ...}` now correctly recognises the property "broad" for broad links.
- **Attributes:**
  - $TextExportTemplate is now deprecated.
- **Find:**
  - It appears to have been possible for a torn-off Find window to receive changes before it is ready to receive them. Steps taken to correct this.
- **Get Info:**
  - Book. This once again displays covers for many (though, of course, not all) books.
- **Hyperbolic view:**
  - If the hyperbolic view is not excessively complex, Tinderbox performs some force-directed layout adjustments to better use the space.
  - When creating a link, the rubber band line starts at the correct position. Previously, it was displaced 40px.
  - The degree of spread is now saved with the tab and restored when the document is reopened.
  - Occasionally, links failed to draw when they fell precisely along the diameter of the unit circle.
  - Previously, the scaling of link labels had been computed incorrectly.
  - Further revisions to the hyperbolic browser, coping with issues regarding notes close to the edge of the universe.
  - Panning the hyperbolic view is now restricted so that part of the graph always remains in view. Previously, it was possible to move the entire graph to the far distance, leaving no hint of where to find it.
  - The tab contextual menu command for setting Hyperbolic view now sets the focus to the tab's currently selected note.
  - Hyperbolic view now defaults to using *untitled links for initial (display) tree-building.
  - Hyperbolic View no longer offers a single crosslinks checkbox, as we now have fine-grain control of spanning tree construction and cross links.
  - Hyperbolic view omits "Delete" from the contextual menu of the focus note, since deleting the focus would leave nothing in the view.
- **Import:**
  - Updated for new Ventura drag flavour for calendar items.
  - Pasting from Microsoft Word® into the text pane set $URL to an applewebdata: URL. Because these URLs aren't useful, we no longer record them.
  - DEVONthink watch folders should now work as expected.
  - When importing a spreadsheet with a column headed by "0", Tinderbox created an attribute named "0" which caused trouble. Tinderbox now prepends "_" to column headings that begin with a digit.
- **Inspector:**
  - Tinderbox Inspector's word count failed to update in large documents, and failed to adapt when changing documents. We now throttle updates to this instead, because counting hundreds of thousands of words can take time. The reported value may lag the actual value by several seconds.
  - Corrected the width of the Badge control of the Appearance inspector.
  - When using the Quickstamp inspector to change a boolean attribute that is a Displayed Attribute, Tinderbox now updates the DisplayedAttributes table to reflect the change. Formerly, this update was deferred to the next selection, which was confusing.
- **Links:**
  - When a new link type is created in the Document Inspector's Links pane, it takes its colour from the current colour scheme. Previously, the new link type used black links.
  - A major refactoring of the link parking space and link tracking is under way. Visible changes should be undetectable or minimal, please report discrepancies.
  - Addressed an error in accounting for text link changes when editing a multiple selection in which text links appear in the second or subsequent notes.
  - When creating links, the Create Link pop-over's swap button now reflects any pending changes to the destination name.
  - Tinderbox no longer stores the source Document ID and destination Document ID in links when the source and destination both reside in the current document.
- **Map view:**

    - An outmoded animation in the Cut handler for map view sometimes left "ghost" views, as animations for removing and selecting the view could wind up interacting.
    - In map view, the inbound and outbound stub counts are once again positioned correctly.
    - Implemented methods for adding new CSS style rules to poster notes.
    - Implemented methods to enable Javascript in poster notes to perform Tinderbox actions and to evaluate Tinderbox expressions.
    - Corrected the text area computation for left tag to have the same size, and therefore the same line breaks, as right tag.
    - TbxDocument's close method could get into trouble if the agent queue was very busy — especially if pending actions would add more work (such as screen updates) to the agent queue. Tinderbox now clears pending work, sinc this will either not need to be performed (for screen updates) or will be performed on reopen (for agent updates) before putting the action recycle operation onto the queue. This should reduce problems on close.
    - The Built-In Prototypes and Hints containers now respect $CleanupAction.
- **Miscellaneous:**
    - Tinderbox once more performs the edict of every note — not only the selected notes — when a document is loaded.
    - Revised handling of the internal indexing primitives. These are especially tricky because they must run in the background, and must be careful to sequence themselves relative to each other and to changes performed by the user and by actions. This issue may have been responsible for crashes sometimes encountered when bringing a Tinderbox window to the foreground after using another application.
    - In building the Stamps menu, separators now become separator menu items (if not named) or disabled menu items (if the separator has a name).
    - Tinderbox Help documented the wrong order of arguments for create, createAgent, and createAdornment.
    - Prevented a crash observed when Tinderbox to attempt to use the index of a note that doesn't exist.
    - Menus: Format ▸ Style ▸ Standard Font and Format ▸ Style ▸ StandardSize can now be applied to a multiple selection.
    - Menus: Format ▸ Style ▸ Reset Margins is available from the view pane.
    - The application's disk image is now signed with the new notarytool.
    - Addressed a crash after closing a document with an inter-document link.
- **Outline view:**
    - When double-clicking a note in the Links pane to select it, Tinderbox will expand any collapsed ancestors of that note in outline view.
- **Roadmap view:**
    - Less space is reserved for link comments in the torn-off Roadmap window. When the selected link comment is empty or brief, or when no link is selected, even less space is reserved.
- **Taggers**
    - Re-installing built-in Hints overrode changes to pre-existing highlighters and taggers that had been user-edited. This should no longer occur.
- **Text pane:**
    - After selecting from the pulldown value menu in the displayed attributes table, the currently-selected attribute remains selected for list and set attributes, since you might well want to add or remove several elements. For other attributes, the next row of the displayed attributes table is selected.
    - The Displayed Attributes table should no longer terminate the editing session of a value when a rule or agent action changes that value.
    - In the Displayed Attributes table, the action of the values pulldown menu has been improved when editing sets and lists. Now, any pending edits are recorded, and then the chosen value is added to the set or list. Previously, pending edits were discarded.
    - Adjusted the placement of the error icon in the text pane. It now aligns with the top of the title.
    - The minimum height of text windows is now increased in proportion to the number of displayed attributes.
    - The behaviour of the Displayed Attributes table's values pulldown is improved when editing lists or sets. Tinderbox now records any pending edits from the text field, and then adds the chosen value to the list or set if it was not present. If the value was present, it is removed.
    - The Displayed Attributes table does a better job of allowing sufficient space for multi-line attribute values.
    - When entering American-style dates without a time, Tinderbox now chooses noon rather than midnight for consistency with other methods of converting text to dates.
    - After editing date attributes in the displayed attributes table, Tinderbox now selects the next value in the table, not the first value.
    - When dealing with long (>100,000 character) texts, Tinderbox now limits the work it performs in cleaning up margins and paragraph styles in order to improve performance.
- **Windows:**
    - See v9.6.0 Release Notes for more detail of minor points.

This version is cited in the following notes:

- %matches (query back-references)
- $N (query back-reference)
- Attribute Browser view
- attribute(attributeNameStr)[keyStr]
- attributes tab
- Backslash: escape character
- Built-in export Templates
- Caret delimiters: export code operators
- changed([scope])
- Choosing to use variables or attributes in code
- Colon: ad hoc delimiter in some action operators
- Colon: dictionary key-value pair delimiter
- Coloured syntax highlighting in Action code
- Comma: function argument delimiter
- Creating draft emails
- Creating new posters
- Curly brackets: defining code blocks
- Curly brackets: dictionaries and nested dictionaries
- Date-Type Attributes
- Designators
- Dictionary-Type Attributes
- Displayed Attributes table
- document()
- Dollar-sign prefix: attribute references
- Dollar-sign prefixed numbers: macro arguments
- Dollar-sign prefixed numbers: query back-references
- Double forward slash: action code comments
- Dragging content from DEVONthink to Tinderbox
- Dragging in Apple Calendar events
- Dragging URLs into Tinderbox views
- Email
- Email-Type Attributes
- EmailSubject
- EmailTemplate
- Explicit declaration of dictionaries using curly braces
- Explicit declaration of lists using square brackets
- Expressions in attribute offset addresses
- fetch(urlStr,headersDict,attrNameStr,cmdStr[,httpMethod])
- Font sub-menu
- Forward slash: folder delimiter in paths
- Full stop: dot-operators
- Geographic Adornments
- Hover tab
- HoverBackgroundColor
- Hyperbolic view controls
- Interior tab
- IsSeparator
- Item Note Designators
- JSON.each([pathStr]){actions}
- JSON.json[keyStr]
- JSON.jsonValue(pathStr)
- link tag
- Links tab

- List.select()
- Map Posters
- map tab
- Navigating hyperbolic view
- nextItem
- nextSiblingItem
- Note Flags
- Notes displaying body text
- Outline view
- Parentheses: arguments for action code operators and user functions
- Parentheses: attribute 'offset' references (offset addressing)
- Parentheses: controlling parsing of code
- Pasting notes to a different TBX: $Created and $Modified
- PosterCSS
- PosterLabels
- Posters and performance
- PosterSettings
- PosterTemplate
- PosterURL
- PosterX
- PosterY
- PosterZoom
- previousItem
- previousSiblingItem
- Printing from Tinderbox
- Prototype: Event
- Punctuation and special characters in definitions, actions and expressions
- require(featureName)
- ScreenHeight
- ScreenWidth
- select()
- select(scope)
- Semicolon: expression delimiter, code line end
- Semicolon: list and dictionary item delimiter
- Separator
- Simulating global variables
- Smart Links and URLs
- Some Tinderbox pre-history
- Sort Transform pop-up list
- Square Brackets in code operator explanations
- Square brackets: dictionary data keys
- Square brackets: dictionary keys with multiple values
- Square brackets: in documentation, optional arguments
- Square brackets: list indexes
- Square brackets: lists and nested lists
- Stamps
- String.eachLine(loopVar[:condition]){actions}
- Symbols used in Mathematical and Logical operations
- Symbols used in Regular Expressions
- System tab
- Text area - Links panel
- Text tab
- The library sorting note
- unlinkFrom(scope[, linkTypeStr])
- unlinkFromOriginal(scope[, linkTypeStr])
- unlinkTo(scope[, linkTypeStr])
- unlinkToOriginal(scope[, linkTypeStr])
- Updating posters
- User tab
- Using long sections of code—code notes
- Using regular expression back-references

v9.5.2b606 (28 Feb 2023)

Released 28 Feb 2023. Build number is 606.

Minor changes not necessarily warranting explicit mention in an aTbRef note:

- **Action Code:**
  - The operator sum_if() and its kin now bind the 'that' designator as expected. For example, `sum_if(children,$Price>$Threshold(that),$Price)`, adds up the prices of each child of the calling note, provided that its price value exceeds a threshold stored in the parent note.
  - Assigning the $SiblingOrder of a note sometimes left the note with a sibling order one greater than the intended position.
  - .captureWord and .captureToken crashed if the captured element terminated at the end of the string.
  - **collect()** and related operators can again use find() as a designator.
- **Attributes:**
  - The **countIf()** operator sometimes reported a parsing error when no error was in fact observed.
- **Attribute Browser view:**
  - In Attribute Browser, group headers were not sufficiently distinguished from normal rows. This was a temporary fix for a longstanding cosmetic glitch which cause the highlighting of group rows to fail intermittently and unpredictably. The underlying cause is that the NSTableView assumes that it is free to set the size and font of the textView field. We now longer use the textView field at all, and so the NSTableView no longer interferes. Essentially, from v9.5.2, the group headers are more clearly distinguished within the view.
  - A potential hang occurred in attribute browser when the attribute browser's query was evaluated during an agent update.
  - Corrected the layout of the Attribute Browser's column picker.
- **Export:**
  - In HTML Export and Preview, superscripts and subscripts are now recognised and exported with the tag pairs $HTMLSuperscriptStart/End and $HTMLSubscriptStart/End.
  - When referring to a template by name rather than full path, Tinderbox now uses the first template note of that name, searching in outline order. Previously, Tinderbox selected any note of that name, without checking that the note was intended as a template. This caused unexpected conflicts when template names like "Preview" were used elsewhere in a large document.
  - When viewing the Preview of a note, Tinderbox was (understandably) confused if the export path of two or more notes were identical. For example, consider two different notes each export their contents to the file `/path/to/Note.html`. Of course, in a normal export, one exported note will overwrite the file exported by the other. Confusingly, however, this meant that if you selected a note while Preview was active, the selection would automatically switch to the first identically-named sibling. Instead, Tinderbox now allows you to select the intended note.
- **Import:**
  - Pasting text from Microsoft Word® no longer places a useless applewebdata:// URL in the $URL attribute.
  - Watching Tot works again, after an error in interpreting markdown caused trouble.
  - Files with extension .md, .mmd, and .markdown that are dragged into a Tinderbox view are now converted to styled text. This was remarkably difficult.
- **Inspector:**
  - Document Inspector, system tab, the $OutlineColorSwatch is now marked as deprecated (indicated by strikethrough text).
  - When you are using the search field in the System Attribute Inspector, switching to another application and then returning to Tinderbox no longer resets the selected attribute. Instead, Tinderbox selects the attribute for which you were searching.
- **Links:**
  - The link info widget (in map view) is again drawn correctly; a workaround for a system font metric issue is no longer needed.
  - Corrected a crash when using only a prototype or a place as input for the ziplink method. E.g. [[@smith]] or [[#Person]].
  - When the parking space control was used with the keyboard, it could display two superimposed copies of the link confirmation popover. This made keyboard cancellation problematic.
- **Miscellaneous:**
  - Tinderbox 9.5 previews sometimes ended too soon, because Tinderbox incorrectly measured the length of the UTF8 string passed to the Preview pane.
  - The built-in Preview template was incorrectly being removed from the Built-In Templates submenu of the File menu.
  - Explode was confused if the explode action moved notes to a new container. Tinderbox now defers performing the action explode has created all its notes, and then perform the action on each note in turn.
  - Avoided a crash when the currently-selected note has its link deleted by an action. The link animator must be notified in advance, as it is left holding a reference to animate the deleted link.
  - Tagging at startup was disabled for documents >1000 notes as a performance optimisation. This may no longer be necessary. Tagging at startup should now be more reliable in all documents.
  - $WordCount now works better in Chinese and several other languages.
  - Resolved a potential crash because the stamps menu could be rebuilt while stamps were being used by actions.
  - Adding Built-In Hints, either directly or by using the Preview pane, deleted existing stamps.
  - Recursive word counts — used to measure the word count of a note and its descendants — are now significantly faster.
  - Resolved a problem in parsing comments that included quotation marks, apostrophes, semicolons, or braces.
  - Resolved a hang that could arise when accessing the $Path of a note in different queues.
- **OS Dark/Light modes:**
  - Improved the dark-mode drawing of the composite name widget, and of monochrome badges against dark backgrounds.
- **Text pane:**
  - Tinderbox text highlighting after Find now respects case-sensitive search.
  - When a note's prototype is set or changed, the note copies the prototype's text if the note has none, or if the text it inherited from its former prototype has not been changed. If the prototype has text links, the note also acquires text links.

See v9.5.2 Release Notes for more detail of minor points.

This version is cited in the following notes:

- Agents & Rules tab
- Attribute name styles in listings
- attributes tab
- Colors
- Colour Swatches
- Dictionary.extend(itemDict)
- Displayed Attributes table
- Dragging in Apple Calendar events
- Flags in outlines
- HTMLSubscriptEnd
- HTMLSubscriptStart
- HTMLSuperscriptEnd
- HTMLSuperscriptStart
- Importing Markdown files
- List/Set.randomItem()
- New note name parsing for prototypes and locations
- Note Flags
- Number.ceil()
- Number.floor()
- Number.round()
- Outline tab
- OutlineColorSwatch
- Participants
- Pattern: bar(30)
- Pattern: vbar(70)
- Prototypes
- show(msgString[, backgroundColor[,colorString]])
- String.show([backgroundColor[,colorString]])
- String.wordCount()
- System tab
- User tab
- values([scope,]attributeNameStr)
- View pane (note selected), pop-up menu
- WordCount

v9.5.1b598 (13 Dec 2022)

Released 13 Dec 2022. Build number is 598.

Minor changes not necessarily warranting explicit mention in an aTbRef note:

- **Text pane:**
  - Preview Pane. Previews sometimes ended too soon, because Tinderbox incorrectly measured the length of the UTF8 string passed to the Preview pane.

See v9.5.1 Release Notes for more detail of minor points.

v9.5.0b597 (9 Dec 2022)

Released 9 Dec 2022. Build number is 597.

Eastgate release page for this version is here.

Minor changes not necessarily warranting explicit mention in an aTbRef note:

- **Action Code:**
  - Changing the link type in 'Browse Links…' now correctly (re-)fires the OnLink action.
  - Converting a string to a dictionary yielded incorrect results if the value contained a colon: `$MyDictionary="1:able:baker"; // → 1:able.`
  - create() was too aggressive in evaluating its arguments, which caused trouble when note names contained apparent operators such as [ ]. Fix applies to createAgent() and createAdornment() too.
  - When retrieving values from a var variable declared to be a set, Tinderbox returned a list rather than a set.
  - Corrected: previously, the .paragraphList() operator was erroneously converting its results to lower case.
  - The stamp menu, and the stamp list maintained by the Document inspector's Stamp pane, are now updated when changes are made to the stamp notes in the Hints container.
  - In code fields, eachLink(){…} is highlighted correctly, as is each(){…}.
  - Corrected a parsing error when evaluating the value of an if() statement — a common idiom in $DisplayExpression: if($MyString) {$MyString} else {$Name;}
- **Agents:**
  - Even the lowest-priority "Occasional" agents are now run when the document is opened.
- **Attribute Browser view:**
  - When switching a tab to use attribute browser from some other view, that tab did not always adopt the correct container if, in the past, the tab had been used for an attribute browser on a different container.
- **Chart view:**
  - The Chart popover controls have a more sensible number of tick marks.
- **Export:**
  - The progress bar is once again displayed during HTML exports.
  - Resolved interference during *Text export* between a sheet asking for details of the export and a dialogue asking where to save the exported file.
  - The export sheet offers export to "Word®" rather than "doc".
  - Notes exporting with the .md extension may now be previewed even if not using an HTMLPreviewCommand.
  - Tinderbox now sets $HTMLExportTemplate correctly when adopting the Preview template. In past test releases, the attribute was set to "/Templates/preview"; the correct value is "/Templates/Preview".
- **Get Info:**
  - The Agent pane of the Get Info popover window now performs syntax colouring in the query and action fields.
  - The layout of the Get Info: Attributes pane is improved.
  - The count of notes found by an agent is now updated promptly in the Agent pane of Get Info. Formerly, the note count was updated immediately while the agent update was performed on a separate, slower thread. We now update the field after the agent has finished its work.
  - The highlight colour for words in context in Get Info: Repetitive is chosen more prudently in dark mode.
  - The export file manager once again removes all punctuation from file names. The experiment to allow a wider use of URL-safe punctuation characters proved problematic with some otherwise 'safe' characters, e.g. characters allowed in macOS filenames.
- **Help:**
  - Updated Help so link to Recent Changes links to the release notes index, not the 9.0.0 page.
  - CompressImages() is now performed after edits are are made, rather than doing this for each note when the document is saved. The latter was slow on large documents.
- **Import:**
  - Text file import is now more tolerant of text files with unusual text encodings.
  - Files with the extension ".json" may now be dragged into the view pane. They are treated as text.
- **Inspector:**
  - The User Attribute Inspector formerly reordered the selected note's displayed attributes if a new user attribute was added to the displayed attributes list. The order is now preserved.
- **Links:**
  - When links were added to the text pane while the link pane was hidden, and then the link pane was displayed before changing the selected note, these newly-added links might not be displayed in the link lists. That should no longer happen.
  - Copying or Cutting text in an alias could cause confusion in the placement of text links.
  - Addressed a potential deadlock when undoing link deletion.
- **Map view:**
  - Duplicating a note now positions the duplicate more predictably, while avoiding existing notes.
  - In map view, zooming out from a container does a better job of restoring the scroll position of the map.
  - The Map Background image is no longer upside down.
  - Geographic Adornments are updated more promptly when their location changes.
  - Using ⌥-Tab to focus on the view pane now triggers then focus animation, as expected.
  - Double-clicking a container in map view could crash if pending processes such as hover timers intervened during the animation.
  - If using @place or #prototype notation at the top level creates the prototypes container, is is now placed above the current note to leave space for the place note.
- **Menus:**
  - The View ▸ Tabs ▸ Close Tab menu is once more disabled if there is only one remaining tab.
  - The stamp menu, and the stamp list maintained by the Document inspector's Stamp pane, are now updated when changes are made to the stamp notes in the Hints container.
  - File menu ⌘⌥^A Deselect all now works in text pane as well as view pane.
- **Miscellaneous:**
  - Fix to help with delays and apparent hangs when activating Tinderbox from the background, when using extremely large documents with lots of text.
  - Resolved a memory leak in action and expression tokenisers.
  - The highlight colour for words in context in Get Info: Repetitive is chosen more prudently in dark mode.
  - Tinderbox no longer creates thumbnail images of the map text if theres not enough space in the map item for the text to be drawn. This saves some battery power.
  - Memory leaks in processes ExportPathAttribute, URLAttribute, and HTMLMarkup have been corrected, removing problems for heavy use of the Preview pane in notes where rules refresh the screen at frequent intervals.
  - String/number coercion. Now, a string is regarded as a numeric value only if the entire string can be parsed as a number. It is preferable, of course, to use intermediate attributes or typed vars in order to avoid the ambiguity entirely.
  - Improved the text position in the breadcrumb bar.
  - Using ⌥-Tab to focus on the view pane now triggers the focus animation, as expected.
  - Fixed a perhaps-possible crash when double-clicking a note in map view.
  - Occasionally, crashes arise when pending changes are executed while the view is being overhauled. Major overhauls now clear the change manager.
  - The customised stop list in /Hints/stoplist was not parsed correctly, and was ignored by the Common Words pane in Get Info.
  - A call to CompressImages was inadvertently omitted in recent releases, leading to excess disk usage for images embedded in text.
- **Tabs:**
  - Improved the text position in the breadcrumb bar
- **Text pane:**
  - Edit ▸ Deselect All now operates in the text pane.
  - Many revisions to editing the text of multiple notes, especially when some of the notes are empty.
  - Format ▸ Text ▸ List… misled Tinderbox into thinking that a single note's text was actually a multi-note text. This should no longer occur.
  - When opening a document in which the preview pane is the current pane, the preview pane shader formerly claimed that "no note is selected" even when a note was, in fact, selected.
  - Copying or Cutting text in an alias could cause confusion in the placement of text links.
  - The text pane is more scrupulous in checking that every edit has been recorded before changing the selected notes.
  - Printing is now possible from the preview pane as well as the other Text panes.
  - The Preview Pane is more likely to receive the focus for printing when wanted. If in doubt, switch to Text and back to Preview to ensure the focus is on Preview pane.
  - Pasting an image into the text pane could sometimes lead Tinderbox to omit text preceding the image when the change was recorded.

See v9.5.0 Release Notes for more detail of minor points.

This version is cited in the following notes:

Previous Versions

Below are the dates, where known, of 159 versions up to the current baseline (v9.5.0). Details of per-version changes can be researched via the Change Log and the app's Release Notes.

Previous Tinderbox releases:

- v9.6.0 b (1 Aug 2023) (Eastgate release page)
- v9.5.2 b606 (28 Feb 2023)
- v9.5.1 b598 (13 Dec 2022)
- v9.5.0 b597 (9 Dec 2022) (Eastgate release page)
- v9.3.0 b566 (4 Aug 2022)
- v9.2.1 b556 (14 Apr 2022)
- v9.2.0 b553 (2 Mar 2022)
- v9.1.0 b542 (10 Dec2021) (Eastgate release page)
- v9.0.0 b523 (7 Jul 2021) (Eastgate release page)
- v8.9.2 b496 (11Feb 2021)
- v8.9.1 b486 (12 Jan 2021)
- v8.9.0 b485 (30 Nov 2020)
- v8.8.0 b479 (6 Oct 2020) (Eastgate release page)
- v8.7.1 b467 (22 Jun 2020)
- v8.7.0 b464 (21 May 2020) (Eastgate release page)
- v8.6.2 b452 (14 Mar 2020)
- v8.6.1 b451 (13 Mar 2020)
- v8.6.0 b448 (11 Mar 2020)
- v8.5.1 b437 (26 Feb 2020)
- v8.5.0 b434 (20 Feb 2020) (Eastgate release page)
- v8.2.3 b426 (21 Jan 2020)
- v8.2.2 b421 (28 Dec 2019)
- v8.2.1 b416 (26 Nov 2019)
- v8.1.1 b410 (31 Oct 2019)
- v8.1.0 b405 (3 Jul 2019)
- v8.0.6 b384 (3 Jul 2019)
- v8.0.5 b383 (2 Jul 2019)
- v8.0.4 b382 (26 Jun 2019)
- v8.0.3 b374 (31 May 2019)
- v8.0.2 b373 (29 May 2019)
- v8.0.1 b366 (8 May 2019)
- v8.0.0 b361 (12 Apr 2019) (Eastgate release page)
- v7.5.6 b335 (19 Sep 2018)
- v7.5.5 b333 (12 Sep 2018)
- v7.5.4 b328 (30 Jun 2018)
- v7.5.3 b325 (22 Jun 2018)
- v7.5.2 b320 (8 Jun 2018)
- v7.5.1 b318 (4 Jun 2018)
- v7.5.0 b316 (30 May 2018) (Eastgate release page)
- v7.3.1 b292 (22 Nov 2017)
- v7.3.0 b289 (7 Nov 2017) (Eastgate release page)
- v7.2.2 b277 (7 Sep 2017)
- v7.2.1 b275 (3 Sep 2017)
- v7.2.0 b274 (31 Aug 2017) (Eastgate release page)
- v7.1.0 b263 (9 Jun 2017)
- v7.0.3 b256 (3 Apr 2017)
- v7.0.2 b255 (31 Mar 2017)
- v7.0.1 b243 (24 Feb 2017)
- v7.0.0 b242 (22 Feb 2017) (Eastgate release page)
- v6.6.5 b216 (14 Sep 16) (Eastgate release page)
- v6.6.4 b215b (28 Jul 16) (Eastgate release page)
- v6.6.3 b215a (25 Jul 16) (Eastgate release page)
- v6.6.2 b215 (21 Jul 16) (Eastgate release page)
- v6.6.1 b209 (6 Jun16)
- v6.6.0 b207 (23 May 16) (Eastgate release page)
- v6.5.0 b198 (22 Mar 16) (Eastgate release page)
- v6.4.0 b177 (23 Nov 15) (Eastgate release page)
- v6.3.2 b174 (22 Oct 15) (Eastgate release page)
- v6.3.1 b162 (27 Jul 15) (Eastgate release page)
- v6.3.0 b153 (1 Jun 15) (Eastgate release page)
- v6.2.1 b141 (25 Apr 15)
- v6.2.0 b137 (20 Apr 15) (Eastgate release page)
- v6.1.3 b121 (21 Jan 15)
- v6.1.2 b120 (20 Jan 15)
- v6.1.1 b114 (13 Nov 14)
- v6.1.0 b104 (2 Oct 14) (Eastgate release page)
- v6.0.4 b93 (28 Aug 14)
- v6.0.3 b92 (25 Aug 14)
- v6.0.2 b86 (8 Jul 14)
- v6.0.1 b82 (18 Jun 14)
- v6.0.0 b72 (29 May 14) (Eastgate release page) NOTE: app codebase re-written, ported to Xcode
- v5.12.3 (1 Sep 15) (realigned pre-post v6 licences)
- v5.12.2 (31 Aug 13)
- v5.12.1 (23 Jan 13)
- v5.12.0 (13 Dec 12)
- v5.11.2 (14 Jun 12)
- v5.11.1 (16 May 12)
- v5.11.0 (16 May 12) (Eastgate release page)
- v5.10.5 (24 Apr 12)
- v5.10.4 (12 Apr 12)
- v5.10.3 (11 Apr 12)
- v5.10.2 (15 Mar 12)
- v5.10.1 (6 Feb 12)
- v5.10.0 (20 Dec 11) (Eastgate release page)
- v5.9.3 (8 Aug 11)
- v5.9.2 (19 Jul 11)
- v5.9.1 (16 May 11)
- v5.9.0 (5 May 11) (Eastgate release page)
- v5.8.1 (1 Apr 11)
- v5.8.0 (4 Mar 11) (Eastgate release page)
- v5.7.1 (23 Nov 10)

- v5.7.0 (26 Oct 10) (Eastgate release page)
- v5.6.0 (15 Sep 10) (Eastgate release page)
- v5.5.4 (30 Jul 10)
- v5.5.3 (12 Jun 10)
- v5.5.2 (11 Jun 10)
- v5.5.1 (10 Jun 10)
- v5.5.0 (20 May 10) (Eastgate release page)
- v5.1.0 (31 Mar 10)
- v5.0.2 (12 Feb 10)
- v5.0.1 (30 Dec 09)
- v5.0.0 (10 Dec 09) aTbRef5 baseline (Eastgate release page)
- v4.7.1 (21 Jul 09)
- v4.7.0 (25 Jun 09) (Eastgate release page)
- v4.6.2 (9 Apr 09)
- v4.6.1 (17 Mar 09)
- v4.6.0 (4 Mar 09) aTbRef46 baseline (Eastgate release page)
- v4.5.3 (21 Oct 08)
- v4.5.2 (25 Sep 08)
- v4.5.1 (1 Sep 08)
- v4.5.0 (26 Aug 08) aTbRef45baseline (Eastgate release page)
- v4.2.5 (16 Jul 08)
- v4.2.4 (29 Apr 08)
- v4.2.3 (26 Mar 08)
- v4.2.2 (20 Mar 08)
- v4.2.1 (19 Mar 08)
- v4.2.0 (28 Feb 08)
- v4.1.0 (3 Feb 08) (Eastgate release page)
- v4.0.2 (14 Oct 07)
- v4.0.1 (4 Sep 07)
- v4.0.0 (1 Aug 07) (Eastgate release page)
- v3.6.2 (7 Mar 07)
- v3.6.1 (9 Jan 07)
- v3.6.0 (4 Jan 07) (Eastgate release page)
- v3.5.4 (10 Aug 06)
- v3.5.3 (9 Aug 06)
- v3.5.2 (18 Jul 06)
- v3.5.1 (17 Jul 06)
- v3.5.0 (27 Jun 06) (Eastgate release page)
- v3.0.6 (3 Apr 06)
- v3.0.5 (7 Mar 06)
- v3.0.4 (13 Jan 06)
- v3.0.3 (12 Jan 06)
- v3.0.2 (5 Dec 05)
- v3.0.1 (11 Nov 05)
- v3.0.0 (10 Nov 05) (Eastgate release page)
- v2.5.0 (30 Jun 05) (Eastgate release page)
- v2.4.1 (16 May 05)
- v2.4.0 (9 Mar 05) original aTbRef baseline (Eastgate release page)
- v2.3.4 (9 Oct 04)

Earlier version releases, before aTbRef author's use of Tinderbox. As there is no formal record of the actual release dates of these older versions, some dates have been estimated from available data:

- v2.3.3 (4 Oct 04)
- v2.3.2 (30 Sep 04)
- v2.3.1 (28 Sep 04)
- v2.3.0 (21 Sep 04) (Eastgate release page)
- v2.2.0 (26 Apr 04)
- v2.1.0 (16 Dec 03)
- v2.0.0 (1 Aug 03) (Eastgate release page)
- v1.2.4 (12 Feb 03)
- v1.2.3 (1 Feb 03)
- v1.2.2 (6 Oct 02)
- v1.2.1 (4 Oct 02)
- v1.2.0 (26 Sep 02)
- v1.1.4 (22 Jul 02)
- v1.1.3 (3 Jun 02)
- v1.1.2 (3 Jun 02)
- v1.1.1. (1 Jun 02)
- v1.1.0 (1 Jun 02)
- v1.0.2 (14 May 02)
- v1.0.1 (18 Apr 02)
- v1.0.0 (18 Feb 02) (Eastgate release page)

Footnotes:

1. during early development, the Tinderbox app was also known as 'Ceres'.
2. Ceres/Tinderbox design also drew on ideas developed for its elder sister Storyspace. The latter is a hypertext authoring/reading program and which has design routes back to the mid-1980s. Since Tinderbox v6 / Storyspace v3 the two applications use a common file format although amazingly Storyspace will still open Storyspace v1 format files.
3. Current build number in releases data from the complete new codebase developed for v6+. Up to v5.x development used MetroWerks's CodeWarrior and older frameworks and had limited Unicode support. For v6, the entire code was ported to Xcode and refactored/rebuilt to offer better use of Apple frameworks. A by-product was the loss of older Tinderbox's rich multi-window UI and the ability to drag links between discrete app windows. Needing to embrace the newer single app window design ethic of the time, other affordances have been added to replace those lose and the single window limitation has been relaxed (though inter-window drag remains impossible due to underlying framework limitations).

Version release vendor URLs

Note: not all versions had (or still have) a per-version vendor page).

Original app: https://www.eastgate.com/Ceres/ (N.B. Ceres was the original app name pre-release)

v2.0.0: https://www.eastgate.com/Tinderbox/updates/Tinderbox2.html
v2.3.0: https://www.eastgate.com/Tinderbox/updates/Tinderbox23.html
v2.4.0: https://www.eastgate.com/Tinderbox/updates/Tinderbox240.html
v2.5.0: https://www.eastgate.com/Tinderbox/updates/Tinderbox250.html
v3.0.0: https://www.eastgate.com/Tinderbox/updates/Tinderbox300.html
v3.5.0: https://www.eastgate.com/Tinderbox/updates/Tinderbox350.html
v3.6.0: https://www.eastgate.com/Tinderbox/updates/Tinderbox36.html
v4.0.0: https://www.eastgate.com/Tinderbox/updates/Tinderbox400.html
v4.1.0: https://www.eastgate.com/Tinderbox/updates/Tinderbox410.html
v4.4.0: https://www.eastgate.com/Tinderbox/updates/Tinderbox450.html
v4.6.0: https://www.eastgate.com/Tinderbox/updates/Tinderbox46.html
v4.7.0: https://www.eastgate.com/Tinderbox/updates/Tinderbox47.html
v5.0.0: https://www.eastgate.com/Tinderbox/updates/Tinderbox5.html
v5.5.0: https://www.eastgate.com/Tinderbox/updates/Tinderbox55.html
v5.6.0: https://www.eastgate.com/Tinderbox/updates/Tinderbox56.html
v5.7.0: https://www.eastgate.com/Tinderbox/updates/Tinderbox57.html
v5.8.0: https://www.eastgate.com/Tinderbox/updates/Tinderbox58.html
v5.9.0: https://www.eastgate.com/Tinderbox/updates/Tinderbox59.html
v5.10.0: https://www.eastgate.com/Tinderbox/updates/Tinderbox510.html
v5.11.0: https://www.eastgate.com/Tinderbox/updates/Tinderbox511.html
v6.0.0: https://www.eastgate.com/Tinderbox/updates/Tinderbox600.html
v6.1.0: https://www.eastgate.com/Tinderbox/updates/Tinderbox611.html (v6.1)
v6.2.0: https://www.eastgate.com/Tinderbox/updates/Tinderbox62.html
v6.3.0: https://www.eastgate.com/Tinderbox/updates/Tinderbox63.html
v6.3.1: https://www.eastgate.com/Tinderbox/updates/Tinderbox631.html
v6.3.2: https://www.eastgate.com/Tinderbox/updates/Tinderbox632.html
v6.4.0: https://www.eastgate.com/Tinderbox/updates/Tinderbox64.html
v6.5.0: https://www.eastgate.com/Tinderbox/updates/Tinderbox65.html
v6.6.0: https://www.eastgate.com/Tinderbox/updates/Tinderbox66.html
v6.6.2: https://www.eastgate.com/Tinderbox/updates/Tinderbox662.html
v6.6.3: https://www.eastgate.com/Tinderbox/updates/Tinderbox663.html
v6.6.4: https://www.eastgate.com/Tinderbox/updates/Tinderbox664.html
v6.6.5: https://www.eastgate.com/Tinderbox/updates/Tinderbox665.html
v7.0.0: https://www.eastgate.com/Tinderbox/updates/Tinderbox7.html
v7.2.0: https://www.eastgate.com/Tinderbox/updates/Tinderbox72.html
v7.3.0: https://www.eastgate.com/Tinderbox/updates/Tinderbox73.html
v7.5.0: https://www.eastgate.com/Tinderbox/updates/Tinderbox75.html
v8.0.0: https://www.eastgate.com/Tinderbox/updates/Tinderbox8.html
v8.5.0: https://www.eastgate.com/Tinderbox/updates/Tinderbox85.html
v8.6.0: https://www.eastgate.com/Tinderbox/updates/Tinderbox86.html
v8.7.0: https://www.eastgate.com/Tinderbox/updates/Tinderbox87.html
v8.8.0: https://www.eastgate.com/Tinderbox/updates/Tinderbox88.html
v9.0.0: https://www.eastgate.com/Tinderbox/updates/Tinderbox9.html
v9.1.0: https://www.eastgate.com/Tinderbox/updates/Tinderbox91.html
v9.5.0: https://www.eastgate.com/Tinderbox/updates/Tinderbox95.html
v9.6.0: https://www.eastgate.com/Tinderbox/updates/Tinderbox96.html

Understanding the layout of aTbRef webpages

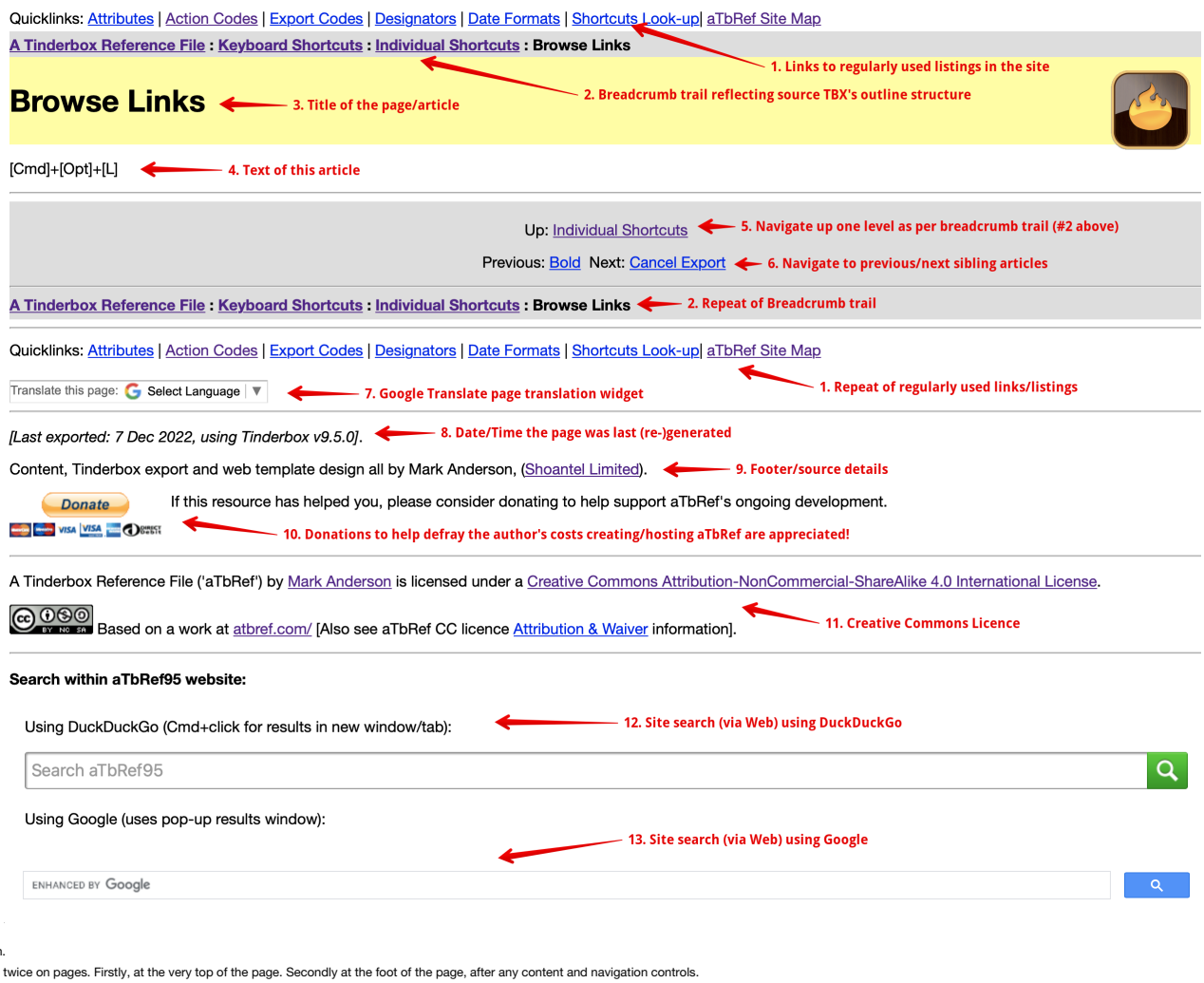
If new to aTbRef, it can be useful to become familiar with the standard elements of the page layout used by every page in the site (except the site map).

The yellow banner holds the note title and is styled, for those who go back that far, in the manner of older Mac OS app Help files.

Sections with a grey background are areas devoted to navigational links.

1. 'Quick' links

This is a set of links to allow you quickly jump from any page to some of the key listing pages on the site: list of system attributes, list of action code operators, list of export codes, list of designators and a list of date formatting codes. There is also a link to the overall site map. Lastly, the 'here' allows quick copying of the current page's URL (with rich text anchor text 'here'). The latter is useful for copying the location to use in email; it is also useful forgetting the current page's URL when working inside a chrome-less window as when using an SSB (Site-specific browser). The label 'quick links' has no relevance to use of similar terms in the body of aTbRef when discussing link creation.



This set of links occurs twice on pages. Firstly, at the very top of the page. Secondly at the foot of the page, after any content and navigation controls.

2. Breadcrumb trail

Tinderbox exports to HTML as a static website based on the source TBX file's outline view, as it reflected by the listing on the site map page. As nested pages are generally sub-topics of their parent page, the breadcrumb trails can help orie the reader. The trail it a list of colon-delimited page names starting at the site root page and going down to the current page. All trail items to the left of the current page are clickable navigation links to that page.

The breadcrumb trail is listed twice as indicated in the image. Once above the page title and once below the main navigation bar.

3. Title & app icon

This part shows the title of the current page and the app icon for the described version of Tinderbox.

4. The article's text

For reason's of compactness, the note illustrated here has very little text. Nonetheless, this part of the page is where the body text of the article is displayed. Articles may have an image (only one per article) and if present this is shown at the top of the page.

5. & 6. Navigation bar

The top level (#5) shows a link to the (parent)article above the current one (the same as seem in the bread crumb trail (#2 above) immediately to the left of the current note title. On the second row, there are links to the previous and next sibling articles (if such exist) under the same parent.

7. Translate the page in Google Translate

Use this widget to access a *machine-translated* version of the current page using Google's translate.google service. Being a machine translate, some aspects may be very poor, but this service is offered with best intent to help the many Tinderbox users whose first language is not English to gain more use from aTbRef.

8. Last Exported

This entry will indicate the last time this page was exported from its source Tinderbox document. A recent date may indicate a change, e.g. fixed typo. If changes occur reflecting new/changed features, the that fact is normally explicitly noted in the text, e.g. "From vX.x.x, there is a…". At each baseline change, i.e. change of major version number, any reference to the last version's subversion releases are removed. Thus, for v9, all in-text references to changes in v8.x are removed but changes to v9.cx.xill be added as they occur.

Do not worry if the version number seems newer than the current release, or has a number never publicly releasedTbRef's author is generally working in the current beta, thus the differing number from the current public release.

9. Footer - provenance information

This simply notes how this content was created and by whom.

10. Donation button

Change in circumstance means *voluntary* contributions towards supporting aTbREf are welcome: aTbRef remains free to use, for everyone. Any donations will go towards hosting aTbRef, time spent testing testing the app and writing the results up for aTbRef and into the apps needed to test their close inter-app integration with Tinderbox. The donations are taken via PayPal and the button links to a single-page process at the PayPal site. Again, importantly, the aTbRef site is free for all to use and donations are strictly voluntary.

11. Creative Commons Licence

This section states the CC-BY-NC-SA Creative Commons Licence under which this information is released (with Eastgate's, Inc.'s blessing), and described here. This is of little relevance to most users, but may prove useful in two circumstances. Firstly, if reusing or formally quoting the content the licence gives you a fairly clear aim as to what you may do without having to ask the to the author. The second is less common now Internet access is less of a novelty. Iy you wish to use aTbRef as a web resource but need to do so on an Intranet, the licence may assist with discussions with local IT staff over whether the resource can be thus used (it can!).

11. & 12. Web Search

Search inputs to use two different search engines are offered: DuckDuckGo and Google. Sadly trying to use major search engines to search changing web content is an exercise if frustration. The search engines index when they like and, despite following the instructions aimed to help their spidering, usually to sub-par effect. Any complaints about this should this be emailed to the search engine company and not to aTbref's author. Generally, the site map page is the best search resource there is. It should not be so, but complain to the Web Search companies, please!

13. Creative Commons Licence

This section states the CC-BY-NC-SA Creative Commons Licence under which this information is released (with Eastgate's, Inc.'s blessing), and described here. This is of little relevance to most users, but may prove useful in two circumstances. Firstly, if reusing or formally quoting the content the licence gives you a fairly clear aim as to what you may do without having to ask the to the author. The second is less common now Internet access is less of a novelty. Iy you wish to use aTbRef as a web resource but need to do so on an Intranet, the licence may assist with discussions with local IT staff over whether the resource can be thus used (it can!).

…and finally…

If still confused, please visit the Tinderbox user-to-user forum (N.B. that is not formal Tinderbox tech support) at https://forum.eastgate.com.