Macro Copier for TEXShop

Michael Sharpe
msharpe at ucsd dot edu

March 17, 2014

Briefly

TEXShop's Macro Editor allows you to import macros from a `.plist`, but is in some cases inconvenient because it imports all macros in the `.plist`, leaving you to delete unwanted duplicates, and gives you no easy way to compare the potential replacement with the installed version. This little program tries to make it a bit more convenient to maintain a macro `.plist` inside or outside of TEXShop.

- It gives you an easy way to selectively copy individual macros (or folders of macros) into or out of TEXShop's Macros Menu using a drag and drop interface. For most users, the main advantage will be that you can examine a macro from a large macro collection such as that distributed with TEXShop, compare it to what is already installed, and copy just the macros that are newer.
- If you also install the `plister` program from the free OnMyCommand package, instructions for which are provided later, you transform the program into a Macro manager of sorts—you can delete or rearrange individual macros or folders of macros, you can create new items, folders and separators, and you can edit the content window directly, assuming of course you have write permission for the `.plist` file. This provides the most convenient method for updating existing macros—use Compare to check for differences, and then to overwrite the installed version with the new one, copy the new one from its Content window with `cmd-A cmd-C` and paste it into the other Content window with `cmd-A cmd-V`.

After dropping one or more macros onto the TEXShop macro side, the Macros Menu should be refreshed from disk by restarting TEXShop. (It is in fact better to close TEXShop while running MacroCopier.) Then, open the TEXShop Macros Menu and delete older versions if necessary and rearrange the order to suit your needs.

Details

The interface window presents two lists, each of which is similar to what you see when you open the TEXShop Macro Editor. Let's assume that the `Macros_LaTeX` is selected from the Source popup menu on the left side. The left hand side always shows the macro files currently installed in TEXShop. The one that shows in TEXShop's `Macros` menu depends on the choice of typesetter for the frontmost document,

though currently (TeXShop 3.26), only the following are recognized in TeXShop's source code:
Bibtex
Context
Index
Latex
Metapost
Tex
with unrecognized typesetters defaulting to Latex. The right side initially shows what is available within the TeXShop application bundle. (This is what is installed by TeXShop on startup if it cannot find a file named `Macros_latex.plist` in the folder `~/Library/TeXShop/Macros`.) You may select a different source, the other options being:

`'New' in TeXShop:` This opens the first `plist` file it finds in the folder `~/Library/TeXShop/New/Macros`. This is where macros that are new or newly modified in this release of TeXShop are placed. Users who do not update very regularly may miss this and will need to import from the `.plist` contained in TeXShop's application bundle.

`TeXShop Macros:` This takes you to TeXShop's `Macros` folder:

`~/Library/TeXShop/Macros`

where you may select one of the `Macros_*.plist` files. It must be different from the one you have open on the left side.

`Empty plist:` You would use this if you wished to create a new `plist` for purposes of exporting some macros from your current collection.

`Other plist:` With this, you may select an arbitrary `plist` file to either export to or import from.

If you select the same macro name in each panel (case-sensitive), a new button Compare will appear just to the right of the Revert button, provided either BBEdit or TextWrangler are installed at the top level (ie, not in a subfolder) of your `Applications` folder. Pressing Compare runs (a version of) `diff` on the two macros. If `diff` declares they are identical, a message box appears to tell you so, otherwise both macros are opened in BBEdit/TextWrangler. (The one on the left is the one currently installed in TeXShop's Macros Menu.) It also opens a third window at the bottom showing the differences—click on a difference line to check whether it is significant. To exit, close the bottom (differences) window. This information may help you decide whether to replace the installed version with the one from the external `.plist`.

When you drag a macro or a folder of macros from one side to the other, a corresponding item is created at the end of the target list. Unless you install `plister`, this program does not allow you to rearrange parts of a `plist`. (The TeXShop Macro Editor can be used for that purpose.) The `.plist` file is modified immediately following the drop, and there is no Undo feature for individual drops. (This should not be as scary as it sounds—in this drag and drop mode, macros are never deleted, only added to the current target.) The Revert button applies only to your currently installed macros—it restores them to their state at the time the list was last loaded from disk. (If the program should crash, you can do this manually—the original macros are saved in the folder `~/Library/TeXShop/Macros` under the name `Macros_latex_orig.plist`, which should be renamed to `Macros_latex.plist` before restarting TeXShop.)

Installing `plister`

OnMyCommand is a free-of-cost package that contains a number of standalone command-line programs, including `plister`, which provides a very nice way to construct or modify a `.plist` file using only the command line. The package was originally intended to allow the user to construct contextual menu item plugins that would run shell scripts or AppleScripts, but such usage has now been abandoned by Apple in favor of Services. Version 3.0 (August 19, 2012) may be downloaded from `http://free.abracode.com/cmworkshop/on_my_command.html`. Don't bother installing OnMyCommand—it offers little else of any current value. If you want to do everything yourself, unzip `OMC.zip` to `OMC 3.0.pkg`, open the package contents and copy `Archive.pax.gz` to the Desktop, expand it and look for `plister` in its folder

`Library/Frameworks/Abracode.framework/Versions/A/Support`

Copy it to `/Library/Application Support/MacroCopier`, creating that folder if necessary. MacroCopier is hard-coded to look there for `plister`. If it doesn't find it there, it will ask if you want to install it, and carry out all these steps for you. If you don't install `plister`, you will not be able to delete or copy content directly within MacroCopier and will be limited to just copying selected macros to another `.plist`, leaving further manipulations to TEXShop's Macro Editor.

Once you install `plister`, you will see further options.

- If you have write permission to the source file (always true for the left panel) and you select a number of items, you will see a Delete selected button appear. If you press it, all selected items will be removed from the source file and the file will be re-loaded in its new form.
- If the selected item is not a folder and you make changes to its `Content` or `Name`, you will see a Save button appear. Pressing that will replace the content of the selected macro with the modified content. This is extremely handy for modifying macros *in place* from newer versions. You may change the Name of both macros and folders in much the same way.
- Macros and folders of macros may be reordered by dragging them to new positions in the same list. Because of limitations of the software used to write this package and the inability of the programmer to work around those limitations, reordering is limited in scope and is less capable than TEXShop's Macro Editor. Specifically:
  - You can move only one item (macro or folder) at a time.
  - You must open a folder before moving an item into that folder.
  - To move an item into a folder as the last item, open the folder and move the item just past the last existing item.
  - There is one case that requires two steps: if an item is located immediately following an open folder, you can't move it up to be the last item in the folder—you have to move it somewhere else and then move it to be the last item of the open folder.
- You will see a Create button appear if you have write permission to the associated file. Press this button to make a new entry that is either a new blank macro, a new folder or a new separator.

Editing external macro lists with TEXShop's Macro Editor

If you don't install `plister`, you'll have to rely on other plist editors. Say you've exported a number of macros to an external `.plist` and now you want to reorganize them. Though you can use a number of (free) plist editors such as Apple's `Property List Editor` that used to be available with `XCode`, or `Pref Setter`, available online at `http://www.nightproductions.net/prefsetter.html`, I find these tools harder to use than TEXShop's `Macro Editor`. Here's how to use that on an external `.plist`. Open `Macro Editor` and insert a Separator at the bottom of your current macro list. Then Save and use `MacroCopier` to copy all your macros from the external `.plist`. They will be inserted below the Separator. You may now reorganize, delete, add separators and submenus to the copied macros. Be sure to keep them below the Separator you added. When finished editing, press Save, reopen the Macro Editor and select all the files you wish to export, then select `Save selection to file...` from the Macro Editor (it won't be visible unless you have saved your changes) to a new `.plist` so that your original `.plist` is available should you change your mind later. Finally, Delete all the items below the separator you added and Save again so they don't appear in your Macros Menu.

Understanding how `plister` operates

The documentation for `plister` is sparse, to say the least, and some of it requires further experimentation to make sense. What follows is my notes trying to follow how `plister` writes to a `.plist`. (The commands that display the content of a file are not relevant to my concerns.)

The `.plist` files used within TEXShop are of simpler structure than the most general `.plist` files, and I'll concentrate on those rather than striving for full generality.

An empty `.plist` file looks like this:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN" "http://www.apple.com/DTDs/PropertyList-1.0.dtd
<plist version="1.0">
<dict/>
</plist>
```

It can be created as the file `e.plist` with the command

```
plister set dict e.plist /
```

The part between the `<plist...>` and `</plist>` is where all data is stored, and the top level `<dict/>` (shorthand for `<dict></dict>`) is the top level container for the data. There are two types of container structures, one called a `dict`, in which each entry has a key and a value, and the other an `array`, in which items are identified by their index within the array, starting at 0. It is common for these structures to alternate, with a `dict` at the top level and an `array` following each submenu item, having dictionaries as its elements, etc.

While the file above is a legal `.plist` it may provoke errors from some plist readers, and the practical minimum is

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN" "http://www.apple.com/DTDs/PropertyList-1.0.dtd
<plist version="1.0">
```

```
<dict>
    <key>name</key>
    <string>ROOT</string>
    <key>submenu</key>
    <array/>
</dict>
</plist>
```

This was created with

```
plister set dict e.plist /
plister insert "name" string "ROOT" e.plist / # add first key-value pair
plister insert "submenu" array  e.plist /
```

It adds `ROOT` as the name of the root of the `plist` and then declares a submenu key followed by an empty `array` container. The data you see represented in TeXShop's Macros menu comes from the elements of that last (ROOT) array—all the surrounding items are just part of the XML structure to be parsed via the plist dtd. The elements of the ROOT array are usually dicts containing specific elements. Eg, it we expanded the ROOT array as follows,

```
<array>
    <dict>
        <key>content</key>
        <string>xxx</string>
        <key>name</key>
        <string>Insert xxx</string>
    </dict>
</array>
```

that would give us an entry `Insert xxx` on the Macros menu whose effect is to write `xxx` at the current location in the text. Entries that create submenus (ie, "folders") follow a different pattern:

```
<array>
    <dict>
        <key>name</key>
        <string>Submenu Title</string>
        <key>submenu</key>
        <array/>
    </dict>
</array>
```

with the `<array/>` to be expanded to list the items in the submenu. Referring to a "node" in a `plist` means specifying the dict or array in an unambiguous way, and one way to do that it by assigning each a `pseudopath`, as if they were in a file system. So, let `/` denotes the top level, one step higher than ROOT, and the ROOT array would be `/submenu`. The first dict in that ROOT array would be `/submenu/0`, and if that node had a submenu element, that "folder" would be `/submenu/0/submenu`. Finally, when we have a pseudopath to a terminal dict node (one with no further submenus), we add the key to specify the leaf items—eg, `/submenu/0/content` or `/submenu/0/name` would allow us access to the `<string>` data associated with those keys.

The general structure of most `plister` command lines takes the form

`plister <command> <params> <path/to/plist> <pseudopath/in/plist>`

in which:

- <command> is one of `set`, `remove|delete`, `add|append`, `insert`.
  - `set` is intended for replacing existing data, and can be dangerous because it can overwrite data unexpectedly. You can use it to create a new dict in an empty array, but in a non-empty array, it will change the last item. The best use is to change leaf data where you can give a pseudopath that precisely specifies the key and use `set` to change the value. Eg,

    ```
    plister set string "Changed value" e.plist /submenu/0/name
    ```

    would change the value a key-value pair in the first item of the ROOT array having the form

    ```
    <key>name</key>
    <string>old value</string>
    ```

    to

    ```
    <key>name</key>
    <string>Changed value</string>
    ```

  - `insert` has two forms, depending on whether the pseudopath points to an array or a dict. With an array, you may use either

    ```
    insert k
    insert ""
    ```

    the first of which is used to insert an item into an array before the item with index `k`, and the second is used to insert an item at the end of the array. (I could not successfully use `append` for this purpose.) If the target is a dict, the form is like `insert "New key" string "New value"`, constructing a new key and string within the dict. Eg

    ```
    plister insert 0 dict e.plist /submenu
    plister insert "New key" string "New value" e.plist /submenu/0
    ```

    first creates a new dict at the beginning of the ROOT array and then inserts

    ```
    <key>New key</key>
    <string>New value</string>
    ```

    inside that new dict.

    The clause `insert k dict` works as expected even if array element `k` doesn't exist, provided element `k-1` exists. This is useful to know if you want to add an element after element `k-1`, even if it isn't the last element.

  - `append` seems to be broken, in that it doesn't work to create an array or a dict, though it does to create individual leaf values, but only in an array. Eg,

    ```
    plister append string "Key value" e.plist /submenu
    ```

    does work, but is not useful.

  - `delete|remove` removes a node and all its contents. Care must be taken when performing multiple deletes, since the indices of all nodes at a lower level (thinking as if this were a file listing) will change. Eg,

```
plister delete e.plist /submenu/0
```

removes the first item of the ROOT array and all its children.

- <params> are the parameters to the command. Here are some useful forms in connection with `.plist` files of the type we are interested in.
  - `dict`, `array`, `string` Both `dict` and `array` are containers and take no arguments. A `string` takes a double-quote delimited string as its argument.
- `copy` is a 'special directive' that can be used to transfer data from one file to another file or to the same file. A `copy` clause like

  ```
  copy e.plist /submenu/0/content
  ```

  can be used in place of another such as `string "XX"` in an `insert` statement. The result is similar to copying files from one directory to another. Eg,

  ```
  plister insert "" copy e1.plist /submenu/0 e.plist /submenu
  ```

  effectively copies the first dict in the ROOT array of `e1.plist` as the last dict in the ROOT array of `e.plist`.