MacTeX Design Philosophy vs TeXShop Design Philosophy

Richard Koch

I went to the Apple Developer Conference in May, 2000. Developers at this conference were supposed to receive the release version of OS X. In the keynote address, Steve Jobs announced that the new release would be renamed OS X Public Beta with a price reduced from $130 to a handling fee of $15. After the keynote, a knowledgable friend translated : "OS X has been delayed by a year."

As a sop to the audience, Apple held a software raffle during this conference, the only time I've heard of them doing so. Every developer got something, but it soon transpired that almost everybody got a schlocky piece of software on a CD, shrink wrapped against a flimsy piece of cardboard.

I was looking through this talk I agreed to give and it isn't very interesting. So I decided to give each attendee of the TUG conference a free piece of software.

The schlocky software Apple gave developers in 2000 was a forerunner of iTunes. This was before the iPod and all that. I, unfortunately, have nothing up my sleeve.

1 The Global PrefPane and the LocalTeX Pane

MacTeX installs a copy of TEX Live owned by root in `/usr/local/texlive`. It also installs a small data structure by Gerben Wierda and Jérôme Laurens in `/Library/TeX`, describing the distribution.These choices were somewhat controversial and I once gave a TUG talk about them. Now I'm happy.

Recall that each year's TEX Live distribution is in a folder named by date in `/usr/local/texlive`, so for instance TEX Live 2014 is in `/usr/local/texlive/2014`. This makes it possible to keep old distributions around, in case a new distribution breaks a crucial class file. We install a Preference Pane, shown below, for Apple's System Preferences, allowing users to switch between distributions. A switch changes all GUI apps to use the selected distribution and also changes the command line so command line programs use it.

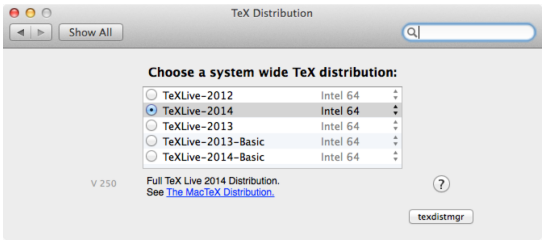

Figure 1: Global PrefPane

The PrefPane we install selects one distribution for all users and requires root access. I'm going to argue that we should have created a Local PrefPane instead, so each user could choose their own default TEX distribution and make this selection without root access. That's how *programs* work on the Macintosh. Programs live in `/Applications` where they are accessed by all users of a given machine. But each user has their own Preference settings for these applications, stored in `~/Library/Preferences`. One user's default Word font might be Times Roman, while another's might be Helvetica Neue.

The LocalTeX PrefPane shown on the next page is such a Pane. It can be installed locally for one user or globally for all users, but it makes independent choices for each user and does not require a password. This Pane does not change any link created by the Global Pref Pane or any element of the TeXDist structure, so it can be used together with the Global Pane, or when the Global Pane is completely missing.

The first item in the distribution list is always "Use Global Preference Pane." Selecting this item activates the Global Pane for the current

user. The next items are distributions with TeXDist data structures, so an individual user can select a different default than the one chosen by the Global Pane.

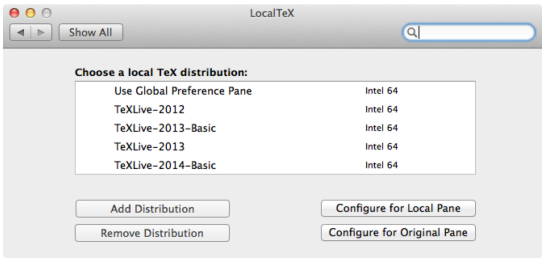

Figure 2: Local Pref Pane

Scrolling down in the list of distributions in the Pane, we see below that the LocalTeX pane can also define and select distributions on external disks, or distributions installed in a user's home directory. Although MacTeX cannot install TEX in such locations, the TEX Live install script from TUG can.

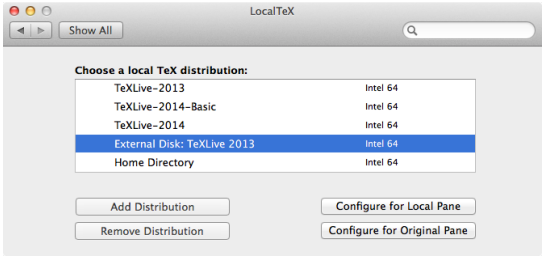

Figure 3: Local Pref Pane

Students may find this ability useful when they use a University owned machine and don't have root access. They can easily install TEX Live on a thumb drive, carry it with them, and have access to TEX in all locations.

The LocalTeX pane only shows distributions that are currently available. So if a thumb drive is removed, its distribution is no longer listed in the pane. Inserting the drive causes LocalTeX to list it again.

The "Add Distribution" button is used to inform the LocalTeX pane of TEX distributions without a TeXDist structure. It brings up a panel shown below. The "Name" field can be any desired name, since it will only appear in the LocalTeX pane. The "Path to Distribution" and "Path to Binaries" fields can be filled in by dragging appropriate locations to the dialog.

This data will only be accepted if the binary location is not empty, and contains a binary with at least one of the following names: tex, latex, pdftex, pdflatex, luatex, lualatex, xetex, xelatex.

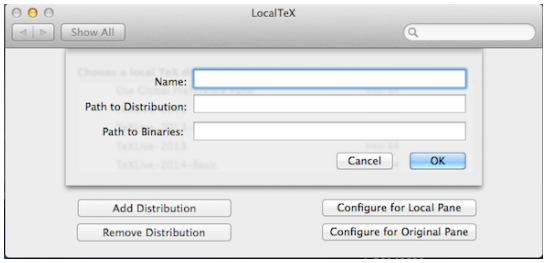

Figure 4: Local Pref Pane

The "Remove Distribution" button produces a list of extra distributions which can be removed one-by-one from those listed by the panel. Only distributions without a TeXDist data structure can be removed.

2 Installing and Configuring the LocalTeX Pane

The LocalTeX Pane can be obtained at `http://pages.uoregon.edu/koch/LocalTeX.zip`. Installing the LocalTeX pane is easy. Find and double click LocalTeX.prefPane. This brings up a dialog offering to install the Pane for all users or for only one user. Choose "only one user" and the Pane is installed for the current user without requiring a password. Or choose "all users" and the Pane is installed for everyone, but acts as a local pane for these users; installing this way requires a password.

After the Pane is installed, push the button "Configure for Local Pane" on the right. This

reconfigures TeXShop, TEX Live Utility, and BibDesk to use the new Pane. It also reconfigures shell for *some* users, namely users whose home directory contains none of the three hidden files bash_profile, bash_login, and profile.

To return to the Global Pane and stop using LocalTeX, push "Configure for Original Pane" to reconfigure TeXShop, TEX Live Utility, and BibDesk. Don't do this if you are merely choosing "Use Global Preference Pane" in the LocalPane.

3 How Does the Pane Work?

The LocalTeX pane creates three symbolic links in `~/Library/TeX/LocalTeX`:

- texbin → binary directory of default distribution
- texroot → folder containing the default distribution
- texdist → texdist structure for the default distribution, if such a structure exists

GUI applications should be configured to look for TEX binaries in `~/Library/TeX/LocalTeX/texbin` rather than in `/usr/texbin`, the corresponding link for the Global pane. This is done automatically by the "Configure for LocalPane" button for TeXShop, TEX Live Utility, and BibDesk. LaTeXiT has a rather baroque preference system which doesn't permit setting its presences using the "defaults" command line tool, but they can be reset by hand, as can the corresponding preference settings for other third party applications. Many of these applications require a full path rather than one containing a tilde.

4 Other Advantages

Wierda and Laurens carefully selected the location for the link `/usr/texbin`, arguing that Apple would probably not change or remove this link. That reasoning turned out to be wrong, and users who upgrade OS X often find that they can no longer typeset even though their TEX distribution remains, because the link has been removed. The location `~/Library` does not present this problem because third party programs use it and wholesale Apple changes would create a nightmare.

Creating Preference Panes with root access requires dealing with Apple's security framework and that tends to change over time. The Local Pane is immune to security concerns. It currently runs on Yosemite betas. It requires Mountain Lion and above, since it uses Apple's newer ARC memory protection scheme.

5 Configuring Terminal

To finish installation of LocalTeX, add `/Users/koch/Library/TeX/LocalTeX/texbin` to your PATH before the item `/usr/texbin`. Here and in the following paragraphs, replace "koch" with your own login name.

If you use some other shell than the default bash, you no doubt know what to do already.

Otherwise follow these instructions. By default, modern versions of OS X use "bash" as a shall. This shell reads bash_profile when it starts up. If this file does not exist, it reads bash_login, and if this file does not exist, it reads profile. All three are hidden files, whose names start with a period.

Many users have none of these files. In that case, pushing the "Configure for Local Pane" button created a .bash_profile file for you and there is nothing more to do.

Otherwise, you need to edit the appropriate file. In Terminal, type

```
cd
ls -a
```

to see a list of hidden files in your home directory. If you have .bash_profile, edit that. If not, but you have .bash_login, edit that. Otherwise edit .profile. Make the same edit in all

three cases. I'll discuss the case when you have .bash_profile. In Terminal, type

```
cd
mv .bash_profile bash_profile
```

Then you have a visible file to edit. Open this file with TeXShop. At the bottom, add the following two lines (the second and third lines below should be on a single line with no space between them).

```
# Added by LocalTeX Preference Pane
export PATH="/Users/koch/Library/TeX/
     LocalTeX/texbin":$PATH
```

and save the file. In Terminal type

```
cd
mv bash_profile .bash_profile
```

6 Removing Everything

If you install the LocalTeX Pane and decide that you don't want it, here is how to remove absolutely every trace from your computer.

- Using the Local Pane, push the "Configure for Original Pane" button to reconfigure TeXShop, TEX Live Utility, and BibDesk. If you reconfigured other apps, return them to their original configuration.
- Move LocalTeX.prefPane from `~/Library/PreferencePanes` to the trash.
- Move the folder LocalTeX from `~/Library/TeX` to the trash.
- If your shell was automatically configured for the new Pane, you will find a file named .bash_profile in your home directory containing the following lines. Throw the file in the trash.

  ```
  # Added by LocalTeX Preference Pane
  export PATH="/Users/koch/Library/
       TeX/LocalTeX/texbin":$PATH
  ```

  Otherwise you edited one of bash_profile, bash_login, or profile and added these lines. Remove them from the appropriate file.
- Finally, the LocalTeX Pref Pane stores its local data in the defaults system of OS X. To remove this data, type the following in Terminal (all three lines should be on a single line with spaces replacing line feeds):

  ```
  defaults remove
       com.apple.systempreferences
       localTeXExtrasData
  ```

7 LocalTeX and MacTeX

Will the LocalTeX preference pane be in a future edition of MacTeX? No. A choice between two Preference Panes would confuse most users, and while it is easy to configure the shell automatically for the global pane, this step requires user intervention for the local pane.

8 MacTeX Design Philosophy

From now on I'll give the promised talk. I work on the Macintosh in a small pond in the big TEX World. I wear two hats. I maintain MacTeX, the TEX install package for the Mac produced once a year by TUG. I also write, with collaborators, a GUI front end for TEX called TeXShop.

MacTeX is a "one button" package installing TEX , Ghostscript, and a few GUI applications. It presents a familiar interface for Mac users, asks no questions, and produces a completely configured installation. The installer was written by Jonathan Kew in an all night programming session at the North Carolina TUG Conference of 2005, and willed it to me at breakfast the next day. I was bleary eyed, but Jonathan was wide awake.

Jonathan's package installed a TEX distribution by Gerben Wierda, based on teTeX. But around this time, Thomas Esser abandoned teTeX and told his users to switch to TEX Live. Gerben produced a new distribution loosely based on TEX Live, which he announced at a TUG conference in Marrakesh in November of 2006. But at that same conference, he announced that he

would immediately end support for the new distribution. This left us in a quandary and for several months it was unclear which distribution we would install. I had been attending TUG meetings since 2001, and in all that time, Karl Berry never asked me "why don't you Mac folks use TEX Live?" I thought TEX Live was nerdy and hard to install until I tried it and found installation very easy. Since 2007, we install a complete version of TEX Live.

Hence a "Design Philosophy for MacTeX." MacTeX installs *a completely unmodified full version of TEX Live on the Mac.* It is exactly the distribution used on Linux, Unix, and Windows (for those not using MikTeX). We refuse to reach into the distribution and make configuration changes. When someone complains "my Mac collaborators cannot typeset my code" we get to respond vigorously "Sir, it is YOUR fault because the Mac folks use standard TEX Live!"

Collaboration is common in research. Kunth worked very hard to make TEX produce the same results on all platforms. We have a responsibility to make TEX platform-independent. Open source forever!

(But a small voice: we are in Portland, Oregon, the home of Textures. Barry Smith rewrote the Pascal compiler for TEX , and then rewrote TEX to produce absolutely precise synchronization between source and output, and to support direct use of Macintosh fonts. His code was commercial, not open source. Textures users remember it with great passion. Every philosophy has a "yes, but ...")

9 TeXShop Design Philosophy

Surprisingly, TeXShop has a very different design philosophy than the MacTeX design philosophy. I'll argue that a GUI front end to TEX should rigorously follow the design standards of the particular platform it supports and should use the latest technology on that platform. This is difficult to achieve if the app supports many platforms.

To understand why, consider the following three messages from the TEX on OS X mailing list:

```
From: Warren Nagourney:
I am using TeXshop 2.47 on a retina MBP
and have noticed a slight tendency for
the letters in the preview window to be
slightly slanted from time to time.
The slant is enough to make the text
appear italicized, which is annoying.
```

```
From: Giovanni Dore:
I think that this is not a problem of
TeXShop. I use Skim and sometimes
I have the same problem.
```

```
From: Victor Ivrii:
Try to check if the same distortion
appears in TeXWorks and Adobe Reader:
TeXShop and Skim are PDFKit based,
while TW is poppler based and
AR has an Adobe engine.
```

All three messages are from knowledgable people active in the TEX on OS X list. As the third message states, TeXShop and Skim use Apple's PDFKit to display pdf files, while Adobe Acrobat Reader has its own pdf rendering code, and TeXWorks uses poppler to render pdf. And indeed, TeXShop and Skim have a display problem but Acrobat Reader and TeXWorks don't.

However, there is a missing ingredient here. The author of the original message has an Apple portable with a Retina Display. TeXShop and Skim support the Retina display because they were written with Apple's Cocoa language. Acrobat Reader and TeXworks don't support the Retina display, so Apple runs them in "magnify by two" mode. The real problem is a bug in Apple's Cocoa Retina code, subsequently fixed. The bug also goes away if you turn off Retina support in TeXShop and Skim.

If you select "Get Info" in the Finder with a program selected, you get a panel of information

about the program. Below is part of that panel for TeXShop on the top, and for Adobe Acrobat Reader on the bottom, displayed on a Retina machine.

The key difference is the option to open in Low Resolution mode. This is selectable in TeXShop but not in Reader. That means that TeXShop by default supports the Retina display, while Reader does not. In case of trouble, TeXShop can be converted to a mode in which it writes at normal resolution and the Mac magnifies by two, while Reader always runs in this magnify mode.

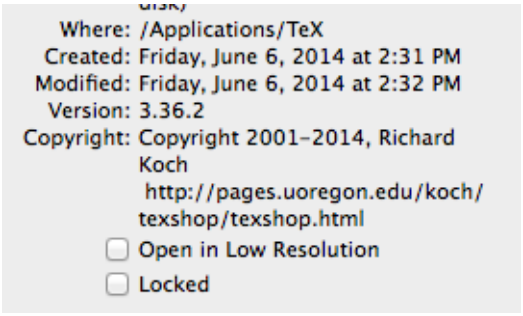

Figure 5: About TeXShop

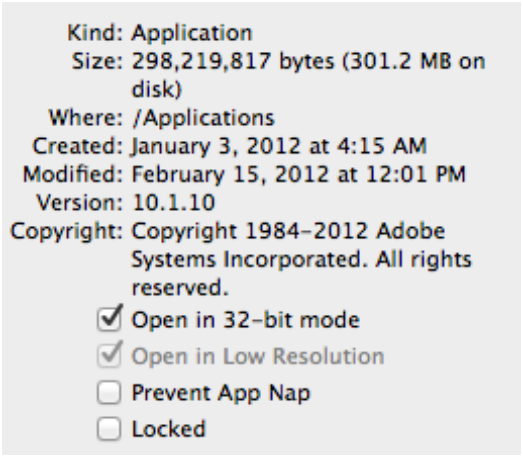

Figure 6: About Adobe Reader

The Retina Display Portable was introduced in June of 2012, but Adobe Reader and TeXWorks still don't support it two years later.

I had a very smart student who now works in the Portland software industry, so I boasted that TeXShop supported the Retina Display from the start. But he was too smart, and without skipping a beat he said "yeah, and how many lines of code did that take?" The answer is zero.

There are many ways to write GUI apps on the Mac: by supporting X11, by using Java, by using third party libraries, by using Carbon, and by using Cocoa. *If your app is written in Cocoa, then it automatically supports the Retina display. Otherwise not.*

10 NeXT at Apple, 1997 - 2007

Many of you read the book about Steve Jobs by Walter Isaacson. It is an interesting book, but has been criticized for getting the story of NeXT, and its role in Apple's second act, wrong. I agree, and here's a short version of that story from my perspective.

Apple bought NeXT in December of 1996, a sale that was finalized in February of 1997. Each May or June, Apple holds a Worldwide Developer Conference, WWDC. So in May of 1997, Apple had to give developers its strategy for using the NeXT operating system.

At the conference, Apple said that old Macintosh applications would continue to run in a sort of purgatory called the Blue Box, but new applications needed to be written in Objective C using NeXT's class library, then called OpenStep. Among commercial developers, the announcement went over like a lead balloon, and Apple got no significant endorsement at the conference. Apple's respected head of developer relations, Heidi Roizen, quit a few months later, calling the strategy "crazy."

So in 1998, Steve Jobs announced a completely different strategy. He called this new model

"Carbon" because, he said, "Carbon is the basis of all life." Carbon programs were written in C and C++ using the old Macintosh API, except that about 10% of the calls were replaced by new equivalents because the original calls wouldn't work on a modern multitasking operating system.This made it possible to start with an old Macintosh program, find the changed calls using an Apple-supplied script, revise them, and release the code on OS X. Apple immediately received endorsements from Microsoft, Adobe, Wolfram Research, and others, who stated that a whole new spirit of cooperation and realism was beginning to appear at Apple.

At this conference, OpenStep was renamed "Cocoa", but its standing at Apple was precarious. Some engineers said that new programs should be written in Cocoa, while others proclaimed vehemently that Cocoa was only for prototyping. At the 2000 developer conference I attended, the Carbon sessions were hald in the main auditorium packed with thousands of developers, while the Cocoa sessions were in a small church across the street, attended by 35 people who all seemed to know each other.

I attended WWDC regularly from 2003 to 2011, and this pattern continued for several years.

In 2005, Apple switched to Intel processors. At WWDC, they told developers that moving a Cocoa app to Intel usually involved a 10 minute recompile. Carbon apps, they estimated, could be moved in a month.

In 2006 the developer conference was postponed until August. At the conference, Apple gave developers a preliminary copy of Leopard, the next version of OS X, promising a release in March of 2007. A key feature of this release was full 64 bit support for all of Apple's important API's. Banners around the conference asked developers to become "64 bit ready" and a key slide of the keynote explained that "Leopard has full 64 bit support for Carbon and Cocoa."

But by June of 2007, Leopard was still not out. Why not? In January of that year, Apple announced the iPhone, and Apple engineers were pulled from the Leopard team to finish the software. But outside developers couldn't program the iPhone, so the 2007 conference was essentially a repeat of the 2006 version, with a keynote address using the same slides.

There was just one electric moment in 2007. Unfortunately, I completely missed its significance. When Jobs came to the slide promising "full 64 bit support for Carbon and Cocoa", the slide had been changed to read "full 64 bit support for Cocoa." Lots of developers noticed, and they mobbed Apple engineers during the lunch which followed the keynote. It rapidly became clear that Carbon was deprecated. Apple work on it had ceased.

So by 2007, Apple had the courage, and the prowess, to kill Carbon and throw their support totally behind Cocoa. Behind the scenes, they knew that both the iPhone and the as yet unannounced iPad could only be programmed in Cocoa. From 2008 on, there have been no Carbon sessions at WWDC. Commercial developers were among the last to switch to Cocoa, and some of their apps are still in Carbon.

During these turbulent times I was mostly oblivious to the drama. TeXShop remained a 32 bit app since I saw little reason to change.

But then TeXShop began crashing. I decided that the solution was to update to the latest Apple technologies. Shortly before Lion was announced, I moved TeXShop to 64 bits, and began planning to support garbage collection. What I didn't know was that dramatic changes were being made at Apple, and my 64 bit conversion was done just in the nick of time.

11 The Fragile Base Class Problem and 64 Bits

An *object* is a self-contained collection of code and date. Its data is referenced by variables

known as *instance variables* and its code defines a series of *methods* or *functions*. According to a common metaphor, an object oriented program contains many objects , which talk to each other through method calls, and act on these calls by processing the data in their instance variables. Cocoa programs are object oriented.

To see how this works in practice, consider the Cocoa object called *NSView*. Each NSView corresponds to a rectangular portion of a particular window. The view has an instance variable pointing to it's window, a second instance variable giving the coordinates of its rectangular region, and so forth. Among the methods defined for an NSView are drawRect, which draws the view on the screen.

When a developer uses NSView, the developer defines a *subclass* of the view with a name like *myNSView*. This subclass has all the instance variables and methods of NSView, plus other instance variables and methods added by the programmer. But in addition, it can override some of the original methods of NSView. For instance, the drawRect command in NSView doesn't draw anything, but myNSView could override drawRect so it draws the logo of this conference. In this situation, we call NSView the *Base Class*, defined in Cocoa, and we call myNSView *a subclass* defined by the programmer.

The advantage of all this is that base classes usually come already connected up. Cocoa calls drawRect when the window first appears, when a covering window is moved out of the way, when a dialog box goes away, etc. Apple once gave developers a teeshirt with the text "Don't call us; we'll call you." The slogan means that the programmer's myNSView doesn't have to worry about when to draw because Cocoa will tell it when to draw. It just has to draw the logo when called.

The takeaway is easy: a Cocoa program runs cooperatively, with some tasks handled by the base classes in Cocoa and other tasks handled by subclasses defined by the programmer. ∎

After object oriented programming appeared, programmers began to dream of a time when the system could be improved by just revising the base classes, without even recompiling the programs. You could install Mavericks, and suddenly say "wow, Word never did *that* before!"

Unfortunately, a barrier stood in the way of realizing this dream. The barrier was called "the fragile base class problem": *when revising base classes, you are not allowed to add extra instance variables or extra methods to the base class.* This was a problem in objective C, in C++, in Java, and elsewhere. The problem wasn't quite as bad in objective C as elsewhere, because it had been designed so extra methods in base classes are legal. But still: no extra instance variables.

When Apple added 64 bit libraries in the Leopard timeframe, they realized that they had a once in a lifetime opportunity to fix this problem. Since there were no existing 64 bit applications, every 64 app would have to be compiled from scratch. So they took the opportunity to make changes to objective C when run in 64 bits, including completely solving the fragile base class problem. If your Macintosh runs in 64 bits, then the dream of improving everything by revising the base classes can be realized.

Incidentally, they also made these changes in the iPhone even though it ran in 32 bits. So objective C on the iPhone, iPad, and 64 bit Mac applications is a different beast than objective C in 32 bit Mac applications.

After this change, Apple rapidly increased the hardware requirements of its operating systems. Snow Leopard required Intel processors, Lion required 64 bit processors, and Mountain Lion required machines running the kernel in 64 bits. After that the policy changed: Mavericks and Yosemite run on all machines that can run the previous systems. I believe that the reason for

these policies is not that 64 bit programs run faster, but instead that Apple can now use all the extra added properties of objective C, including adding instance variables and methods to base classes.

12 Lion

Lion is the first Apple system to make real this great dream of improving programs by revising the base classes. Programs written in 64 bits with Cocoa got crucial added functionality for free, essentially without a recompile.

One of the standard requests for TeXShop was that it remember window sizes and positions when quit, and restore these windows automatically when next restarted. To shame me into working on this, users told me of other GUI's for TEX which already had the ability.

Imagine my surprise, then, when I discovered that TeXShop on Lion got the requested ability automatically for free.

An advantage of letting Apple do this is that Apple had second thoughts and slightly modified the behavior in Mountain Lion and Mavericks. TeXShop inherited those changes for free. For instance, it is now possible to turn the feature on or off in Apple's System Preferences. If windows are generally saved when quitting, then holding down the option key changes the Quit menu to "Quit and Close All Windows." If windows are not generally saved, hold the option key while quitting to save the windows. Finally, push the shift key when opening a program if you don't want to open old windows. These tricks work in TeXShop and all other Cocoa applications.

13 Automatic Saving

Saving window positions is something I could have done myself if I weren't lazy. But the second Lion feature is something I would never have tried on my own: automatic file saving.

Suppose you are using TeXShop in Lion, you have several source files open and have made changes in each. Suddenly you receive an emergency call and quit TeXShop. You won't receive pesky dialogs asking you to save each file; instead TeXShop will immediately quit.

But the next time you open TeXShop, your edit changes will be in all the source files.

But wait — there's more. TeXShop doesn't just save when you quit. It saves every five minutes or so. If you live in a thunderstorm area with frequent power outages, no need to worry. When your computer starts up again, all that source you added will reappear.

"Gulp. Every five minutes the computer saves my 1000 page document?" Of course not. The program only saves changes, and in five minutes how much source did you change? In actual practice you never notice the saving process. There are no momentary glitches, no disk activity, and with a modern solid state drive no noise.

"Whoa. When I send a document to someone else, are all those changes in the document? My reference letter says 'works like a dog', but originally I wrote 'even a dog wouldn't be interested in his line of research.' " Not-to-worry, files only contain the latest version. That's done behind your back when you grab hold of a file to transfer it somewhere, and you won't notice.

"But there are so many edge cases where this scheme could go wrong." I absolutely agree. Indeed, I would never dare add automatic saving to TeXShop myself, or monkey around in any serious way with the file system. I dread getting a letter from a user claiming my program destroyed the only copy of his masterpiece. But this is Apple making the change, with a thousand engineers testing the code. Things slip by them, but destroying documents isn't something they'd take lightly.

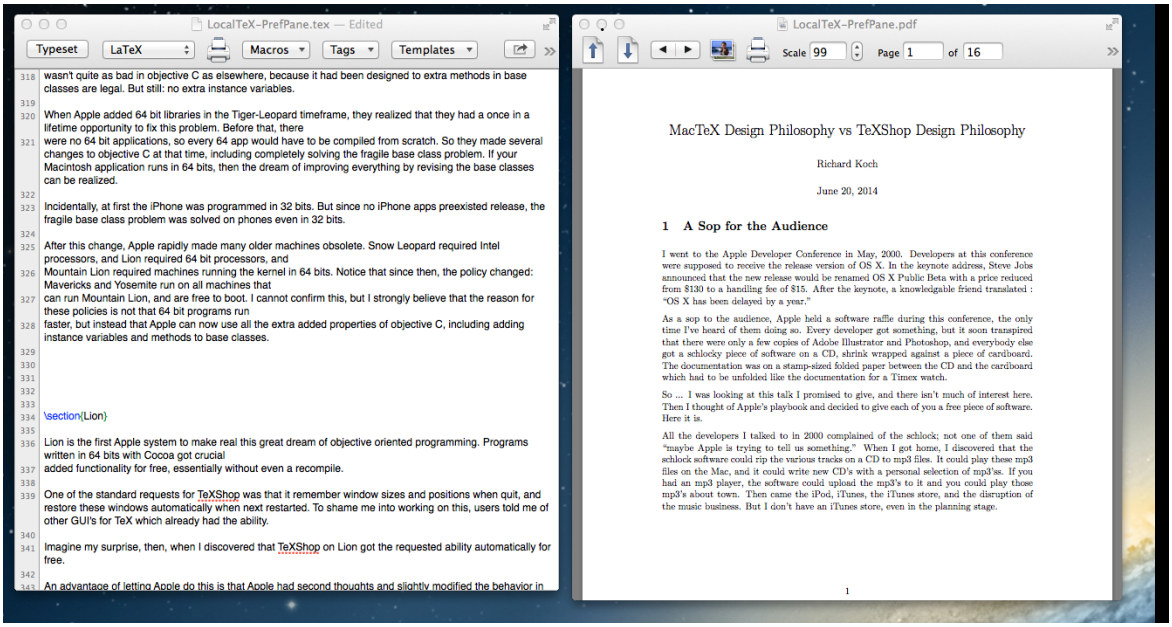

Figure 7: Edit

Incidentally, there is still a "Save" button in case you've solved a great problem and absolutely want to write your discovery to disk.

"But wait. Suppose I delete some material, type an experimental new sentence, and then decide not to keep it. In the old system, I just don't save. But with automatic saving, the new stuff I don't want may be part of the document. Terrible!"

No, it's not. The top picture on this page shows the document you are reading while it was being edited. The top picture on the next page shows the effect of selecting the menu Revert To → Browse All Versions.

As you see, this gives a Time Machine view of the document, and we can retreat to an earlier version, or copy a portion of an earlier version to the current document. Time Machine need not be running to get this. Any application with AutoSave activated gets it for free.

I'll confess that I don't use Time Machine because I don't like the sound of a Disk Drive. The new feature gives Time Machine for TEX documents.

Apple has been refining the interface for AutoSave. It is intrusive on Lion, less intrustive on Mountain Lion, and less still on Mavericks. I couldn't live without it. If your TEX GUI has it, then it works the same as your other Mac applications.

AutoSave makes many changes under the hood. One of the most surprising is changes to program menus. The most controversial is the loss of a "Save As..." menu. I received many email messages demanding that I put back this menu. I replied that it was still present in my code, and Apple removed it while running the program. My correspondents found this explanation incomprehensible.

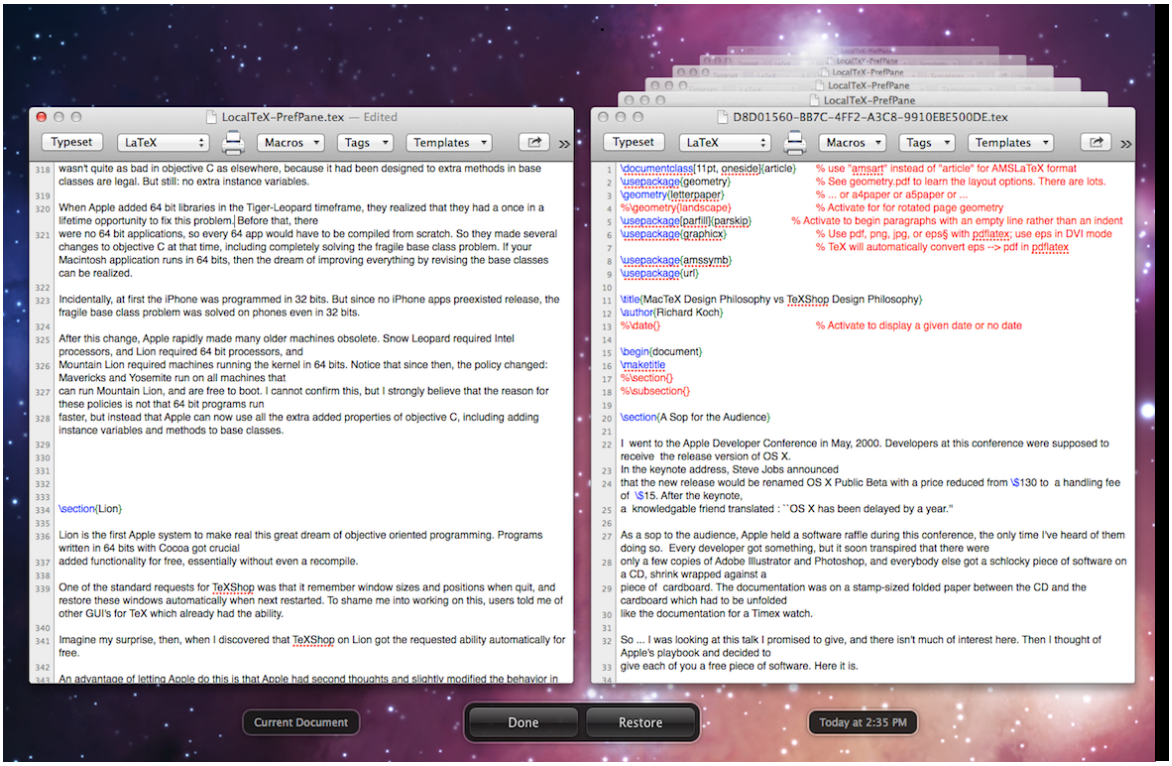

Figure 8: Browse All Versions

The truth is that Apple automatically modifies the program Save menu when AutoSave is turned on. This is shown on the following page. On the left is the File menu as defined in current TeXShop source code. On the right is the actual menu as displayed in Mavericks. As can be seen, the middle section of the menu has been dramatically altered.

After one email exchange on "Save As", I wrote what I thought was a brilliant defense of Apple's actions, telling my readers to "grow up and go with the flow." The next day another user pointed out that "Save As..." had been restored by Apple in Mountain Lion. Sure enough, if you hold down the option key when accessing the File menu, "Duplicate" changes to "Save As." Apparently the people on the mailing list were also writing Apple.

The main point I'm trying to make here is that for programmers who use Cocoa, the solution of the Fragile Base Class Problem allows Apple to make surprisingly many changes under the surface.

After all this, you probably want me to come clean. To implement Auto Save, how much code did I actually write?

Apple's NSDocument object contains a function called autoSavesInPlace. This routine returns NO. In TeXShop I override it to instead return YES. That's it. One line of code gives AutoSave for free.

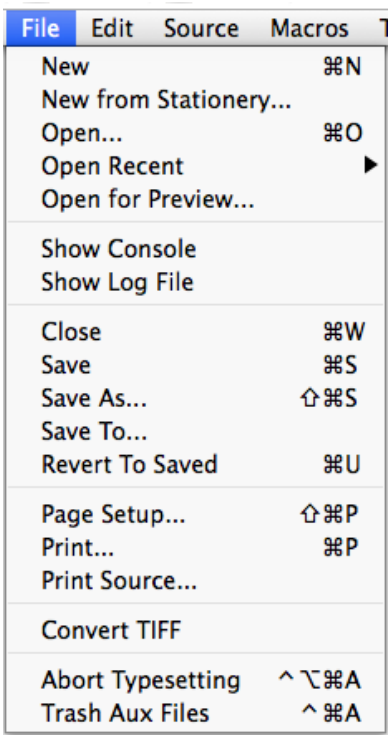

Figure 9: File Menu in Source Code

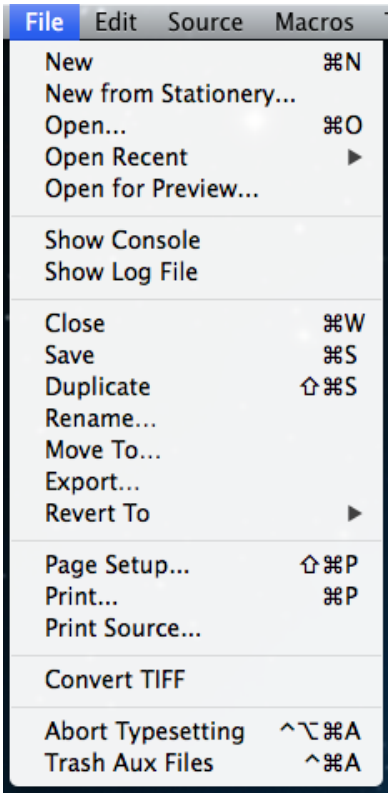

Figure 10: File Menu as Displayed by Mavericks

Incidentally, TeXShop's adoption of AutoSave has been popular. On the next page I show just one of the accolades I've received after releasing the Lion version.

Lots of collaborators help with TeXShop, providing features I haven't mentioned. Today I just wanted to show what is made possible by adhering to Apple's Cocoa standards.

TeXShop doesn't adopt everything, of course. It isn't in the Apple Store because working in a sandbox would limit its interaction with TEX Live and third party programs. It doesn't allow you to store documents in the Cloud because the Cloud is only available to applications in the store. But when an addition makes sense, it will be adopted.

So that's the end of my talk.....

14 Automatic Reference Counting

But I hear one of you shouting me down.

"I couldn't care less about the Retina Display or Preserving Window Locations when Quitting, and I Hate Auto Save. Back there five or six pages, you mentioned "crashes". Why don't you talk about TeXShop crashes?"

OK.

One problem with object oriented programming is that a program can create hundreds of different objects as it runs. The program is supposed to throw away objects after it is done with them; if it doesn't, then computer memory becomes clogged and the program becomes sluggish.

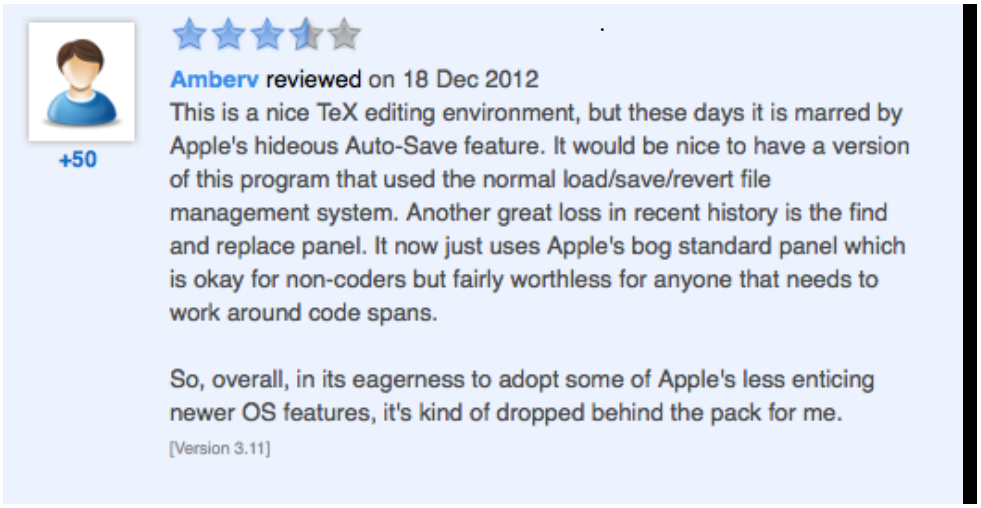

Figure 11: A Review

Objects can be passed around, so just because one part of the program is done with an object doesn't mean that it isn't used by someone else. If an object is thrown away too soon, the program will crash when another part of the program tries to use the object.

There are three solutions. The first is to force programmers to manually handle memory management. That is how TeXShop worked until recently, and it is prone to errors that are hard to find.

The second method is called "garbage collection." Apple introduced it in Leopard, but it didn't work well on the iPhone.

Then as part of the enhancement of objective C, Apple introduced Automatic Reference Counting, or ARC, the third memory management technique. In ARC, the compiler automatically adds the code to handle memory management, and the programmer can ignore it. Since ARC does what a programmer would do managing memory manually, some files in a program can be compiled with ARC and some can be compiled without it.

This spring, I spent several weeks recompiling TeXShop with ARC, gradually working through the program file by file. The ARC code first appeared in TeXShop 3.34 and makes the program much more stable. A couple of remaining issues are solved in TeXShop 3.38, released at this conference, and this version ends the transition to ARC.

Adding ARC is an example of extensive work with no immediate gain; no interface changes are visible. It is essential work if the program is to survive for the long run.