MacTeX Design Philosophy vs TeXShop Design Philosophy

Richard Koch

June 27, 2014

1 A Sop for the Audience

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 there were only a few copies of Adobe Illustrator and Photoshop, and everybody else got a schlocky piece of software on a CD, shrink wrapped against a piece of cardboard. The documentation was on a stamp-sized folded paper between the CD and the cardboard which had to be unfolded like the documentation for a Timex watch.

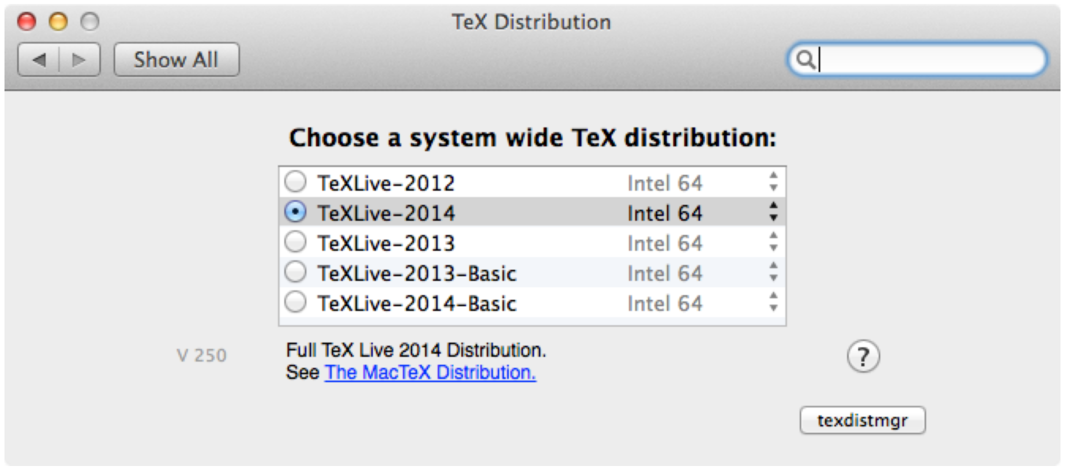
So ... I was looking at this talk I promised to give, and there isn't much of interest here. Then I thought of Apple's playbook and decided to give each of you a free piece of software.

All the developers I talked to in 2000 complained of the schlock; not one of them said "maybe Apple is trying to tell us something." When I got home, I discovered that the schlock software could rip the various tracks on a CD to mp3 files. It could play these mp3 files on the Mac, and it could write new CD's with a personal selection of mp3'ss. If you had an mp3 player, the software could upload the mp3's to it and you could play those mp3's about town. Shortly after that, Apple introduced the iPod, iTunes, and the iTunes store. I, unfortunately, have nothing up my sleeve.

2 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 invented by Gerben Wierda and Jérôme Laurens describing the distribution in `/Library/TeX`. These choices were somewhat controversial and I once gave a TUG talk about them. Now I'm happy.

Each year's 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 new distribution and also changes the command line so command line programs use it.



Choose a system wide TeX distribution:

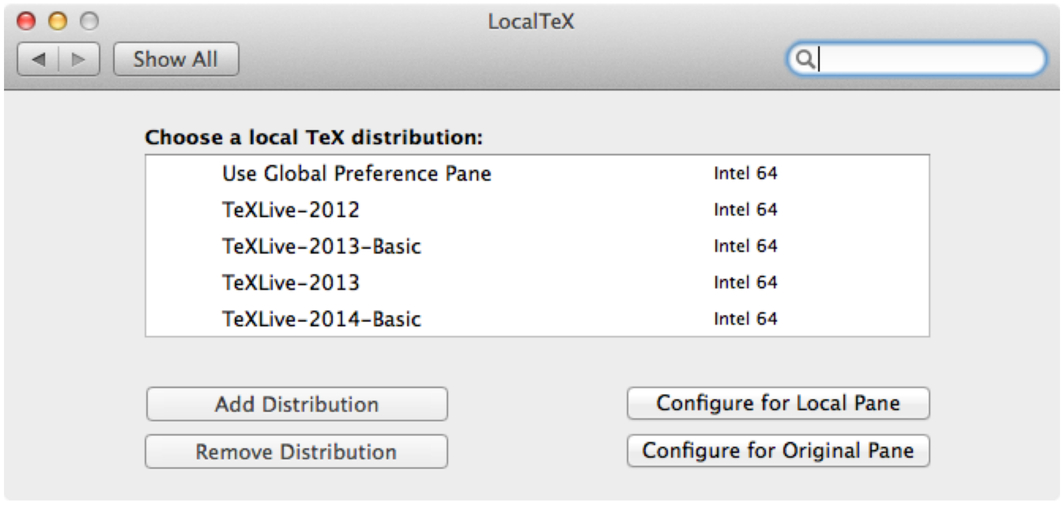
○ TeXLive-2012	Intel 64
● TeXLive-2014	Intel 64
○ TeXLive-2013	Intel 64
○ TeXLive-2013-Basic	Intel 64
○ TeXLive-2014-Basic	Intel 64

Full TeX Live 2014 Distribution.
See The MacTeX Distribution.

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 below, is such a Pane.



Choose a local TeX distribution:

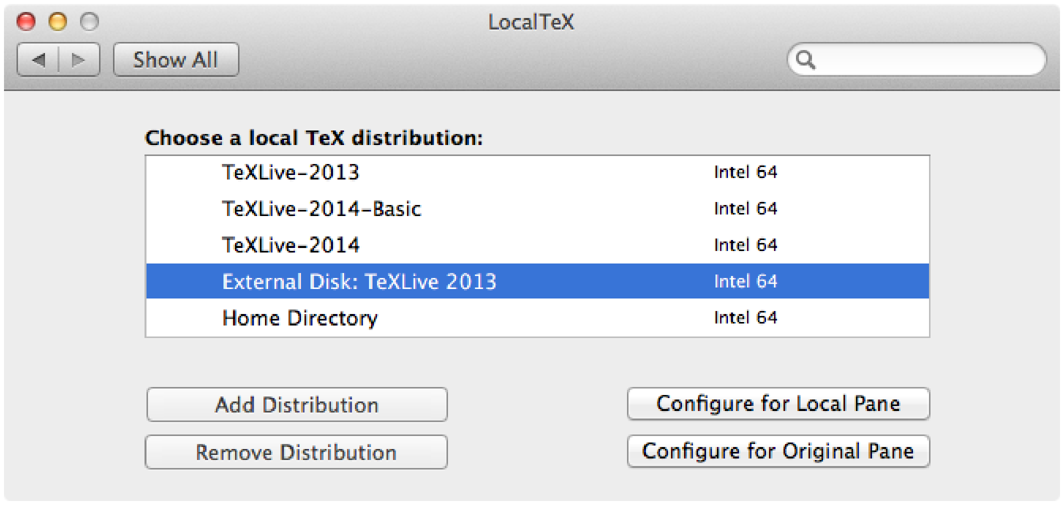
Use Global Preference Pane	Intel 64
TeXLive-2012	Intel 64
TeXLive-2013-Basic	Intel 64
TeXLive-2013	Intel 64
TeXLive-2014-Basic	Intel 64

Figure 2: Local Pref Pane

It is installed 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.

Scrolling down in the list of distributions in the Pane, we see on the next page 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.



Choose a local TeX distribution:

TeXLive-2013	Intel 64
TeXLive-2014-Basic	Intel 64
TeXLive-2014	Intel 64
External Disk: TeXLive 2013	Intel 64
Home Directory	Intel 64

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 on the next page. 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.

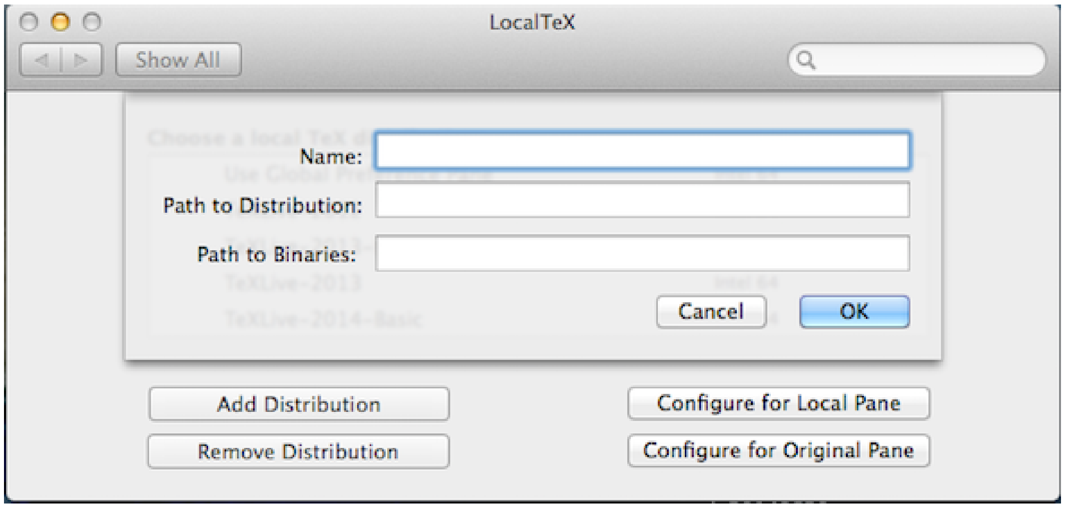


Figure 4: Local Pref Pane

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.

3 Installing and Configuring the LocalTeX Pane

Installing the LocalTeX pane is very easy. Find and double click LocalTeXInstall.pkg and install in the standard manner. This step requires root access and does just one thing: it installs LocalTeX in `/Library/PreferencePanes`, where it is accessible by all users.

A user without root access could instead double click the LocalTeX.prefPane file. 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.

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.

Done. You are ready to go.

To return to the Global Pane and stop using LocalTeX, push “Configure for Original Pane” to configure TeXShop, TeX Live Utility, and BibDesk for the original pane. Done. Don’t do this if you are merely choosing “Use Global Preference Pane” in the LocalPane.

4 How Does the Pane Work?

The LocalTeX pane creates three symbolic links in `~/Local/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 then 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, so instead of writing `~/Library/TeX/LocalTeX/texbin`, I'd write `/Users/koch/Library/TeX/LocalTeX/texbin`.

5 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, since it is used by almost all third party applications and wholesale Apple changes would create a nightmare.

Apple's increasing security work caused several Pref Panes to fail with system updates. For most of the year, the Global Pane didn't work on Mavericks. It was fixed just before the release of MacTeX-2014. But the current Global Pane is again broken in Yosemite. The Local Pane is immune to security concerns, and works on Mavericks and Yosemite. It requires Mountain Lion and above, since it uses Apple's newer ARC memory protection scheme.

6 Configuring Terminal

To finish configuration of LocalTeX, add /Users/koch/Library/TeX/LocalTeX/texbin to your PATH before the item /usr/texbin. Here and in the following paragraphs, always 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.

Files starting with a period are in your home directory, but are hidden. 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 top, add the following two lines

```
# 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
```

7 MacTeX Design Philosophy

On to the main course. 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 was demanded by Wendy McKay, who is at this conference. It is a "one button" install package for TeX, Ghostscript, and various GUI applications, which presents a familiar Installer interface for Mac users, asks no questions, and produces a completely configured installation ready to do serious work. Wendy organized an innocent lunch at the North Carolina Conference of 2005, said "we need an installer", pointed to Jonathan Kew and said "you write it, Jonathan." Kew had to leave early the next day. He stayed up all night, wrote the installer using scripts which made it easy to maintain, and willed it to me at breakfast the next day. I was bleary eyed, but Jonathan was wide awake and then he was gone and there was nothing I could do.

That first version installed Gerben Wierda's teTeX based version of TeX. But around this time, Thomas Esser abandoned teTeX, telling his users to switch to TeX Live. Gerben responded by producing 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 the time since then, Karl Berry never asked me "why don't you Mac folks use TeX Live?" But I knew perfectly well why we didn't. It was nerdy, it asked lots of obscure questions during installation, and it was difficult to maintain. Pressed against the wall in 2006, I decided to try to get it working. Thirty minutes later (almost entirely download time) it was working without asking a single nerdy question. 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!"

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 is for the moment moribund, but its users remember it with great emotion. Every philosophy has a “yes, but ...”)

8 TeXShop Design Philosophy

Surprisingly, TeXShop has a very different design philosophy, which may be more controversial here. 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.

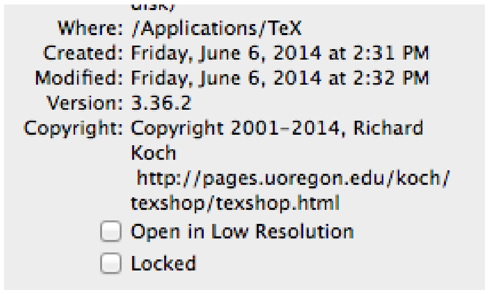
To understand why, consider the following three messages from the TeX on OS X mailing list:

```
From: Warren Nagourney <wnagourney@comcast.net>
Subject: [OS X TeX] Fwd: Slight pdf distortions in preview (TeXshop)
Date: April 20, 2013 2:47:12 PM PDT

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.
This seems to happen after scrolling the window using gestures on the trackpad.
The slant is enough to make the test appear italicized, which is a bit annoying.
Has anyone else noticed this?

From: Giovanni Dore <giovanni.dore@unibo.it>
Subject: R: [OS X TeX] Fwd: Slight pdf distortions in preview (TeXshop)

I think that this is not a problem of TeXShop. I use Skim and sometimes
I have the same problem, it seems that the lower half of the line is shifted
of one pixel with respect to the upper half.


From: Victor Ivrii <vivrii@gmail.com>
Subject: Re: [OS X TeX] Fwd: Slight pdf distortions in preview (TeXshop)
Date: April 20, 2013 3:07:19 PM PDT

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

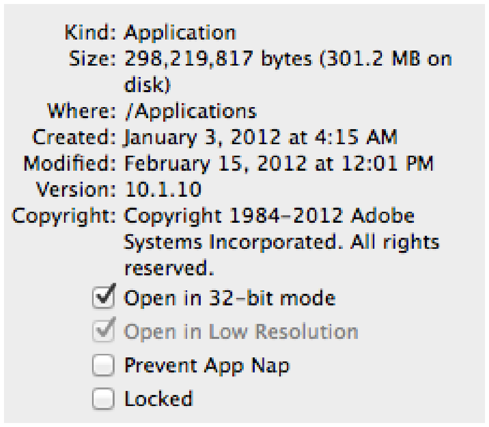
All three messages are from knowledgable people active in the TeX on OS X list. The third message is particularly helpful because TeXShop, Preview, and Skim use Apple's PDFKit to display pdf files, while Adobe Acrobat Reader has its own pdf rendering code, and TeXWorks uses poppler to render pdf. And indeed, TeXShop, Preview, and Skim have a display problem while 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, Preview, 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, Preview, or Skim.

If you select "Get Info" in the Finder with a program selected, you get a panel of information about the program. Here is part of that panel for TeXShop on the left, and for Adobe Acrobat Reader on the right, displayed on a Retina machine.



Where: /Applications/TeX
Created: Friday, June 6, 2014 at 2:31 PM
Modified: Friday, June 6, 2014 at 2:32 PM
Version: 3.36.2
Copyright: Copyright 2001-2014, Richard Koch http://pages.uoregon.edu/koch/texshop/texshop.html
☐ Open in Low Resolution
☐ Locked



Kind: Application
Size: 298,219,817 bytes (301.2 MB on disk)
Where: /Applications
Created: January 3, 2012 at 4:15 AM
Modified: February 15, 2012 at 12:01 PM
Version: 10.1.10
Copyright: Copyright 1984-2012 Adobe Systems Incorporated. All rights reserved.
✓ Open in 32-bit mode
✓ Open in Low Resolution
☐ Prevent App Nap
☐ Locked

Figure 5: About ...

There are some differences. Reader has both 32 bit and 64 bit code, while version 3 of TeXShop shown here only has 64 bit code. Reader supports App Nap, while TeXShop does not (something for me to do when I'm home). But the key difference is the option to open in Low Resolution mode, while is selectable in TeXShop but not in Reader. This 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.

The Retina Display Portable was introduced in June of 2012, but Adobe Reader and TeXWorks still don'tn support it two years later. If you bought a portable for the Retina display, and you spend most of your time in TeX, and and you use a cross platform GUI, then you probably wasted your money.

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.*

9 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. NeXT had earlier developed a technology which ran its applications on Windows, adopting the Windows Look and Feel on these machines. This software cost $600 a pop, but Apple told developers that they could put the software on their distribution disks for free. So Apple's strategy was: write your software for the new Apple system, and it will then also automatically run on Windows.

I was wildly enthusiastic about all this, telling my colleagues that we could soon write software for students which would run on *both* Macs and PCs. But among commercial developers, the announcement went over like a lead balloon. It is not difficult to see why. Developers were being told that they should write first for Apple, a company perilously close to bankrupcy, using an operating system which wouldn't be released for five more years. As for windows, where they made all their money, their software would also run there and would use any windows feature which Apple decided to support in its conversion software. Not a single major software developer endorsed this announcement. Apple's respected head of developer relations, Heidi Roizen, quit, 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 multi-tasking 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, and even that could be done with Java rather than Objective C. 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. (The software allowing Cocoa apps to run on Windows has never been mentioned since its 1997 appearance.)

I began regularly attending WWDC in 2002, and this pattern continued for several years. Carbon sessions were in the main room to huge crowds. Cocoa sessions were in half-filled rooms, very slowly gathering steam. Carbon could not support all the features of Apple’s new graphic system, so Apple invented a “High Level Toolbox” for Carbon to provide this support. At the next WWDC, the main auditorium was filled with Carbon developers learning how to use the High Level Toolbox.

In 2005, Apple switched to Intel processors. They told developers that moving a Cocoa app to Intel often involved just a recompile. Carbon apps, they estimated, could be moved in a month. Half of carbon apps were written with a third party system, Metrowerks, which decided not to make the transition at all. These developers had to first move their source code to Apple’s XCode.

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 iPhone software. But outside developers couldn’t program the iPhone. Therefore 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, but 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. It surely helped to know that the iPhone could only be programmed in Cocoa. And Apple probably knew by then that the iPad, not yet introduced, was also programmed solely 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 apps like Mathematica are still in Carbon.

During these turbulent times I was mostly oblivious to the drama. TeXShop was written in Cocoa, but compiled in 32 bits since I saw little reason to change. I didn’t adopt new OS X features.

But then TeXShop began crashing. I eyed the garbage collector available in 64 bits with envy. Shortly before Lion was announced, I moved TeXShop to 64 bits, and began planning for GC. 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.

10 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, and mouseDown, which is called when the main mouse button is pressed inside the View.

The NSView defined by Cocoa does nothing when drawRect is called, and does nothing when mouseDown is called.

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, myNSView might override drawRect so it draws a picture of Portland rather than drawing nothing. 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 knows when NSView should draw, so it 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 slogan "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 a picture of Portland 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 on 32 bit Mac applications.

After this change, Apple rapidly made many older machines obsolete. Snow Leopard required Intel processors, Lion required 64 bit processors, and Mountain Lion required machines running the kernel in 64 bits. Notice that since then, the policy changed: Mavericks and Yosemite run on all machines that can run Mountain Lion, and are free to boot. I cannot confirm this, but I strongly 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.

11 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.

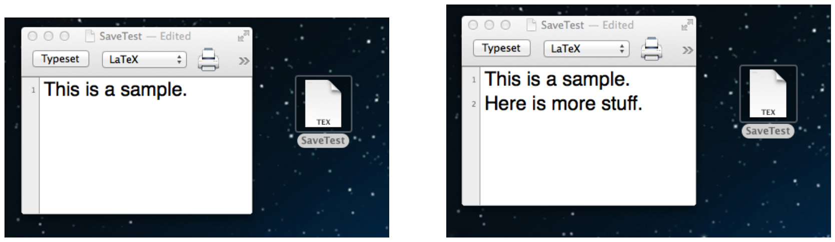
One of the standard requests for TeXShop was that it remember window sizes and positions when quit, and restore these windows automatically when next restarted. To shame me into working on this, users told me of other GUI's for TeX which already had the ability.

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 possible to turn the feature on or off for all applications in the General pane of Apple's System Preferences: check or uncheck "Close windows when quitting an application." 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 consistently in all applications, so they work in TeXShop.

12 Automatic Saving

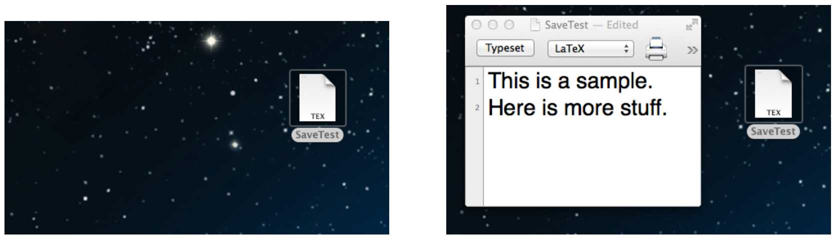
Saving window positions is something I could have done myself if I weren't lazy. But the second Lion feature is something I would never have tried on my own: automatic file saving. Here is a brief demonstration.



This is a sample.

This is a sample.
Here is more stuff.

Figure 6: Steps 1 and 2



This is a sample.
Here is more stuff.

Figure 7: Steps 3 and 4

On the top left, I opened the file SaveTest, which had one line of text. Just to the right I added a second line of text. On the bottom left, I quit. TeXShop didn't ask whether it should save the changes. On the bottom right, I opened TeXShop. It automatically opened the file, which had automatically been saved.

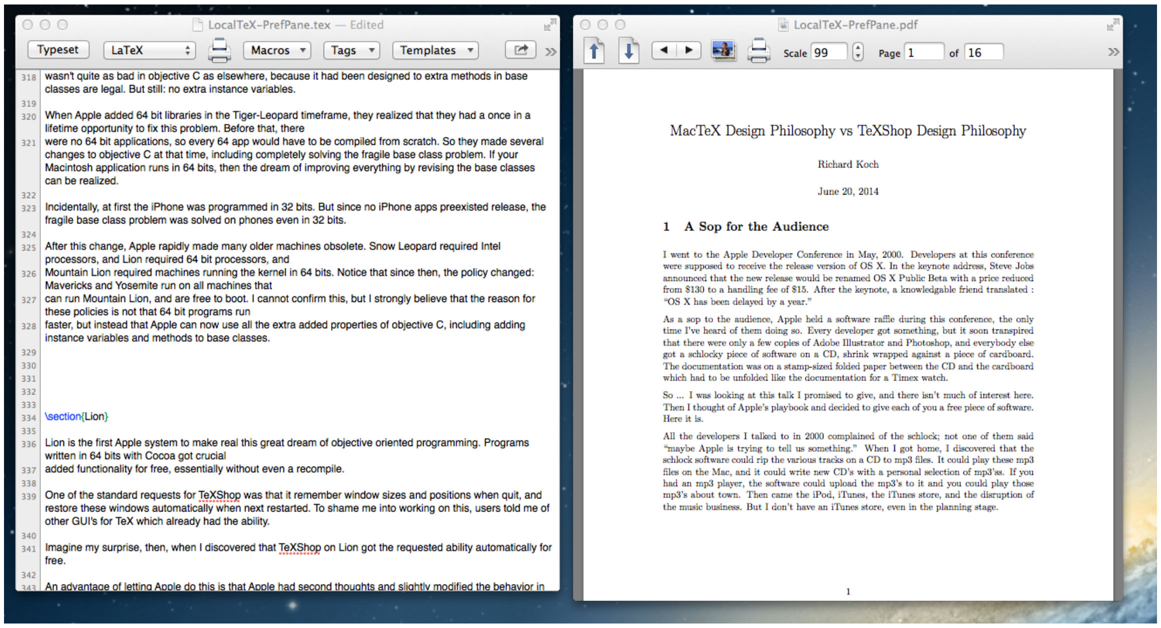
But wait. There's more. Open files are saved every few minutes, automatically. Only the changes are saved, so there is no noticeable disk activity, no momentary pauses, and no sound at all on a Mac with solid state drives. TeXShop never asks if it should save a file because it automatically saves if a file is duplicated, or copied, or in other obvious circumstances. There is still a Save menu item in case you made a change that immediately needs protection, but essentially you can forget about saving.

But wait. Suppose I send a file to someone? Will they get all the changes? Of course not; they get the latest version. The Mac handles all the details of making that happen. If I ask the Finder for the size of a file, it gives the size of the latest version. Etc.

System memory of old versions is gradually pared down, so you'll have several versions from today, fewer from last week, and fewer still from last month.

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 in the next page shows this document during editing. The bottom picture shows the effect of selecting the menu Revert To → Browse All Versions.



```
wasn't quite as bad in objective C as elsewhere, because it had been designed to extra methods in base classes are legal. But still: no extra instance variables.

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

Incidentally, at first the iPhone was programmed in 32 bits. But since no iPhone apps preexisted release, the fragile base class problem was solved on phones even in 32 bits.

After this change, Apple rapidly made many older machines obsolete. Snow Leopard required Intel processors, and Lion required 64 bit processors, and
Mountain Lion required machines running the kernel in 64 bits. Notice that since then, the policy changed: Mavericks and Yosemite run on all machines that
can run Mountain Lion, and are free to boot. I cannot confirm this, but I strongly 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.





\section{Lion}

Lion is the first Apple system to make real this great dream of objective oriented programming. Programs written in 64 bits with Cocoa got crucial
added functionality for free, essentially without even 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
```

MacTeX Design Philosophy vs TeXShop Design Philosophy

Richard Koch

June 20, 2014

1 A Sop for the Audience

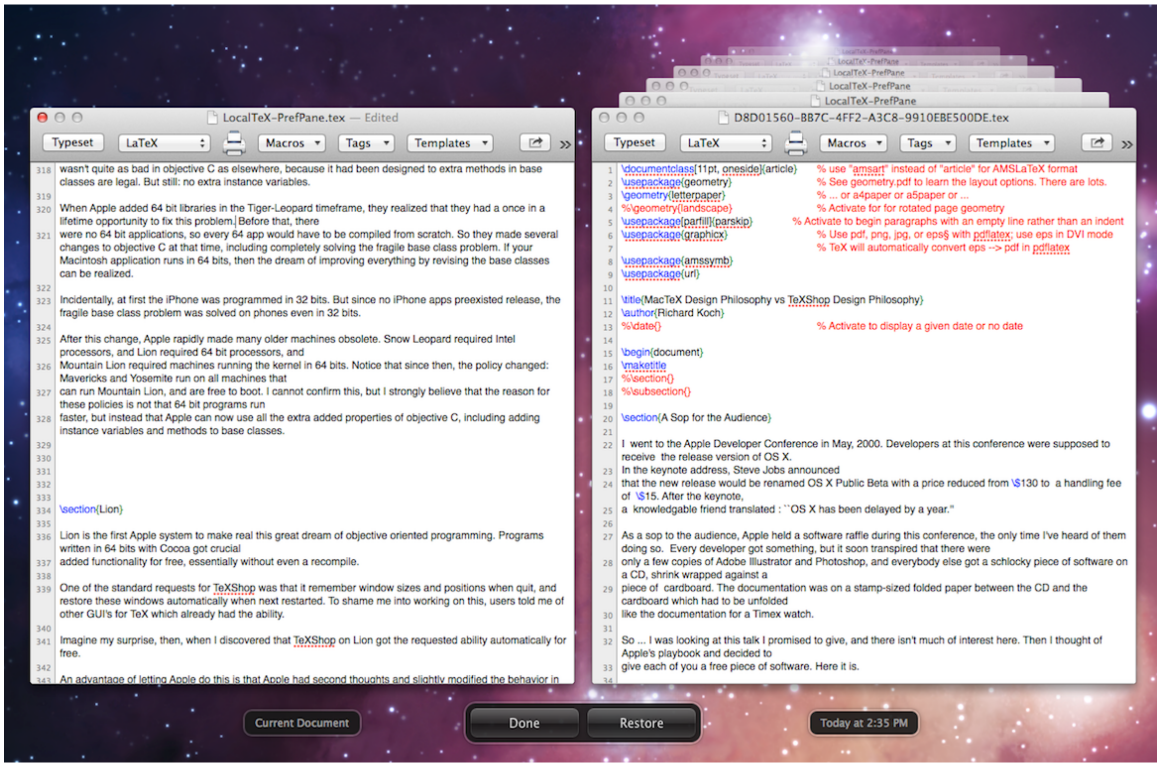
I went to the Apple Developer Conference in May, 2000. Developers at this conference were supposed to receive the release version of OS X. In the keynote address, Steve Jobs announced that the new release would be renamed OS X Public Beta with a price reduced from $130 to a handling fee of $15. After 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 there were only a few copies of Adobe Illustrator and Photoshop, and everybody else got a schlocky piece of software on a CD, shrink wrapped against a piece of cardboard. The documentation was on a stamp-sized folded paper between the CD and the cardboard which had to be unfolded like the documentation for a Timex watch.

So ... I was looking at this talk I promised to give, and there isn't much of interest here. Then I thought of Apple's playbook and decided to give each of you a free piece of software. Here it is.

All the developers I talked to in 2000 complained of the schlock; not one of them said "maybe Apple is trying to tell us something." When I got home, I discovered that the schlock software could rip the various tracks on a CD to mp3 files. It could play these mp3 files on the Mac, and it could write new CD's with a personal selection of mp3'ss. If you had an mp3 player, the software could upload the mp3's to it and you could play those mp3's about town. Then came the iPod, iTunes, the iTunes store, and the disruption of the music business. But I don't have an iTunes store, even in the planning stage.

1



```
wasn't quite as bad in objective C as elsewhere, because it had been designed to extra methods in base classes are legal. But still: no extra instance variables.

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

Incidentally, at first the iPhone was programmed in 32 bits. But since no iPhone apps preexisted release, the fragile base class problem was solved on phones even in 32 bits.

After this change, Apple rapidly made many older machines obsolete. Snow Leopard required Intel processors, and Lion required 64 bit processors, and
Mountain Lion required machines running the kernel in 64 bits. Notice that since then, the policy changed: Mavericks and Yosemite run on all machines that
can run Mountain Lion, and are free to boot. I cannot confirm this, but I strongly 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.





\section{Lion}

Lion is the first Apple system to make real this great dream of objective oriented programming. Programs written in 64 bits with Cocoa got crucial
added functionality for free, essentially without even 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
```

```
\documentclass[11pt, oneside]{article}      % use "amsart" instead of "article" for AMSLaTeX format
\usepackage{geometry}                       % See geometry.pdf to learn the layout options. There are lots.
\geometry{letterpaper}                      % ... or a4paper or a5paper or ...
%\geometry{landscape}                       % Activate for for rotated page geometry
\usepackage[parfill]{parskip}           % Activate to begin paragraphs with an empty line rather than an indent
\usepackage{graphicx}                       % Use pdf, png, jpg, or eps§ with pdflatex; use eps in DVI mode
                                            % TeX will automatically convert eps --> pdf in pdflatex
\usepackage{amssymb}
\usepackage{url}

\title{MacTeX Design Philosophy vs TeXShop Design Philosophy}
\author{Richard Koch}
%\date{}                                    % Activate to display a given date or no date

\begin{document}
\maketitle
%\section{}
%\subsection{}

\section{A Sop for the Audience}

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 there were
only a few copies of Adobe Illustrator and Photoshop, and everybody else got a schlocky piece of software on a CD, shrink wrapped against a
piece of  cardboard. The documentation was on a stamp-sized folded paper between the CD and the cardboard which had to be unfolded
like the documentation for a Timex watch.

So ... I was looking at this talk I promised to give, and there isn't much of interest here. Then I thought of Apple's playbook and decided to
give each of you a free piece of software. Here it is.
```

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.

When I first saw AutoSave, I was slightly dubious. But I've never had the slightest difficulty, and the Revert command has saved my hide more than once. I'll confess that I don't use Time Machine because I don't like the sound of an External 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, as shown on the next page. On the left is the TeXShop File menu as defined in NSMenu.nib in the developer source code. On the right is the same menu as it actually appears when running under Mavericks. Notice the changes. 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.

After one firey email discussion, 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 accesing 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?

In Apple's NSDocument, there is a routine named 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.

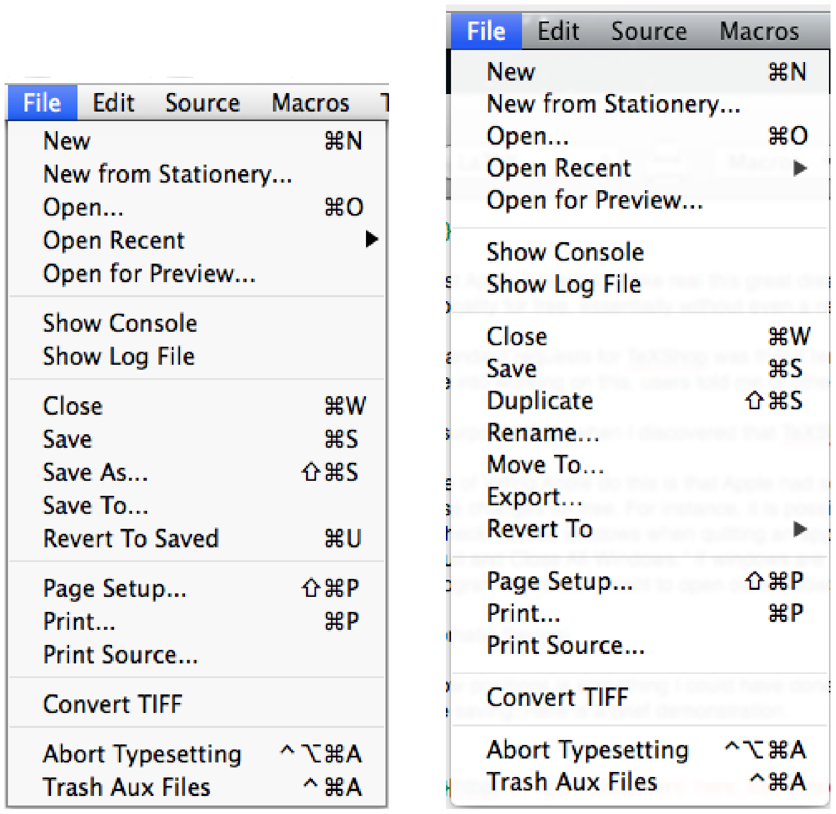
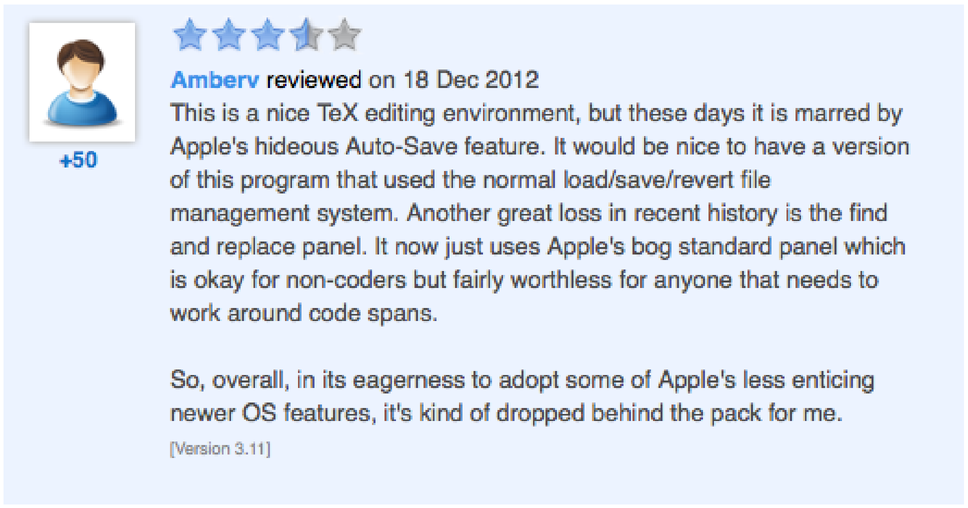


Figure 8: File Menu in Source Code and File Menu as Displayed by Mavericks

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



Amberv reviewed on 18 Dec 2012

This is a nice TeX editing environment, but these days it is marred by Apple's hideous Auto-Save feature. It would be nice to have a version of this program that used the normal load/save/revert file management system. Another great loss in recent history is the find and replace panel. It now just uses Apple's bog standard panel which is okay for non-coders but fairly worthless for anyone that needs to work around code spans.

So, overall, in its eagerness to adopt some of Apple's less enticing newer OS features, it's kind of dropped behind the pack for me.

[Version 3.11]

13 Automatic Reference Counting

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 and adopting relevant changes. 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.....

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? That's the important topic!"

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. On the other hand, 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." The essential idea is that every so often the program stops what it is doing, examines each object to find out how many parts of the program are using it, and throws away objects not used by anyone. Java and a number of other languages use garbage collection, and Apple added it in the Leopard time frame. In Snow Leopard, Apple required that 64 bit Preference Panes use garbage collection.

But it turned out that garbage collection didn't work well on the phone. A small delay to collect garbage isn't appreciated if a phone rings but cannot be answered. So on the phone and the iPad, programmers were back to manually handling memory.

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. This system worked so well that Apple deprecated garbage collection and recommended ARC for modern programming. Since ARC essentially 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. This ARC code first appeared in TeXShop 3.35 and makes the program much more stable. There are a couple of remaining issues to be tackled later this summer.

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