TeXShop Manual
Version 1.1

Richard Koch

July 3, 2023

Contents

Chapter 1

About TeX, LaTeX, and TeXShop

1.1 Introduction

TeX and LaTeX are typesetting programs, available without cost on most computer platforms. The user enters instructions using symbols on an ordinary typewriter, and the program converts these instructions into a document that looks like it came from a professional book publisher.

TeX has been carefully designed to produce exactly the same output on all computer platforms. If the source file is written on a Macintosh and sent to someone using Windows, Linux, or Unix, the typeset output on both platforms will use the same fonts, have the same line breaks, have the same page breaks, and look identical. This makes collaboration easy.

TeXShop is a GUI front end for TeX. In contrast to TeX, the program runs only on the Macintosh, making extensive use of the interface features available on that platform. It is available without cost.

Making TeX platform independent, but making the front end dependent on the platform is a deliberate design decision. Books, articles, letters, and slides can be read by anyone. But when an author is writing such a document, the author should be able to use all the advanced features that a particular operating system provides.

TeXShop displays an editing window where users type the text they want to typeset. It provides a button which users push to call the typesetting program, and it provides a preview window where the typeset document is displayed.

TeXShop does not do the actual typesetting. That task is performed by the TEX and LATEX command line programs not visible to an ordinary user. TeXShop does not manipulate the source in any manner before passing it to TEX to typeset. Below is a typical TEX source file.

```
\documentclass[11pt,oneside]{article}
\usepackage[parfill]{parskip}

\title{Short Sample}
\author{Richard Koch}

\begin{document}

\maketitle

\section{Introduction}
This is the first sentence. We type
{\bold this command} to get bold type and
{\emp this command} to get italics.  TeX ignores      extra
spaces because it knows how to correctly format.

A blank line starts a new paragraph.

\TeX\ produces beautiful mathematics, like
$\sqrt{x^2 + 5 x + 6}$ and $\Gamma^i_{j_1 j_2}$ and
$$\int_0^\infty e^{- x^2}  \ dx = {{\sqrt{\pi}} \over 2}$$

\end{document}
```

Below is the output when this source is processed by T_EX:

<div align="center">

Short Sample

Richard Koch

September 13, 2022

</div>

1   Introduction

This is the first sentence. We type **this command** to get bold type and
this command to get italics. TeX ignores extra spaces because it knows the
rules of typesetting.

A blank line starts a new paragraph.

T_EX produces beautiful mathematics, like $\sqrt{x^2 + 5x + 6}$ and $\Gamma^i_{j_1 j_2}$ and

$$\int_0^\infty e^{-x^2} \, dx = \frac{\sqrt{\pi}}{2}$$

1.2   Donald Knuth and the Creation of TEX

Donald Knuth is a professor of computer science at Stanford University. In 1962 while he was a Caltech graduate student, Knuth agreed to write a book on theoretical computer science. He began the first draft in 1963 after receiving his PhD, and finished it in 1965. His initial draft had 3000 hand-written pages, which Knuth believed would shrink to about 600 printed pages. But the publisher informed him that in fact those 3000 pages would produce about 2000 printed pages. Plans changed after that, and the publisher agreed to publish the work in seven volumes, to be called *The Art of Computer Programming.* These volumes are classics, owned and consulted by most computer scientists working today. The first three volumes were published in 1968, 1969, and 1973, and work on volume four began in 1973.

A couple of years later, Knuth revised volume two for reprinting. But when he got the galleys from the publisher in March, 1977, he was unhappy with the result. He learned that the hot lead process used originally had been replaced with photo-offset printing, and thus with a different collection of fonts. Knuth told the publisher to postpone further action for six months while he consulted experts at Stanford and obtained better fonts for the system. That six month project actually took ten years, as Knuth brought typography experts to Stanford and learned the intricate details of typesetting.

Knuth wrote a specification of TEX in May, 1977. The first version was released in 1978, written in the SAIL programming language then in use at Stanford. In that same year, Knuth gave the 1978 Gibbs Lecture, an annual lecture for the American Mathematical Society. An expanded version was published by the AMS in December of 1979, titled *TeX and Metafont, New Directions in Typesetting.* That is how mathematicians first learned of the project.

In 1982, Knuth substantially revised TeX and rewrote the entire program in Pascal. This is the version that most early adopters first saw. Use of the program gradually increased during the 1980's, as experts from other countries began making important contributions. Originally, each font used in TeX had at most 128 characters. This was sufficient for the letters on an American typewriter, but not sufficient for the umlauts, accents, upside down question marks, and other characters used in Western Europe. The Europeans convinced Knuth to increase font table sizes to 256 characters for version 3.0 of TeX, released in 1989. In 1991, Knuth froze the design of TeX, believing that the stability of an unchanged system was more important than new features. As we will see, other programmers later created pdfTeX, luaTeX, XeTeX, and other systems with Knuth's approval, but if a program is named TEX, it is the 1989 version by Knuth.

Knuth returns to the program every few years to fix minor bugs. The current version as of 2022 is 3.141592653. Knuth has decreed that the version number will be set to exactly π when he dies, and the code will not change after that.

Knuth's typesetting project produced three important products: TeX, MetaFont, and the Computer Modern Fonts. He described the results in five books, standing alongside the volumes of the *Art of Computer Programming*: one book with the complete TeX program, one book with the complete Metafont program, one book showing the design of the Computer Modern Fonts with an entire page devoted to each character, one book with the instruction manual for TeX, and one book with the instruction manual for MetaFont.

After TeX was frozen, Knuth resumed work on the Art of Computer Programming. Plans for Book 4 had expanded, and Volume 4A, published in 2011, was followed by volume 4B published in October, 2022. Coming next is volume 4C and perhaps 4D. These will be followed by Volumes 5, 6, and 7, time permitting.

1.3   TeX and LaTeX

The TeX program understands a limited number of very primitive commands. Because these commands are primitive, several must be strung together to produce desired results. TeX allows users to define *macros*, new commands made by stringing primitive commands together, and Knuth wrote a large collection of such macros known as "Plain TeX." When users today say they use TeX, they are really claiming to use TeX and the Plain TeX macro package.

In the early 1980's, Leslie Lamport wrote a much more extensive set of macros called LaTeX. Some users today think of TeX and LaTeX as separate programs, but that is incorrect; instead TeX is the TeX typesetting program used with the Plain TeX macro package, and LaTeX is the TeX typesetting program used with the LaTeX macro package.

Versions of LaTeX were released in 1984 and 1985. In 1986 Lamport published a user-manual for LaTeX titled *LaTeX User's Guide and Reference Manual*. This has remained a standard reference ever since. The manual covered LaTeX 2.09, the version then being used.

Meanwhile, Michael Spivak wrote a series of macros for mathematicians called the AMS-TeX Macro Package and documented this package in his book *The Joy of TeX: A Gourmet Guide to Typesetting*.

This led to requests to combine LaTeX and AMS-TeX, which turned out to be a non-trivial task. In 1989, Lamport turned over maintenance of LaTeX to Frank Mittelbach, who then formed a team with Chris Rowley and Rainer Schopf to update LaTeX to version 3. In 1994, they released a preliminary version, known as LaTeX2e. This version made it possible to merge LaTeX with AMS-TeX, and has become the standard version of LaTeX. Lamport updated his manual to cover it, and all current versions of the LaTeX User's Guide document LaTeX2e.

Since LaTeX2e accomplished the main task of the upgrade process, work on version 3 gradually subsided. But more recently the project has become active and new versions of LaTeX often appear.

1.4   Outline Fonts

Knuth thought his project was about two equally important programs, TeX and MetaFont. MetaFont is the program used to design font families, and to manipulate these families. In particular, it has code to rasterize fonts, that is, to convert them into an array of dots for the screen or printer.

The key feature of fonts created with MetaFont is that they are defined mathematically rather than as a series of dots. Once the mathematical description of a character is given, it can form a letter so small that it is barely visible, or so large that it covers a complete page. Mathematically-defined fonts are often called *outline fonts* and that is the term I will use, although technically Knuth's fonts weren't outline fonts.

Knuth was the second person to introduce such fonts. For an interesting history, see https://simoncozens.github.io/fonts-and-layout/history.html.

It isn't enough to design a program which can manipulate these new fonts; it is also necessary to actually produce such a font. TeX requires many variations: bold, italics, ligatures, etc. Moreover, fonts with a large number of mathematical symbols are required. The creation of these fonts is the third leg of the typesetting stool which Knuth constructed. He called his fonts *Computer Modern Fonts*.

In 1984, Adobe invented its own version of outline fonts, known as *Adobe Type 1 Fonts*, and used them in Postscript, which it licensed to makers of Postscript Printers. The first fonts in macOS were bitmap fonts, but Apple soon realized the importance of outline fonts. After attempting unsuccessfully to license Adobe Type 1 fonts, Apple created its own outline fonts, *TrueType Fonts*. These fonts first appeared on the Mac in 1991. Apple licensed these fonts to Microsoft and soon they were used on both Macs and PCs. This caused Adobe to release the specifications for Adobe Type 1 fonts, and eventually the Mac could also use these fonts. The history summarized here is actually much more complicated; see https://www.truetype-typography.com/tthist.htm.

The final step in the process was the creation of OpenType fonts by Microsoft and their adoption by both Apple and Windows. The font data in such a font can be either TrueType or Adobe-Type-1, but the font package also contains additional information used to display glyphs, like kerning tables and ligature information.

1.5   MetaFont Today

When outline fonts are used on a computer platform, it is essential to postpone the rasterization step until the last moment. If a typeset document is saved in pdf form, the pdf should contain the mathematical version of the fonts rather than the bitmaps because the pdf may be viewed on many devices with different resolutions.

This is not possible with fonts created by MetaFont, because neither Apple nor Microsoft adopted Knuth's formulas for outline fonts. TeX distributions still contain MetaFont, and an enormous number of fonts created for it over the years. But using these fonts brings many disadvantages, and for twenty years a major project in the TeX world has been modifying the system so it can use TrueType, Adobe Type 1, and OpenType fonts. Part of this project involved creating the LatinModern fonts, which are Adobe Type 1 versions of Knuth's Computer Modern Fonts, expanded to contain the symbols commonly used in Western Europe.

You might think that such a replacement would involve an enormous rewrite of TeX itself, but that was not necessary. The original TeX program output a *device independent file*, that is, a dvi file. This file did not contain any fonts. Instead it contained instructions of the following form:

```
select font named "CM" of size "16 pt"
print glyph 43 at position (23, 69)
print glyph 68 at position (32, 69)
etc.
```

Separate programs then handled the job of reading the dvi file, opening the corresponding font files, rasterizing them, and sending the final data to the printer. The most famous was dvips, written by Thomas Rokicki, which created a postscript file with rasterized font information from a dvi file and associated mathematical font files.

Around 2001, modified versions of TEX and LATEX named pdfTEX and pdfLATEX were created by Hàn Thế Thành. These programs contained the original TEX and could output dvi files, but they could also output pdf files directly. These pdf files could contain either mathematical descriptions of the fonts in Adobe Type 1 or TrueType form, or rasterized MetaFont font data. Nowadays, most people typesetting with TEX or LATEX are actually using pdfTEX or pdfLATEX.

Knuth thought of MetaFont, TEX and the Computer Modern Fonts as equal partners in the new typesetting system. Nowadays, MetaFont has mostly been replaced by font technology from other companies, (although the program plays an important role in some projects). and the Computer Modern Fonts are one of several commonly used font families for technical documents. TEX remains unsurpassed as a typesetting engine.

1.6   Front Ends and TeXShop

TeX is a command line program. A user creates a source text file, say Sample.tex, and typesets it to produce an output file Sample.pdf. The call to activate TeX can be done using Terminal or some other Unix shell, but otherwise there is no need to directly access the TeX program. Often the program is hidden away in the system and hard for a casual user to even find.

Interaction with TeX is usually done via a Graphical User Interface (GUI) front end. This is a standard program which provides an editing window used to create the source file, a button to call the typesetting engine, and a preview window to display the resulting output file. Many front ends are available, often open source and free.

TeXShop is a GUI front end for the Macintosh. It was introduced in 2001, while macOS was still in beta, and has been extended ever since. A great many programmers all over the world contributed code to the program. Since the actual TeX program is invisible, sometimes users get the false idea that TeXShop is itself doing the typesetting. Nothing could be farther from the truth. TeXShop does not modify the source in any way before typesetting. It simply calls TeX, saying in effect "Here's a source file. Typeset it and call me when you are done. In the meantime, I'm going to take a little nap."

1.7   Brief History of TeXShop

I first became acquainted with TeX and LATEX in 1991 when I bought a NeXt machine. This computer came with a TeX distribution, and a front end by Thomas Rokicki. His front end made use of display postscript, the graphic system on the NeXT. It was a wonderful setup, and I gradually switched to it from my previous Macintosh machines.

Around 1995, Apple ran into serious trouble. It was unable to complete its new operating system Copland, and there were rumors that the CEO was trying to find a buyer for the company. In December, 1996, Apple announced that it would use NeXtStep as its new operating system, having bought NeXT lock, stock, and barrel. And Steve Jobs would return as an unpaid consultant.

This announcement thrilled me. At last the Mac would have an honest Unix foundation, advanced user interface, and modern programming environment. As a mainstream machine, it would support standard commercial programs like Word and Excel. For mathematics it only needed a Web browser, an email program, Mathematica, and TeX, and the first three would clearly be available. I began pestering our Apple representative about the importance of TeX. He agreed, but I saw few signs of progress.

I began running beta versions of OS X, and soon had TeX itself running. I tried to compile the Rokicki front end, but Apple switched their graphic system from display postscript to

Quartz, a PDF-based system, so that was a no go. I had written a few NeXtStep programs, and finally one day I thought that it should be possible to write a front end myself.

I do not know when I started coding TeXShop. I know for sure that I had a working TeXShop by May of 2000. I suspect the program was started not much earlier, perhaps in the spring of 2000.

The 2000 Worldwide Developer Conference, WWDC, was in San Jose in May. At this conference, Apple was slated to give developers the release version of OS X, which would be sold to users several months later. I was only teaching one course that spring, a Discrete Mathematics course for Computer Science students. The authors of the book were also at the University of Oregon, so I asked them to teach my class for a week and went to the conference. Just before this conference, I bought a 17 inch PPC Portable. I immediately erased the hard disk and partitioned it so I could install OSX release at the Conference.

In the keynote address in 2000, Steve Jobs said that in the computing industry, marketing considerations sometimes dictate how software is presented. He then announced that the release version of OS X would instead be named OS X Public Beta and instead of costing $120, it would be sold for a $15 handling fee. After the keynote, a friend said to me "Wasn't that smooth? Jobs just announced that OS X is delayed a year."

By this time, TeXShop was definitely working on beta versions of OS X, but the Apple PDF software had a significant bug: the software could not read embedded fonts in files. Consequently, TeX documents written with Times Roman displayed beautifully, but any mathematics in the document would appear as blank areas. I hurried back to the motel at the end of the Monday session, installed the public beta, and ran TeXShop. Embedded fonts still did not display.

The first release of TeXShop to the open source community occurred shortly afterward, on July 23, 2000. This version called Ghostscript to rasterize pdf output so it could display mathematical fonts. This created a somewhat fuzzy final display. All the while, Apple was working on OS X, and in March of 2001 they announced the actual release version. This time, developers got their copy only a week before the official release. When I got my copy, I immediately tried TeXShop. Apple had fixed the embedded font problem. I went to the UO and bought five letter sweaters with the University of Oregon logo, and send them to Apple's developers as my thanks for the bug fix. Then I ripped out that call to Ghostscript and released the first official version of the program. That code, of course, ran on 32 bit PowerPC processors. But recently I recompiled it for 64 bit Intel and Arm processors and it still ran. The initial version, with source code, is on my web site. After the release, an enormous number of people from all over the world contributed to the program. Their names are listed in the "about file". So it is a joint effort of all of these developers, most of whom I have never met.

Chapter 2

Installing TeX on a Macintosh

2.1   TeX Live

To use TeX and LaTeX on the Macintosh, you must install two items: a TeX distribution and a front end.

A TeX distribution contains thousands of files, because as TeX grew more popular, people all over the world contributed to the project. It contains the actual programs, but also hundreds of fonts, style file, class files, scripts, and documentation files.

There are several different TeX distributions for the Macintosh, and TeXShop can work with all of them. Some people install TeX using *MacPorts* or *Homebrew*, projects porting command-line Linux programs to the Mac. The most common Windows distribution, *MikTeX*, is also available on the Macintosh.

But by far the most common distribution on the Mac is *TeX Live*. This distribution works on all computer platforms: Macintosh, Windows, Linux, Unix, etc., using the same support files, fonts, etc. The command-line programs, which must vary depending on computer platform, are compiled from the same source files. TeX Live is produced by several cooperating TeX User Groups across the world, pulling sources from *CTAN*, the main repository of TeX-related files.

Installing this distribution on the Macintosh is very easy because it has been packaged as a standard install package using Apple's install software. To install from such a package, double click and answer a few straightforward questions, push a button, and watch installation proceed. At the end the system is completely configured and both TeX Live and TeXShop are ready to typeset.

2.2 Installing TeX Live and TeXShop

Two install packages are provided at https://tug.org/mactex/, named *BasicTeX* and *MacTeX*. Each installs vanilla versions of TeX Live; nothing has been modified for the Mac. These two distributions do not interfere with each other, so one can be installed and then later the other can be installed without erasing anything. If both are installed, both can be used and switching from one to the other is easy. But it is also easy to erase an installation by just dragging one folder to the trash.

BasicTeX is for users experimenting with TeX, who are perhaps uncertain whether they will keep using the program. The package is small, about 94 megs, but capable of typesetting with either TeX or LaTeX using most facilities.

MacTeX is for users who definitely need TeX for serious projects. It is large because it installs everything that a user could conceivably need. The size of the package is 4.7 gigabytes. It can be difficult and frustrating to add extra fonts, styles, or classes to TeX after it is installed, so installing everything avoids that frustration. After MacTeX is installed, essentially any example from a book about TeX will just work.

MacTeX is sometimes criticized for installing too much, including items the user will never need. But the advice to install everything comes from talks given at TeX User Group Conferences by speakers who used TeX for less than a year. They unanimously advised installing everything and thereby avoiding frustration.

BasicTeX installs everything in /usr/local/texlive/2022basic. MacTeX installs everything in /usr/local/texlive/2022. These distributions are updated once a year, so the 2022 will become 2023 around April of 2023. To uninstall BasicTeX, just drag the 2022basic folder to the trash. To uninstall MacTeX, just drag the 2022 folder to the trash.

MacTeX installs TeXShop in /Applications/TeX. After installing, a user should select the menu item "Check for Updates" to update TeXShop to the very latest version.

BasicTeX does not install TeXShop. To obtain it, go to https://pages.uoregon.edu/koch/texshop/texshop.html. Download the program and drag it to /Applications or /Applications/TeX.

2.3 Capabilities of TeX

The typesetting system installed by TeX Live is capable of typesetting in virtually any language used in the world. It supports the umlauts and accents used in Western Europe. It supports Cyrillic, Modern Greek, and ancient Greek. It contains CJK fonts for work in Chinese, Japanese, and Korean. It can typeset in Arabic, Hebrew, and Persian, three languages written from right to left, and the appropriate source code will be then entered

into TeXShop also from right to left. The program has been used to typeset the complete Bible, and the complete Koran.

Many TeX contributors come from India; others come from Vietnam. These countries use entirely different scripts, but they are supported by TeX Live.

TeX Live was originally designed to support mathematical typesetting, and provides superb output of mathematics. But it is also used for poetry, novels, and the like in fields far removed from technology.

2.4 Understanding TeXShop

The initial versions of TeXShop were extremely straightforward, TeXShop became known as an easy to use but very basic front end. Over the years, additional facilities were added at the request of users, and today the program has features making T$_E$X input fast and efficient. But these features make the program more complicated.

It is tempting to begin TeXShop work by examining all the menu items and opening TeXShop Preferences to understand and select the various options available. *I do not recommend this approach. Many features will only make sense after you have used T$_E$X for several months.*

The next chapter lists everything you need to know to do serious work with TeXShop. I recommend paying close attention to these items, and then immediately starting to write T$_E$X or L^{A}T$_E$X code. Everyone knows how to use a basic editor, and that will suffice at first.

It will take from a week to a month to become proficient in T$_E$X. After that, return to this manual and browse through the various chapters for extra information.

It is not necessary to read this manual linearly. Browse through it briefly to see what is covered, and then carefully read the chapters which matter at the moment.

Chapter 3

Getting Started

3.1 First Steps with T_EX and TeXShop

Go to `/Applications/TeX` and find TeXShop. Drag its icon to your dock. Click this icon to run the program. You will be presented with a blank window; at the top right of this window you'll find a pull down menu named "Templates". From this menu choose "LaTeX Template". The blank window will fill with some standard boilerplate required in each T_EX source file. The red lines are comments which T_EX ignores.
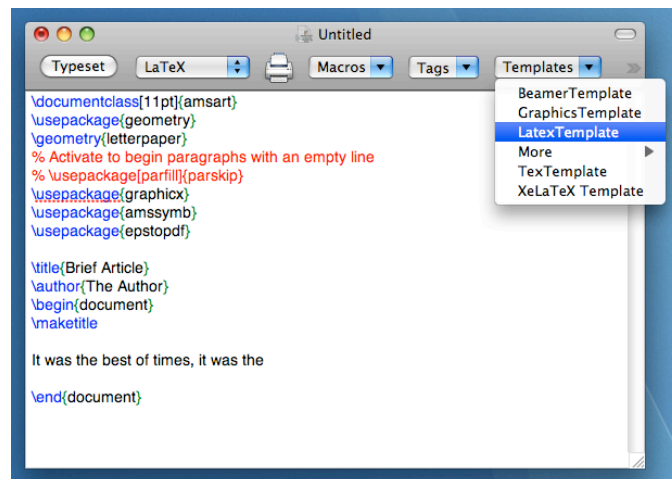
New material goes between the lines

    \begin{document}

    \end{document}

18

Type some sentences there now. TEX will ignore most carriage returns because it knows how to format text, so insert them randomly if you wish. Use a blank line to indicate the start of a new paragraph.

Uncomment the source line with the words "usepackage[parfill]parskip" if you prefer to separate paragraphs with an empty line instead of an indent.

When you have some material, hit the "Typeset" button at the top left of the window. A dialog will appear asking you to save the document. You can give the document a name in a field at the top; a field below that shows the folder where the document will be saved. At the right side of this field there is a small arrow. If the arrow points down, click it and the dialog will expand to a form which allows you to navigate to other locations and create new folders.

When TEX typesets, it creates three or four additional files, so it is not a good idea to save directly to a location with many other files. Instead, navigate to a reasonable location, say `~/Documents`, and then click the "New Folder" button at the bottom of the dialog. Accept the default name or choose another and create the folder. Then name the document and save it. The default "Untitled" name will do.

As soon as you save, TEX will typeset the document and open a second window showing the result.
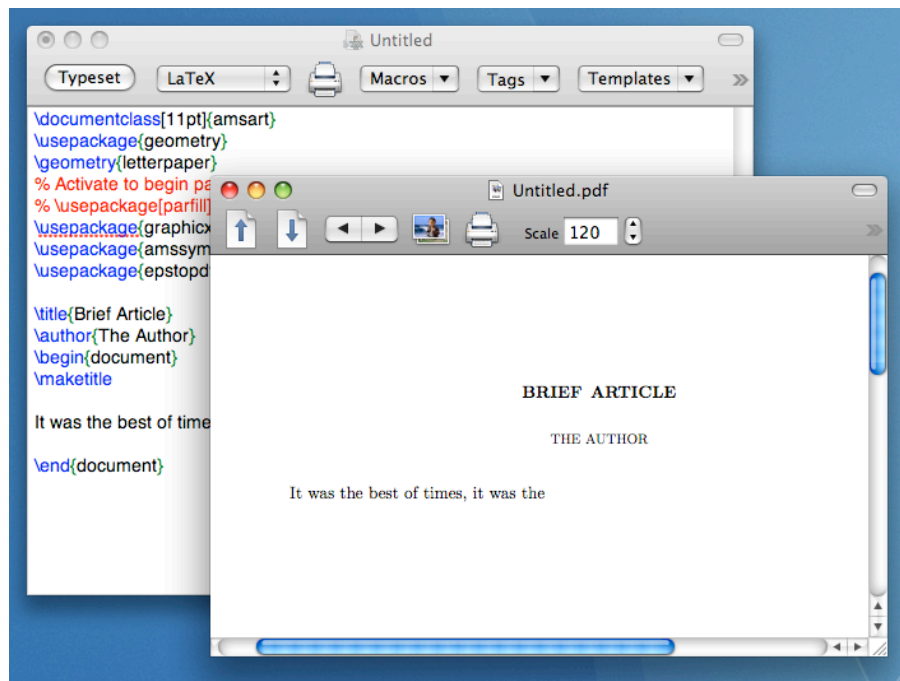
Go back to the original window and add some additional text. Hit "Typeset" again. This time TeX immediately typesets the new material and updates the output window.

Close the document by hitting the red button at the top of the edit window. There is no need to save the program first because TeXShop automatically saves regularly. Quit TeXShop.

Restart TeXShop by clicking on the icon in the dock.

It is possible to reopen the project by double clicking on the Untitled file or by starting TeXShop and navigating to Untitled in the Open dialog, but there is a much easier way. Start TeXShop and under the File menu choose the item "Open Recent". In the resulting list, choose the document's name and you will immediately return to it.

If you do not close the document before quitting TeXShop, then it may reappear automatically the next time you run TeXShop. This depends on a setting in Apple's System Preferences. Apple revised System Preferences in Ventura. If you are using an earlier system, run System Preferences and select the General module. If you are using Ventura, run System Preferences and select Desktop & Dock. In the resulting dialog, boxes appear labeled "Ask to keep changes when closing documents" and "Close windows when quitting an app". If both are unchecked, then TeXShop and most other programs will reopen windows automatically if they were open when the program quit.

One configuration step is highly recommended when you first run TeXShop. Open a project which has both source and preview windows. Resize and position these windows as you prefer. Most users position the source window on the left half of the screen and the preview window on the right half of the screen, with both covering most of their half of the screen, but the choice is up to you. Select the source window and at the bottom of the TeXShop "Source" menu select "Save Source Position". Select the preview window and at the bottom of the TeXShop "Preview" menu select "Save Preview Position." After this step is done, all TeXShop windows will open in these positions.

Those are the basics. You will find further information in the TeXShop Help menu. The item "TeXShop Demos" contains two short movies. The first illustrates starting and typesetting a short document, and the second illustrates the configuration just made.

3.2   A Few Unfortunate System Preferences

MacOS comes with a few system preference settings that complicate working with TeX. It is useful to reset them immediately.

In the Keyboard module of System Preferences under the Text tab on systems before Ventura, and in the same module in Ventura obtained by pressing "Edit" after "Input Sources", turn off the items "Correct spelling automatically", "Capitalize words automatically", and

"Add period with double-space". Automatic spell checking will remain on; the first item is more pernicious and can change text you type into completely different text. Moreover, it does not recognize TEX commands and will change them and thus break your source.

At the same spot, uncheck the item "Use smart quotes and dashes". TEX has an entirely different method of handling quotes and dashes, which this item will break.

In the General module on systems before Ventura and in the Desktop & Dock module in Ventura, the item "Prefer tabs:" presents a pull-down menu with the item "in full screen" selected. That choice is fine. The choice "always" may interfere with TeXShop's method of using tabs, to be explained much later.

3.3   Second Steps with TEX

The next crucial step is to learn TEX. The easiest way to do that is to pick one or two short introductions to TEX and work through them while trying examples in TeXShop. After a week or so working in this manner, it will be time to start using the program to write lecture notes and articles.

Ask friends where to start. Because it is how I started, I'd recommend Leslie Lamport's book *LATEX, A Document Preparation System*. Be sure to get the second edition, which will say "Updated for LATEX 2e". It would be enough to read chapters 2 and 3, a total of 52 pages.

TeX Live contains a free 139 page book on LATEX, *The Not So Short Introduction to LATEX2e*, by Tobias Oetiker, Hubert Partl, Irene Hyna and Elisabeth Schlegl. See `file:/Library/TeX/Root/texmf-dist/doc/latex/lshort-english/lshort.pdf`.

The link `https://learnlatex.org` leads to a series of online LATEX lessons, with interactive examples, which were partly funded by the TeX User Group in 2020.

Another source is the first portion of George Grätzer's book *More Math Into LATEX*. Grätzer gave permission to put this section in the TeXShop Help window, so you already have this as well.

After reading one of these short works, I'd recommend starting to use the program for serious writing. This will be frustrating at first, and you'll probably want to buy one of the excellent books on TEX for reference as you work. But don't ignore Google! A vast amount of information is available about TEX and LATEX by asking succinct questions of Google. Try "how do I type a url in LaTeX" if you don't believe me.

Chapter 4

Configuring TeXShop

4.1 TeXShop Preferences

Although Macintosh computers are often used by a single person, the operating system is designed for multiple use. If your machine is used by several family members, each with their own login name, then a family member can change TeXShop's Preference settings without disturbing the settings when you use the machine.

Each user's home directory contains a special folder named ~Library containing the configuration files for that user. Because this folder's contents are highly technical, Apple sometimes hides the folder. To further confuse matters, there is also a system library denoted /Library, which has nothing to do with user preferences. TeXShop has a menu item in the TeXShop menu to open the location `~/Library/TeXShop`.

TeXShop's Preference settings are in `~/Library/Preferences` in the file TeXShop.plist. It is never necessary to edit TeXShop.plist because the items in this file are set by the Preferences Dialog opened by an item in the main TeXShop menu. In very rare situations, TeXShop.plist can be mangled when TeXShop is updated to a new version. This seems to happen to only a few people, and only when new preference items are added for a new version. When the problem occurs, the updated TeXShop refuses to open any file, or displays other bizarre behavior. To fix the problem:

- Quit TeXShop.
- Go to ~/Library/Preferences and drag the item TeXShop.plist to the desktop.
- Restart TeXShop.
- In TeXShop's Preference panel, reset any items you may have changed.

4.2 ∼/Library/TeXShop

Other configuration files are in subfolders of the folder `~/Library/TeXShop`. Users will definitely want to edit some of these files. Some will never be touched; others will be explained shortly. Use the menu item **Open ∼/Library/TeXShop** to access these folders.

When TeXShop first runs, it creates the folder TeXShop and all of its subfolders. Users may then modify the contents of these folders. Occasionally, a user modification may hopelessly mangle the contents of a particular folder. In that case, move that folder to the desktop and then restart TeXShop. It will recreate the folder and give it default contents.

For example, the folder Macros contains the various macros available in the source window's toolbar. These macros can be edited using an editor built into TeXShop, as will be explained later. Suppose an ill-advised editing session results in corrupted macros. Move the Macros folder to the desktop and restart TeXShop. The Macros folder will be recreated with default values. Since the old Macros folder is still available on the desktop, you may be able to recover any uncorrupted changes you made earlier.

In most cases, updating TeXShop after it first runs does not modify folders because that would overwrite user edits and additions. A small number of folders *are* modified because they contain items which users can move into active locations, and we want these optional items to be up to date.

The following is a list of all folders which are rewritten during a TeXShop update:

- bin
- Documents
- Engines/Inactive
- ExternalEditorScripts
- HTML/Inactive
- New
- Scripts

4.3 Templates

Recall that the toolbar on the Source Window has a pulldown menu named **Templates**. Selecting one of these templates copies its contents to the current cursor location in the source. Each template is a standard .tex file, which can be created or edited using TeXShop. These files are stored in the Templates subfolder of `~/Library/TeXShop`. The Templates

folder can also contain subfolders containing template files. These become submenus in the Templates menu.

Templates can contain small snippets of code commonly used, like the code to include a graphic file, or the code to activate an interactive element in an TeX4ht document. But templates are more commonly used for the headers of standard types of documents. For instance, there can be headers for Articles, Books, Beamer Slides, Letters, XeTeX articles, LuaTeX articles, and so forth.

While TeXShop provides sample Templates, it is assumed that most users will create their own templates for articles and books. Your template should input all the standard packages you regularly use, as well as definitions and the like. As you work, you can easily add items to your standard templates to make them more robust.

Creating a new template with TeXShop is easy. If you like, start with an existing template like "LatexTemplate.tex". Duplicate it and rename the duplicate, leaving the new file in the Templates folder. Open this new file in TeXShop and modify it or add extra items. Done. The new template will automatically appear in the Templates pulldown menu.

Chapter 5

Editing Preferences

5.1   Preference Design

TeXShop has a Preference Panel, which can be opened using the **Preferences...** item in the main TeXShop menu. In macOS Ventura and later, this item is renamed **Settings**. The panel has tabs which display various related groups of preference settings. At bottom right are two buttons labeled **Cancel** and **OK**. It is necessary to push **OK** after making changes if you want these changes to take effect. Some changes take effect immediately, while others become active when documents are reopened.

TeXShop also has menu items to change its behavior. These items often duplicate items that can be set in Preferences. There is a big difference between setting a behavior with a menu item, and setting the same behavior in Preferences. *Modifications using menu commands affect only the active document, and only while it is open in the current session; modifications using TeXShop Preferences determine the default behavior when documents are first opened.*

For example, the Source menu has an item called **Show Syntax Color** and Preferences under the Edit tab has a check box labeled **Syntax Coloring**. The menu toggles syntax coloring on or off in the active source window. Select one of two source files, say file A, and toggle until syntax coloring is on. Select a second file B, and toggle until syntax coloring is off. Notice that each window remembers its state, so if you make file A active, it will be syntax colored, and if you make file B active, it will not be syntax colored. When file A is brought forward, the syntax color menu item is checked and when file B is brought forward, the syntax color menu item is unchecked.

The **Syntax Coloring** preference item does not change syntax coloring in these two files. Instead, it determines whether new documents will have syntax coloring or not when first

opened. Uncheck the item and press OK. Then open a new document and notice that it
has no syntax coloring.

5.2   Source Font and Font Size

In a few cases, the design philosophy described above is slightly modified. An example is
the method to set the font size and font in the source window. This is an important case,
so we'll completely describe it in this section.

Suppose first that you want to magnify the source font temporarily, perhaps to look care-
fully at a formula or perhaps to show it to someone looking over your shoulder. This is
easy. Activate the source window and then type command-plus to magnify the text and
command-minus to shrink the text. Here command-plus means that the command key
and the plus key should be depressed at the same time; command-minus means that the
command key and the minus key should be depressed at the same time. Although the plus
symbol is typed with the shift key, the shift key should not be used with this shortcut. No-
tice that only the active source window is affected by these changes. Moreover the changes
are temporary; if the document is closed and then opened, it will have the standard font
size.

These commands are actually menu shortcuts. The *Source* menu has a *Font* submenu,
which has two items *Bigger* and *Smaller*; the key shortcuts call these menus. However
making text bigger or smaller is a common task and the menu items are hidden away, so
it is best to just remember and use the shortcuts.

The remarks about keyboard shortcuts in the previous two paragraphs apply to a U.S. key-
board. Many users, perhaps even a majority, will select another keyboard in the Keyboard
Module of Apple's System Preferences (using the Input Sources item). These selections
map physical keys on the keyboard to characters, and can change menu shortcuts. To
define appropriate shortcuts for Font Sizes, read the section "Redefining Menu Shortcut
Keystrokes" in chapter 23.

The source font itself can be changed using a menu command, but this is almost never
done. Instead users make the change in Preferences so it will affect all future source
windows.

To change the font in the Source Window, open the *Source* tab of Preferences. Push the
Set button and the Font Panel will appear. Use this panel to select a new source font and
its size. What happens next is one of those slight modifications of our design philosophy
— the font and font size immediately change in all open windows. We do that so users can
immediately see the effect of the change and experiment until they get exactly the correct
behavior. Selection is not yet finished. In the end, push *Cancel* or *OK*. Pushing *Cancel*

reverts to the old font and size in all open windows, while *OK* accepts the change for all open windows and for windows opened in the future.

It is useful to choose a monospace font for the source file, since then all characters have the same width and list items and the like will line up.

A Tricky Special Case I once got a bug report from a user in Israel. He was writing a physics book in Hebrew. When the size of the book reached about 100 pages, TeXShop's editor slowed to a crawl. He would enter a couple of words and several seconds later those words appeared on the screen.

His source file was interesting. There were standard LaTeX commands in English starting from the left margin and flowing right, followed by Hebrew source starting from the right margin and flowing left. Each page contained many such switches in direction. While trying to debug the problem, I happened to change the source font and to my surprise the editor was suddenly snappy again. What happened?

The Macintosh offers many "virtual keyboards", which transform your existing keyboard to a different language. An item in the bar at the top of the screen can rapidly switch keyboards. He typed commands with the English keyboard, reached up and switched to the Hebrew keyboard, and typed the contents of the book.

Most standard Macintosh fonts do not contain Hebrew letters. But Apple's programmers know that and provide a special trick. When a user selects the Hebrew keyboard, the Macintosh silently switches to a font containing Hebrew. Thus to process his source file, the Macintosh was silently switching the source font many times on each page. When new text is entered into the source, some processing needs to be done to the entire source. So each new word required thousands of font switches.

The solution was simple: choose a source file which contains Hebrew. The Apple Font Book can list languages supported by each font on the system, and many of these fonts support a large number of languages, so finding a suitable source font was easy.

5.3 Other Source Preference Items

This is a useful time to consider a few other Preference items which affect the source window. These items occur under the first tab, Source. But several items in this tab are better explained later and will be skipped.

Document Font This was just explained.

Source Window Position This preference was set in chapter 3, when you were asked to fix the location and size of the source window and then select the item *Save Source Position*

in the Source Menu. That action selected the item *All windows start at fixed position.* I do not recommend the other choice, but try it if you wish.

On Startup If the first item is checked, TeXShop opens a blank document if it is opened without selecting a file. I recommend checking this when you first use TeXShop. Eventually it may become annoying and you can uncheck it. (External editors will be discussed in a separate chapter.)

Parens Targets & Highlight Color A preference item to be discussed in the next section briefly flashes an opening parenthesis when the corresponding closing parenthesis is typed. This preference determines which parenthesis pairs follow this rule. The behavior is useful for ( ), [ ], and { } pairs. It doesn't work well for angle brackets because inequalities don't produce matching pairs.

Find Panel (after restart)

TeXShop provides three different Find Panels. Select each and try it. After making a selection, TeXShop must be restarted before the new panel becomes active. The Apple Find Panel was the original choice in TeXShop twenty years ago. It is very basic. The Apple Find Bar is the newer method by Apple, used in Safari, XCode, and many other official Apple programs. It looks very minimal, but a close inspection shows that it has unexpected features. The OgreKit Find Panel is an open source project from Isao Sonobe in Japan. It is full featured and supports Regular Expressions. This is the panel that most users ultimately adopt. It is not the default choice when you first use TeXShop because the OgreKit Panel does not work well in the Chinese localization.

5.4 Other Editor Preference Items

The source window is also configured by various items in the Editor tab in Preferences. See the picture on the next page. We will discuss these items in a different order than listed in the Panel.

Several source window features can also be changed using menu items in the Source menu. We'll list these items first; the corresponding Preference items select the default behavior when a window is first opened. For example, the text in the source window is usually syntax colored, but a menu item can toggle this behavior for the active window. The **Syntax Coloring** Preference item determines whether syntax coloring is on when a window is first opened.

Line Numbers When selected, source lines are numbered.

Highlight Current Line When selected, the background of the line containing the cursor is tinted a light blue. Lines can cover several physical lines, ending when the user finally types a linefeed.

Syntax Color When selected, the source text is syntax colored.

Check Spelling When selected, misspelled words are underlined in light red.

Show Invisible Characters When selected, invisible characters like spaces, newlines, and tabs are shown. The preference allows users to pick which characters are shown and by what symbol. The final **On by default** item is usually left unchecked, so these characters are not shown when a window is first opened. If puzzling behavior later develops, it can be turned on using a menu item.

The remaining items are rarely changed, so they can only be set in the Preferences Pane:

Use Tabs for Indent Menu The third and fourth items in the TeXShop source menu are named *Indent* and *Unindent.* Select several source lines and then choose *Indent* to push these lines right by four spaces. Choose *Unindent* to push the lines back left. Indentation either adds spaces to the beginning of lines or else adds tabs. The preference item determines which are used. (Perhaps readers are familiar with the television program *Silicon Valley*, in which the hero loses a programming girlfriend because she uses spaces rather than tabs in her code.)

Select on Activate When checked, a click in the Source Window both selects the window and moves the cursor to that spot in the source. When unchecked, a click selects the window, but does not move the cursor.

Correct Spelling When selected, the Macintosh will not just underline misspelled words. It will also correct the spelling, and sometimes even change the wording. This item is present so the user can turn it off! We explain. A global preference turns correct spelling on or off for all applications. If this global preference turns it on, the **Correct Spelling** item can turn it back off in TeXShop while leaving it on in other applications.

The remaining items in the first column and the first block of items in the second column will be explained later in the manual.

Wrap Lines TeX usually ignores line feeds because it knows how to break lines. Some authors like to create a line feed near the end of each line, so their source contains a long list of relatively short lines. Other authors like to create a line feed only between paragraphs, so their lines span multiple lines of the window. In that case, TeXShop inserts a soft line break between words so the cursors jumps down to the next line when appropriate. This line break is just for the display and isn't sent on to TeX. Said another way, if you enlarge the window, the breaks will be recalculated; they aren't permanent elements of the source. Long ago, some users didn't want breaks except if they actually typed them, hence this preference setting. I have no idea why anyone would change the default value, which is **by word**.

Style The first item sets the width of a tab; by default, the value is 4. The next two items format paragraphs of source created by writing several lines without line feeds. By default, TeXShop starts all lines of such paragraphs at the left margin, but it is possible to indent the first line and not remaining lines, or start the first line at the margin and indent remaining lines or format in other ways.

The last item sets the spacing between lines, so the source can be single spaced or double spaced or somewhere between these extremes.

Preference values for various style fields are integers. These values are constrained in various ways. Tab indent can be between 2 and 50, paragraph indent can be between 0 and 100, and interline spacing can be between 1 and 20. When larger or smaller values are entered, they are replaced with the closest allowed value.

If a source window is open when the style items are set, the changes will appear immediately in the source, so users can experiment with values until they find a pleasing set. However, there is a mild complication. When number entry boxes are used in preferences, changes in these boxes are not sent to the Mac until the box is deactivated. The easy way to deactivate a box is to activate another box by clicking in it. At that moment, the source window will be updated to reveal changes. It is tempting to press RETURN to deactivate a box, but the Preference dialog reacts to RETURN by accepting all changes and closing the Preference window.

Aside: The first version of this manual had a different explanation for the style items. It read "The first item sets the width of a tab; by default, the value is 4. The remaining items are left from an earlier version and do nothing. There are no plans to revive them." To my surprise, a user wrote telling me that the remaining items actually worked, but TeXShop had to be restarted to see the changes. Sure enough. So I revised the items to immediately show their effect in source windows.

5.5   Parenthesis Pairs in the Source Editor

The source editor inherits the following useful behavior from Cocoa: if you click once in a spot, the insertion cursor is moved to that spot; if you click twice in a spot, the full word in that spot is selected; if you click three times in a spot, the full paragraph containing that spot is selected. To select a specific letter, click and hold the mouse down and slide slightly to pick up the letter.

TeX and LaTeX make extensive use of the parenthesis pairs (,) and [,], and {, }, so double clicking on one of these characters does something different; it selects the matching member of the parenthesis pair and all text between them. Often this is just a few words or symbols, but it can be enormous sections of text set off by the pair. You will often use this feature to troubleshoot TeX errors.

Parentheses cause particular trouble when entering mathematical equations. It is all too easy to enter the wrong kind of parenthesis, or to forget a matching element. So TeXShop has several features to help with this process. Some of these come from my wonderful collaborator Yusuke Terada. The items are governed by the **Parens Matching** item on the left side of the Editing Preferences pictured earlier, and the various **Parens Matching Settings** at the top right of that picture.

Here is a brief list.

Default If all of these items are unchecked, then when a matching parenthesis is typed, both it and it match are colored red. After the next character is typed, these elements are given their standard syntax colors.

Parens Matching. If this item is checked, the previous behavior continues to hold, but in addition the matching parenthesis is very briefly given a yellow highlight. The effect is fairly subtle.

The remaining items come from the right side. Some of these items replace the behavior just described, and a few only work if the option to highlight the current line is turned off.

Blink Highlight When a matching parenthesis is typed, the matching parenthesis noticeably blinks.

Highlight Enclosed Characters When a matching parenthesis is typed, the text between it and its match flashes red for a moment. If the two parentheses are widely separated, this can be quite alarming, so use this feature only if you know what you are doing.

Show Indicator in Moving When a matching parenthesis is typed, the entire text between it and its match turns red. Moreover the red remains, even if the source is scrolled, until another character is typed. This requires that the current line not be highlighted, that the **Highlight Enclosed Characters** option be checked, and that the **Blink Highlight** option be unchecked. This option is also alarming and most people leave it unchecked. But if two parentheses include a large amount of text and there is a strange error in that section, the option can be temporarily useful.

Beep for Isolated Parens Make a short beep sound when an unmatched parenthesis is typed.

Flash Back for Isolated Parens Briefly flash the entire screen when an unmatched parenthesis is typed. This can also be alarming, so use this feature only if you know what you are doing.

Chapter 6

Encodings

6.1 About Encodings

In TeXShop Preferences under the *Source* tab, the top item in the second column is labeled **Encoding**. This item is a drop-down menu listing 31 choices. The default choice is **Unicode (UTF-8)** and that choice will work fine for most users. If you are just starting to work with TeX, by all means select UTF-8.

If you have older files written using a different encoding, or if other people send files in unusual encodings, it is important to know more.

A computer file contains a long stream of whole numbers, each between 0 and 255. These are called bytes, and the ability to encode virtually any kind of information into a stream of bytes defines the current digital age. Thus a byte stream might represent music on a CD, or a movie on a DVD, or a jpg picture, or a computer program, or an encyclopedia. Many computer files are ordinary text files. From the beginning of the personal computer era, text has been encoded into bytes using the ASCII (American Standard Code for Information Interchange) encoding scheme. This method encodes all the characters on a standard American typewriter: small letters, capitol letters, punctuation marks, numbers, tab, carriage return. There are less than 128 such characters, so ASCII only uses bytes between 0 and 127. In the original version of TeX, source files were standard ASCII files.

ASCII is not sufficient in Europe and in regions of the world that use completely different scripts. Scripts in Western Europe use accented vowels, umlauts, upside down question marks, and unusual Scandinavian letters. Russia and other countries use the Cyrillic alphabet. Greek is written with the Greek alphabet.

To solve such problems, programmers assigned the unused 128 entries to new characters.

Different encodings were invented for each country, each with special characters in those 128 spots. One of these encodings, IsoLatin1, contained all of the characters routinely used in Western Europe. When the European currency was introduced, IsoLatin1 was extended by adding the Euro symbol, to become IsoLatin9, and for a time TeXShop adopted that as default. Over the years, users requested additional encodings in TeXShop. This eventually led to the list of 31 encodings in the current version.

Sadly, files encoded in this way do not contain a "magic byte" defining the encoding. So a user has to remember which encoding was used to write each file, because the computer has no way to know.

But meanwhile, computer manufacturers realized that the increasing globalization of the world required a new approach to text. They formed an independent organization, which created Unicode, a standard that can represent all of the scripts of the world, including hieroglyphics, Arabic, Chinese, and mathematics. Virtually all computers have switched to Unicode; it is a central part of macOS. TeXShop is entirely Unicode-based internally. You can easily type English, Cyrillic, Chinese, Hebrew, and Arabic in a single TeXShop document; the Hebrew and Arabic will even be written from right to left.

Unicode does not define a standard method to read and write Unicode to a file. But one very popular method is called UTF-8 (Unicode Transformation Format, 8-bit). This method has the advantage that ordinary ASCII is correct UTF-8 output. The remaining unicode symbols are coded using an elaborate scheme we need not explain.

Because there is no standard way to write Unicode to disk, every routine in macOS to read or write text requires an *encoding parameter*, which describes the type of encoding to be used by the operation. If this parameter is UTF-8, then the contents of the editor will be completely preserved. If the parameter is IsoLatin9, then ASCII and Western European characters will be preserved, but Chinese, Arabic, Cyrillic, etc. characters will be lost.

There is one crucial difference between most of the encodings and UTF-8. Since most encodings just define characters for the bytes between 128 and 255, any random stream of bytes is an acceptable file. But UTF-8 files use coded input, so random streams will contain illegal bytes and a computer asked to read such a stream will reject the entire file. If that happens, TeXShop will put up an error dialog, and then open the file in IsoLatin9.

Until recently, IsoLatin9 was the default encoding assumed by LaTeX. It could accept files encoded in other ways, but then a command in the source had to state the encoding. Two years ago, LaTeX switched to using UTF-8 as the standard encoding.

Users new to TeX should use UTF-8 encoding. Then anything typed in the source window will be preserved when they save the file. Moreover, modern typesetting engines based on TeX assume UTF-8 input. Luckily, very old sources from books and CDs will also work

because the original TeX used ASCII and pure ASCII is UTF-8.

But people who have been using TeX for a number of years are going to have files saved with a different encoding. If the text is in English, the only problematic section may be the bibliography.

TeXShop has several features to help users deal with encoding issues.

Open Dialog If the *Open...* item is used to open a file, a dialog appears allowing users to navigate to the file. At the bottom of this dialog, a pull-down menu can select the encoding used to read the file. By default, this will be the encoding selected in TeXShop Preferences, but it can be changed. In Ventura, a button named *Options* must be pushed to display the pull-down menu.

Magic Comment Line A comment line can be added to top of a source file listing the encoding that should be used to read and write that file. This line has the following form

```
% !TEX encoding = UTF-8 Unicode
```

The line overrides all other methods used to determine the encoding. If you have a source written in an unusual encoding, you can insure that TeXShop always uses the correct encoding for that particular file. The magic line must be in one of the first 20 lines in the source file, must start at the left margin, and have exactly the form shown above. For instance, there must be exactly one space on both sides of the equal sign, and the name of the encoding must be exactly that used by TeXShop, including capitalization. Luckily, there is an easy way to achieve all of these conditions. In the TeXShop *Macro* menu, there is an item named **Encoding**. Select it and a list of known encodings will appear. Select the one you want, and the magic line will be added to the top of the source file.

Save Dialog Versions of TeXShop before 5.04 had an encoding menu in the Save Dialog, allowing users to select the encoding used to write the file. By default, this was the encoding used when the file was opened, so the menu was not needed in most cases. To avoid misuse, the menu was removed in TeXShop 5.04. If you updated from an earlier version of TeXShop and routinely used the menu, you can restore it using a hidden preference item. Open Terminal in /Applications/Utilities and type the following line:

```
defaults write TeXShop EncodingMenuInSaveDialog YES
```

I do not recommend this. See the following section for details.

6.2 More on Encoding Issues

Recall that I live in the US, where ASCII is often sufficient. Rather than using this manual, it is best to ask TeX experts in your area about encoding matters.

But I can give a minor piece of advice. Many people in the U.S. and Western Europe have used either UTF-8 or IsoLatin9. If you have a mixture of such files, one way to proceed is to switch the default encoding to UTF-8. If TeXShop opens an old document without complaint, then either it is UTF-8 or it is standard ASCII, and at any rate the encoding is correct. Otherwise TeXShop will complain and open the document in IsoLatin9. But then you know for sure that it is encoded in IsoLatin9. So use the TeXShop Macro menu's "Encoding" item to insert the magic line

```
% !TEX encoding = IsoLatin9
```

at the top of the file. From then on, the file will open without complaints, and you will know for sure what encoding was used. Since LaTeX nowadays uses UTF-8 by default, the following lines should be added to the header of such IsoLatin9 documents:

```
\usepackage[T1]{fontenc}
\usepackage[latin9]{inputenc}
```

6.3   Changing Encodings of Files

Changing the encoding of a file is tricky. Do not attempt it without backing up the original version of the file. TeXShop users have sometimes tried to change from one encoding, A, to another, B, by using the open dialog's encoding menu to open the file with encoding A, and then using the save dialog's encoding menu to save the file with encoding B. Assume that neither file is UTF-8 and the file is currently encoded with encoding A. When TeXShop opens the file, it converts all symbols to their appropriate Unicode form. Then when TeXShop saves the file, it converts all symbols contained in B to their appropriate B form, and discards all other symbols. So symbols in both A and B are correctly moved, while symbols only in A are discarded. Since the same file was opened and saved, the original form of the file is lost.

Life is even trickier if the encoding of a file is unknown. Suppose a file was written with encoding A, but opened with encoding B, and then converted to encoding C. Suppose A encodes α as character 200. Suppose B encodes α as character 210 and encodes β as character 220. Suppose encoding C encodes α as 200 and does not encode β. Then when the file with encoding A was opened with encoding B, the α became a β, and when this file was saved with encoding C, this β was dropped, even though all three encodings encode α.

There is a command line program which can convert from one encoding to another, *iconv*. This program gives different names to the input and output files, so the input is preserved. Use it instead of TeXShop if you must convert.

Chapter 7

Typesetting Engines I

7.1 History

When TeXShop was released in 2001, it could only typeset using TeX or LaTeX. The actual typesetting was done by *pdftex*, an extension of TeX which directly output a pdf file rather than a dvi file.

A few months after the TeXShop release, a user suggested offering users an alternate way to typeset. Instead of calling pdftex to create a pdf file, TeXShop could call TeX to create a dvi file, and then call dvips to convert this dvi file to a postscript file, and finally call ghostscript to convert the ps file to pdf. Except for the last step, this is how TeX and LaTeX originally typeset.

The advantage of offering both typesetting methods was that pdftex can accept pdf, png, and jpg illustrations, but not eps illustrations. The dvips chain can accept eps illustrations, but not pdf, png, and jpg illustrations. In addition, there is a package called *pdftricks* which directly inserts postscript into the output file. This only works when the dvips route is used to typeset. Finally, back in 2001 some publishers required authors to submit a dvi file, and such a file was created by the second chain.

Nowadays, typesetting with the second method is only necessary if you use pstricks. Starting in 2011, TeX Live automatically converts eps illustrations to pdf illustrations during typesetting, and publishers almost never ask for a dvi file.

So since 2001 TeXShop could typeset in four ways: TeX with pdftex, TeX with the dvips chain, LaTeX with pdftex, and LaTeX with the dvips chain. Perhaps as much as 70% of TeX typesetting today still uses one of these original toolchains, so this chapter will explain how to configure and use TeXShop in one of these ways.

7.2 Configuring TeX and LaTeX Typesetting

Go to TeXShop Preferences and select the **Typesetting** tab. In the first column, select either TeX or LaTeX as the default command. Most people use LaTeX, so that the is default choice. We'll discuss the third option later.

Below this item, select either "Pdftex" or "TeX + DVI" as the default script. Most people select "Pdftex". If you use pstricks or have many very old documents, "TeX + DVI" may be a better choice. We'll discuss the third item later.

In the second column of the same tab, select "SyncTeX" as the Sync method. The other choices are there for historical reasons and can be ignored.

After making these choices, typesetting with TeXShop is easy. When you want to typeset, just push the "Typeset" button in the Source toolbar or select the "Typeset" menu at the top of the Typeset menu. This menu has a keyboard shortcut, command-T. After a few days, you'll ignore the toolbar item and the menu item and type command-T when you want to typeset.

To switch between "Pdftex" and "TeX + DVI" for an occasional document, go to the Typeset menu and select the appropriate option at the bottom. This only affects the active document, so if you have two documents open, you can switch the typesetting method for one without changing it for the other.

7.3 Bibtex

The Typeset menu contains two other items, "BibTex" and "MakeIndex". These are explained in this section and the next section.

BibTeX is a tool used to create bibliographies in LaTeX documents. The tool assumes that you have a large database of references which you often quote. Simple commands allow you to cite certain of these references in your LaTeX source, and BibTeX then creates a bibliography for your article containing only the items cited.

Here is a brief example taken from The LaTeX Companion by Goossens, Mittelbach, and Samarin. Consult this book for many additional details. Suppose the database is a file called "mybibliography.bib" containing the text shown on the next page. In this text, the entries "Felici:1991, Knuth:WEB," and "Liang:1983" are key values used to cite the articles in the LaTeX source.

```
@article{Felici:1991,
author ={James Felici},
title ={{PostScript versus TrueType}},
journal ={Macworld},
volume =8, pages={195--201},
month =sep, year = 1991 }
@techreport{Knuth:WEB,
title ={{The \textsf{WEB} System of
Structured Documentation}},
month =sep, year = 1983,
author ={Donald E. Knuth},
address ={Stanford, CA 94305},
number ={STAN-CS-83-980},
institution={Department of Computer
Science, Stanford University} }
@phdthesis{Liang:1983,
author ={Franklin Mark Liang},
month =jun, year = 1983,
school ={Stanford University},
address ={Stanford, CA 94305},
title ={{Word Hy-phen-a-tion by
Com-pu-ter}},
note ={Also available as Stanford
University, Department of
Computer Science Report
No. STAN-CS-83-977} }
```

Suppose the LaTeX source file is called "myfile.tex" and contains the following text:

```
\begin{document}
Consider the argument of Felici~\cite{Felici:1991} in light of these comments.
\nocite{Liang:1983} We provide further remarks later.
\bibliographystyle{plain}
\bibliography{mybibliography}
\end{document}
```

When this source is typeset, a reference to Felici's article will appear in the text, and a bibliography will be created at the end of the text containing the articles of Felici and Liang, but not the article of Knuth.

TeXShop can be used with this example in the following way. First edit and typeset the document "myfile.tex" as usual. Citations will appear in the output as "[?]" and the bibliography will be missing. Then select "BibTeX" under the Program button and run

BibTeX. Next select "LaTeX" and typeset again. Citations will still appear as "[?]", but the bibliography will be added to the output. Typeset a final time, and citations will have their correct values.

Notice that the BibTeX menu command has a keyboard equivalent.

The file "mybibliography.bib" can be opened and edited by TeXShop. If you use TeXShop to create the file "mybibliography.bib" in the first place, use the pulldown tag labeled "file format" to save the file as a bib file rather than as a tex file.

In recent years, additional programs like biber have been written to handle bibliographies. Japanese typesetting is done using a modified TeX and still other programs like pbibtex and ubibtex are available there for bibliographies. So TeXShop has a preference setting to set the program called when the BibTeX menu item is chosen. It is at the bottom of the Engine tab. Fill in with the appropriate name, and any flags that should be used when calling this command line program.

Often users use different bibliography programs for different projects, or when working with different coauthors. TeXShop allows the bibliography program to be set on a file-by-file basis. To set it for a given source file, include the following line within the first twenty lines of the source, replacing "biber" with the desired program names and any desired flags.

```
% !BIB TS-program = biber
```

For compatibility with other platforms, the following syntax is also acceptable

```
% !BIB program = biber
```

If this line is missing from the source file, the corresponding Preference entry will be used when the BibTeX menu command is selected.

7.4 MakeIndex

MakeIndex is a tool used to add an index to a LaTeX document. To create such an index, you should add the command below to the top of the source file:

```
\usepackage{makeidx}
```

Put the command below in the document preamble:

```
\makeindex
```

Put the following command where the index should appear, often just before "enddocument":

```
\printindex
```

Put index references in the source document, as in the example below:

```
There are many animals in the world\index{animal}.
Examples include bears\index{animal!bear}
and tigers\index{animal!tiger}.
```

To create the index, typeset the document as usual. Then run MakeIndex. Then typeset again.

Many additional details can be found in The LaTeX Companion by Goossens, Mittelbach, and Samarin.

7.5 Other Typesetting Preference Settings

If you installed using MacTeX or BasicTeX, you can ignore this section because TeXShop is preconfigured for those distributions.

Otherwise open Preferences and select the **Engines** tab. The first item in this dialog is a full path to the TeX Distribution command line programs. When TeX Live is installed, it creates a symbolic link to these binaries with path /Library/TeX/texbin. That is the default value for the preference. An advantage is that when TeX Live is updated for a new year, the link is also updated and thus the preference setting does not change.

If you installed TeX using MacPorts, Homebrew, or MikTeX, the location of the binaries will be different. Consult the documentation for these distributions to discover the required value.

The second item in the tab is a full path to the Ghostscript binary. This is /usr/local/bin for the copy of Ghostscript installed by MacTeX. The standard build sequence for Ghostscript will also install it at this location. If you use the Ghostscript in MacPorts or Homebrew, consult their documentation for the proper location. Many typesetting engines do not use Ghostscript, so this is not a crucial preference item.

The Alternate Path preference will be explained later and is not used by the toolchains discussed in this chapter.

Below these items, the command line calls are listed for each of the four typesetting methods discussed so far. Suppose you have a LATEX source file named Galois.tex and you have opened Apple's Terminal program and changed to the directory containing this source. You could typeset the source file with pdflatex by typing the following line in Terminal:

```
pdflatex --file-line-error --synctex=1 Galois.tex
```

TeXShop calls the typesetting engine in exactly the same way. These four preference settings list the appropriate four calls.

The pdftex defaults use two flags. The first, –file-line-error, tells pdftex how to display errors in the console; TeXShop uses this to take you directly to the source lines with the reported error, as explained later. The second flag, –synctex=1, activates code by Jerome Laurens which tells pdftex to generate sync data during typesetting. TeXShop later uses this data to sync between spots in the source file and corresponding spots in the preview output, as explained later.

There is one other flag you might want to add, –shell-escape. This flag gives pdftex permission to call other programs during typesetting. If you typeset source files which other people send you, this could be dangerous because pdftex will call any program requested, even a program which erases your entire home directory. So we recommend avoiding the flag. But if you use it, you might want to check the item **Shell Escape Warning** underneath. After that, when you typeset a file for the first time, a warning dialog will appear reminding you of "shell-escape" and asking if you still want to proceed.

Typesetting with "TeX + DVI" calls a script named "simpdftex" which is part of TeX Live. If you want to know how the script works, open it in TeX Live and read it.

Chapter 8

Typesetting Engines II

8.1   More History

The design discussed in the previous chapter uses built-in code to call one of four programs: TeX, LaTeX, BibTeX, or MakeIndex. Around 2004, it became clear that this design was inadequate because other typesetting programs exist: XeTeX, LuaTeX, ConTeXt, TeX4ht, and on and on. Consequently, a new approach to typesetting was adopted. In a perfect world, the methods described in the previous chapter would have been ripped out, but I didn't want to disturb previous users. So the old and new methods are both present, but users should concentrate on the new methods. All typesetting, including the original TeX and LaTeX support, fits naturally into the new design.

The basic idea of the new design is simple. Each typesetting job is determined by a small shell script, which can be created and edited directly in TeXShop. This script might be very simple and call a single TeX binary. Or it might be complicated and call several programs one after another, say pdflatex followed by bibtex followed by two more calls to pdflatex.

Typesetting scripts are called "engines" and are stored in ~/Library/TeXShop/Engines. Next to the typeset button in the TeXShop source window is a pulldown menu listing all available engines. Select one and then press the Typeset button to typeset with that script.

It is also possible to name the script which should typeset a given source file in a magic line at the top of the source. A typical magic line might read

```
% !TEX TS-program = LuaLaTeX
```

Then command-T applied to this source would typeset with LuaLaTeX.

The event that triggered this new design was the release by Jonathan Kew of XeTeX. His version of TeX accepted UTF-8 Unicode input and thus could typeset source written in any script: Chinese, Japanese, Korean, Hebrew, Arabic, and many, many others. XeTeX also supported using ordinary Macintosh fonts alongside fonts in the TeX distribution. This was important for Unicode support since Macintosh fonts were available for most languages. Finally, some languages are difficult to typeset because of complicated rules governing the script. For example, Arabic is written in cursive and letter forms can vary depending on whether a character is in the middle of a word or the end. To support such languages, XeTeX used AAT, Apple Advanced Typography. A few years later, XeTeX was was extended to work on all platforms and AAT was replaced by equivalent open source libraries for all platforms.

8.2 Engines

In the TeXShop menu, select the item "Open ∼/Library/TeXShop" and in the resulting Finder window open the Engines folder. You will find several files with extension ".engine". Each engine file contains a list of typesetting instructions for TeXShop. Each engine can be opened and edited in TeXShop. If you want to create a new engine, the easiest way to do so is to duplicate an existing engine, rename it, and then open and edit that file. Engine files must have their execute bit set, and this procedure does that automatically.

Open the file XeLaTeX.engine. Its contents are shown below.

```
#!/bin/tcsh

set path= ($path /Library/TeX/texbin /usr/texbin /usr/local/bin)
xelatex  -file-line-error -synctex=1 "$1"
```

This file is what is known as a "shell script." It is a series of commands which could be typed in Terminal. This particular script starts with a line telling Terminal to run the tcsh shell. Various shells are available, including zsh, bash, tcsh, sh, and others. Each has its own particular language. Recently, Apple selected zsh as the default shell, so we recommend using that shell.

Then the script tells Terminal where to search for TeX binaries. This may not be necessary since most of these locations are already in a user's path. The key line is the final one. It tells the system to typeset the file "$1" using xelatex with our two standard flags. The only difference between using this engine and using the built-in LaTeX command is that this engine calls xelatex rather than pdflatex.

When TeXShop calls a shell script, it executes "cd" to change to the directory containing the source file, and then it sends the shell script several pieces of information, which the script can access as "$1", "$2", etc. The most important of these is "$1", which contains

the name of the source file, including extension. So if TeXShop typesets Galois.tex, which is found in the folder ∼/Sources/GaloisStuff, the directory will be

```
/Users/koch/Sources/GaloisStuff
```

and "$1" will equal

```
Galois.tex
```

In a later section we will have more to say about creating engine files.

8.3   Typesetting with an Engine

Active engine files are listed in the pull-down menu to the right of the Typeset button in a source window toolbar. To typeset, choose the appropriate engine in the pulldown menu and then either push the typeset button or type command-T.

If most of the time you typeset with a particular engine, you can make it the default engine. In TeXShop Preferences under the Typesetting tab, the first entry selects the default typesetting command. Select the item **Command Set Below** and type the desired engine name. This name must have exactly the capitalization and spelling of the desired engine, without the "engine" extension.

The pulldown menu changes typesetting only for the source in its window. You can have two files open, one typeset with your default engine and the other typeset with xelatex. You only need to select xelatex the first time you typeset that file.

There is another way to choose the typesetting engine for particular source files. Near the top of the source, include a magic line

```
% !TEX TS-program = XeLaTeX
```

Then this particular source will always be typeset with the listed engine.

This command must start at the left margin and be written exactly as above, and the name of the desired engine must have the correct spelling and punctuation. There is an easy way to achieve all of this. Select the macro named "Program" from the Macro menu. A dialog will appear listing all active engines. Select the desired engine. The appropriate line will be added to the top of the source file, replacing a previous engine determination if one exists.

If you get in the habit of adding magic lines to the top of sources files, then you can use a variety of engines to typeset and you'll never have to think about which is appropriate. Simply type command-T and the correct choice will automatically be made.

In all of these places you select one of the four original typesetting methods using the following scheme:

tex	TeX with TeX and DVI
pdftex	TeX outputting pdf
latex	LaTeX with TeX and DVI
pdflatex	LaTeX outputting pdf

In rare cases, you might want to invent your own engines for some of these four built-in methods; that will work fine, just don't use the four specific names on the left side of the above list.

That is all you need to know to use typesetting engines. The rest of the chapter will discuss particular engines provided by TeXShop, and then explain how you can construct your own engines. Less than half of TeXShop users construct new engines, so you can skip that step until you need something unusual.

8.4   pdflatexmk

TeX is a single-pass compiler; it reads the source file just once as it typesets. But some typesetting tasks cannot be finished in a single pass. If a book has a table of contents, TeX cannot create the table of contents until it knows the names of the book chapters and the pages where they start. To solve this sort of problem, users typeset twice. In the first pass, TeX writes a file called the "auxiliary file" to the source directory; in the second pass, this file is consulted to find the required extra information.

Sometimes other programs are run between the passes. If a file has an index, an idx file is constructed during the first pass. Then the program "makeindex" is run to create an index from the information in this file. Finally TeX is run again to incorporate this index.

Some tasks require three or more passes through TeX before the job is complete.

There is a script in TeX Live named **pdflatexmk**, by John Collins, which automates this process. It calls LaTeX as many times as necessary to produce an up to date document, with intermediate calls to other programs if necessary. My colleague Herbert Schulz is an expert on this script and works with Collins to insure that it is well-supported on the Mac. He has created an engine which calls pdflatexmk to typeset. Many TeXShop users make this their default typesetting engine. The extra typesetting time with smaller projects is almost unnoticeable. The extra work on large books is essential, because the author can then preview a complete version of the work.

The pdflatexmk engine is already a possible choice when you first install TeXShop. The subfolder Engines/Inactive/latexmk contains several documents by Herbert Schulz with

more details, and related engines if you want to replace pdflatex with XeLaTeX or LuaTeX or other typesetting methods and use the script.

If typesetting speed is an issue, TeXShop has an obscure trick to help. While a document is being written, use pdflatex, which only typesets once. But every so often, switch to pdflatexmk so all features are correct. To make this easier, there is a special typesetting shortcut, control-option-command-T, which typesets with pdflatexmk. Therefore you can set the typesetting engine to pdflatex, typeset with command-T as a rule, but typeset with control-option-command-T occasionally.

This trick fails if you typeset with unusual engines. For example, suppose you typeset with XeLaTeX rather than pdflatex. Then there is a special engine in Engines/Inactive/latexmk which calls XeLaTeX rather than pdflatex. So the special command for control-option-command-T should call that XeLaTeX form of pdflatexmk rather than ordinary pdflatexmk. In TeXShop Preferences under the Misc2 tab there is a setting named "Alternate Engine" allowing pdflatexmk to be changed to any other desired engine, including the special XeLaTeX version of pdflatexmk.

8.5 The Inactive Folder

The Engines folder contains a subfolder named **Inactive**. This folder contains a large number of subfolders with additional typesetting engines for special workflows. The folders often contain documentation and some contain sample documents.

When TeXShop is updated, the files directly in Engines are not touched because users may have modified these files. But the Inactive folder is completely rewritten, so this material is as up to date as possible. This is important because some engines support Sage, ConTeXt, PreTeXt, and others projects under active development, where major changes can occur.

8.6 Creating a New Engine

The rest of this chapter is about creating engines. You may want to skip this material until you need a special engine.

Engine files must have extension ".engine" and have the execute bit set. The easy way to meet this requirement is to duplicate an active engine, rename it, and open and edit it in TeXShop.

Engine files are shell scripts, and the first line should name the appropriate shell. A sample line might be

```
#!/bin/zsh
```

Since zsh is the current default shell in macOS, I'll use that in this description, but macOS includes many shells including sh, tcsh, bash. Use the shell you know best.

I'll confess that I'm not a shell expert, so I often have to consult other engines to see how to accomplish various tasks. Google is also a useful source of help. The main tricky point is that shells use different syntax to accomplish the same task, so a line which works in one shell may fail in a different shell.

When an engine is called by TeXShop, it is given the name of a source file, say Galois.tex. Each line of the engine is then executed. After all lines are executed, TeXShop looks in the source folder to see if there is a file with the same name as the source, but extension ".pdf". For instance, it might look for Galois.pdf. If this file exists, it is opened in the Preview window. Otherwise nothing happens. This default behavior can be changed, as explained later in this section.

When engines are called, TeXshop gives the engine three pieces of information. Each is a string, denoted "$1", "$2", "$3". The first, "$1", is the name of the source file. For instance, it might be Galois.tex. The other two parameters are often not used, but might be employed in special circumstances. The parameter, "$2", will usually be just a space " ". But if further information is needed by an engine, it can be sent using a magic comment line near the start of the source file. This comment can have one of the following forms:

```
% !TEX TS-parameter =
```

```
% !TEX parameter =
```

The equal sign must be followed by a space, and then a word or sequence of symbols without spaces. If a project has a root file and several include files, the magic comment must be near the top of the root file.

The magic line might be used to pass one or more flags to engines. For instance, Herbert Schulz used this line to add a –shell-escape flag when pdflatex needs to call an external program during typesetting.

Finally, "$3" contains a path to the TeX binaries. Therefore "$3" will usually contain "~/Library/TeX/texbin". However users who install TeX using MacPorts or MikTeX or something else will have set a different location for binaries at the top of the Engine tab in TeXShop Preferences, and then "$3" will contain that location. For the cases of TeX Live, MacPorts, and MikTeX, this parameter is not needed because the Cocoa call to a binary contains its location. So "$3" is usually not mentioned in an engine.

But there is one special situation when it is needed. ConTeXt is a set of macros by Hans Hagen providing a different approach to typesetting. It is usually used with LuaTeX, another version of TeX. The ConTeXt community has its own distribution of both ConTeXt

and LuaTeX, which is small and updated often. So TeXShop has a preference item under the Engine tab named **Alternate Path** listing the location of this distribution. The ConTeXt users then add a magic comment to the top of their source code

```
% !TEX useAlternatePath
```

which causes TeXShop to set "$3" to this alternate path. The ConTeXt engines use this information to find the appropriate ConTeXt binaries in their special distribution.

8.7   Useful Engine Commands

When writing an engine, it is sometimes useful to decompose the string "$1". Imagine this string equals "Galois.tex". It can be useful to strip off the extension and obtain "Galois", and then add a different extension, like "Galois.pdf" or "Galois.aux".

Here are recipes to do this using the shell zsh. These recipes may or may not work in other shells.

```
#!/bin/zsh

filename="$1"
basename="${filename:r}"
extension="${filename##*.}"

echo $filename
echo $basename
echo $extension
echo $basename.pdf

if [[ "$extension" = "ptx" ]]; then
 echo "This is a PreTeXt file"
 exit
fi

if [[ "$extension" = "tex" ]]; then
 echo "This is a TeX file"
 exit
fi
```

If this script was used as an engine script and the file Galois.tex was typeset, no typesetting would occur but the console would display the following messagesa

```
Galois.tex

/Library/TeX/texbin
Galois.tex
Galois
tex
Galois.pdf
This is a TeX file
```

The second line is blank because $2 contains nothing. The word filename is a variable, defined and given a string value in the script, and $filename is the value of that variable. Thus $filename, $basename, and $extension are the full name of the source with extension, the name without extension, and the extension. The string $basename.pdf equals the old basename with new extension ".pdf".

Unfortunately, shell syntax is very picky. The line

```
filename="$1"
```

will not work if there are spaces around the equal sign, and the line

```
if [[ "extension" = "ptx" ]]; then
```

will not work if the spaces just after [[or just before]] are omitted. I am not a shell script expert, so I tend to copy lines from other scripts, consult Google often, and pay attention to spaces in the examples I copy.

8.8   Recent Additional Engine Commands

In TeXShop 5.00, an additional html-Preview window is provided, so typesetting jobs which output html can be previewed live on the web. Engines can now determine exactly which files are opened in preview windows after typesetting, using the new commands

```
!TEX-noPreview
!TEX-pdfPreview
!TEX-htmlPreview
!TeX-bothPreview
!TEX-noConsole
```

These are analogues of the magic comment lines used in TeXShop source, but there are a few differences. First, the lines can appear anywhere in an Engine file, because Engine files are short and it costs nothing to search the whole file. Second, the magic lines do not have to start at the beginning of a line; they can appear anywhere in the text.

It is crucial, of course, that the lines look like comments to the shell interpreter. So for standard engines, each line should be prefaced with the # comment symbol. But engines can be written in a variety of scripts, and some use different comment symbols. Therefore the comment symbol is not part of the new commands.

The standard behavior of TeXShop still applies if an engine has none of these lines. After executing all lines of the engine, TeXShop looks in the document directory to see if there is a file with the same name as the source, but extension ".pdf". If so, it opens this pdf file in the Preview Window, unless the Option Key is depressed.

If one or more commands are present, then

- if the line "!TEX-noPreview" is present, no preview window appears even if some of the other lines are also present;

- otherwise if the lines "!TEX-pdfPreview" and or "!TEX-bothPreview" are present, TeXShop searches the document directory to see if there is a file with the same name as the source, but extension ".pdf"; if so, it opens this pdf file in the Preview Window;

- if the lines "!TEX-htmlPreview" and or "!TEX-bothPreview" are present, TeXShop searches the document directory to see if there is a file with the same name as the source, but extension ".html"; if so, it opens this html file in the HTML Window;

- and if the line "!TEX-noConsole" is present, the console is not shown.

It is unlikely that this final line will be used with any engine except the one which directly opens html code without any intervening typesetting.

Chapter 9

Projects with Multiple Input Files

9.1   A Magic Line

Many book authors arrange their sources for a book as follows. A preliminary "root document" contains the preamble, defining macros, loading various packages, , and so forth. Then this document "includes" or "inputs" separate source files for each chapter. Some authors put these chapter sources in separate folders which also contain the illustrations for that chapter. This organization dates back to Lamport himself, whose famous manual on LaTeX contained the "include" command and an "includeonly" command, so authors working on a subset of chapters could only typeset those chapters to save time, and then typeset everything at the end.

Not all book sources are organized this way. I know several authors who created a single source file for a 700 page book. But the organization is not unusual. When it is used, TeXShop creates a separate window for the source of each chapter, opening only the window for the chapter currently being written. But the typeset button on such a window cannot just typeset the source in the window because that source has no preamble. Instead, it should typeset the root document, and create a single preview window showing the entire book, or possibly only chapters listed by "includeonly".

TeXShop has a simple way to make this happen. Each chapter file (that is, each source file to be included or input by the root file) should have a magic line at the top listing the name and location of its root file. The root file should not have this magic line. When a chapter file active and the author types command-T, the main root file should be typeset. The magic line, to appear in the first twenty lines of the source, has the following form, where in this case Final.tex is the root file.

```
% !TEX root = /Users/koch/TeX/MyBook/MyRoot.tex
```

The first letters of this magic line must be exactly as written above. This name of the root can have several forms illustrated below. Here MyRoot.tex should be replaced by the name of the root file.

```
MyRoot.tex
../MyRoot.tex
../../MyRoot.tex
/Users/koch/TeX/MyBook/MyRoot.tex
```

Use the first form if the root file is in the same folder as the chapter file. Use the second form if the root if in a folder which contains a subfolder for each chapter file. Use the third form if the root is further up the folder chain. Use the fourth form if you want to give a full path to the root file. Note that the fourth form is not portable, so moving the source folder of a project will require rewriting the magic lines of such chapters.

The Macros menu has an item labeled "Root". To add the magic line to a chapter source file, activate that source file and then select the macro. A Finder window will appear which you can use to navigate to the root file. The macro will then offer to create either a relative or an absolute magic line, replacing any earlier root magic line.

If a chapter file with a root magic line is opened in TeXShop, TeXShop will also open the root file, and thus also the associated Preview Window for the entire book. It will not open remaining chapter files. Thus an author can revise the chapter text and typeset as usual; the Preview Window will show changes. During this operation, it is usually not necessary to edit the root file, so some authors prefer to immediately minimize the root source. In TeXShop Preferences under the Misc2 tab there is an item to immediately minimize the root source after opening a chapter file and its root. This action is a little surprising, so I recommend not selecting it at first. Later on, you can decide if you want to activate it. If multiple files open in the same window using tabs (as explained in a later chapter), then minimizing the root is completely unnecessary.

The root magic line is also important for the "Goto Error", "Kill Aux Files", and "SyncTeX" features to be discussed next.

Chapter 10

Goto Error, Trash Aux Files, SyncTeX

10.1 Goto Error

When TeX typesets and finds a syntax error, it stops and the console window comes to the front. TeX prints an error message at the end of the console output. Pushing RETURN causes TeX to continue typesetting until it comes to another error. Pushing RETURN many times will often lead to finishing the typesetting job and thus outputting new preview contents.

The Console window has a button labeled Goto Error. There is also a menu in the Edit menu and a keyboard shortcut for this command. The command will activate the source file, scroll to the line containing the first error, and highlight that line. Pushing the button again will go to the second error line, and so on. It is often useful to fix four or five errors and then typeset again, because one error can cause another so fixing one error may fix many.

If the project has a root file and various chapter files, GotoError will open the source file containing the error if it is not currently open, and then go to the appropriate line in that source file.

The Goto Error command parses the console output to find error messages, and requires that TeX report these errors in suitable format. So calls to LaTeX, pdfTeX, XeTeX, and other engines should contain the flag

    --file-line-error

to activate this formatting.

10.2 Trash Aux Files

As explained earlier, when LaTeX first typesets, it creates an "aux file" containing information needed for tables of contents, equation numbers, and the like. The information in this file is then used in a second pass of LaTeX. Several passes may be needed before the document is completely up to date. If the source is named Galois.tex, then the aux file will be named Galois.aux.

If the source has an error and the user aborted typesetting when the error was reported, the aux file may be corrupt. In that case, LaTeX will open the aux file when it typesets a second time and become confused. This often happens shortly before the "begindocument" line. A source file which claims to have an error there may instead have a corrupt aux file. The solution for this problem is to throw the aux file in the trash and typeset again from scratch.

This sort of problem tends to be rare for small documents and common for large documents or documents with many illustrations. TeXShop has a button in the console labeled "Trash Aux Files" which trashes the aux files in the document's folder. There is also a menu item in the Source menu and a keyboard shortcut to trash these files. When you run into the problem, push the button or menu item and then typeset again.

Because the problem is common for large documents, there is an even better shortcut. Recall that command-T typesets. A similar shortcut, option-command-T, trashes the aux files and then typesets. If the Typeset menu is open, notice that pushing the Option key changes the item titled "Typeset" to read "Trash Aux & Typeset".

If a project has a root file and then chapter files, the aux files are associated with the root file. So if a chapter file is typeset and the chapter file has the correct magic line pointing to the root file, then the aux files of the root file will be trashed.

The aux file is not the only auxiliary file which LaTeX writes during typesetting. It also can write a bbl file, a toc file, and many others. Each saves pieces of information that are used in later passes of LaTeX. Therefore the command which trashes the aux file also trashes many other files. Here's a list:

```
aux, blg, brf, glo, idx, ilg, ind, loa, lof, log, lot, mtc, mlf,
out, ttt, fff, ent, css, idv, wrm, 4ct, 4tc, lg, xref, bcf, gtex,
gaux, glog, pdfsync, synctex, synctex(busy),latexmk,
fls, toc
```

If typesetting goes seriously wrong, even the pdf file can become corrupt. Generally users do not want to trash the old pdf file since it is the preview image they will use when a new image is not created. But if you hold down the Shift key while activating Trash AUX Files, the pdf file will also be removed.

There is a complication for projects with a root file and included chapter files. In that case, typesetting the root file can leave root.aux and root.pdfsync in the folder with the root file, and chapter.aux in the folders with the various chapter files. So more extensive cleanup is needed. To make that happen, hold down the Option key while activating Trash AUX Files. Then all files in the root folder or any of its subfolders, regardless of name, will be trashed if they have an appropriate extension.

The remaining paragraphs in this section get into the weeds for experts. Some users need to add extra elements to the list of trashed files. That can be done using a hidden preference setting. A hidden preference setting is a setting that is not shown in TeXShop Preferences, and thus must be manipulated using Terminal and the command line.

For example, to add "dvi" to the list, activate the Terminal and type

```
defaults write TeXShop OtherTrashExtensions -array-add "dvi"
```

To remove all additions and return to the original default list, type

```
defaults write TeXShop OtherTrashExtensions -array
```

To make the aggressive behavior created by holding down the Option key the standard behavior of Trash AUX Files,

```
defaults write TeXShop AggressiveTrashAUX YES
```

10.3   SyncTeX

SyncTeX is a feature which allows users to rapidly move from a point in the source file to the corresponding point in the output file, and vice-versa. This feature was first provided by *Textures*, a front end for the original Macintosh system written by Barry Smith in Portland, Oregon. My understanding is that Smith had rewritten the Pascal compiler and then large portions of the TeX code and used this knowledge to provide very accurate sync information. His company, Blue Sky Research, was in the process of porting *Textures* to macOS when he tragically died in 2012.

SyncTeX was first provided for regular systems on macOS by Jerome Laurens. His original method was a package which users activated in their source; this package output a synctex file containing information which front ends could use to sync between source and preview windows. Unfortunately, this package sometimes modified typesetting, so output when using it differed from output when not using it.

Later TeXShop provided a method directly in the program, bypassing Lauren's package; this didn't work very well and the less said about it, the better.

Eventually Laurens provided a version of SyncTeX which outputs sync information during typesetting directly from the typesetting engines. This no longer modifies typesetting output. His code is built into most of the typesetting engines supplied by TeX Live. When a source file, say Galois.tex, is typeset by one of these engines, and the engine is called with the flag "–synctex=1", it outputs a file named Galois.synctex containing the required sync information. Laurens provides a library written in C to interpret this information; TeXShop contains that library and uses it to provide syncing.

SyncTeX is turned on in TeXShop Preferences by an item under the Typesetting tab. Be sure to select the last item under Sync Method, called "SyncTeX (TeX > 2010)". Other items are for previous sync methods now obsolete.

To sync from the source to the preview window, hold down the Command key and click at a point. The Preview window will scroll to the appropriate spot and highlight the corresponding text in the output. To Sync the other way, hold down the Command key and click at a point in the Preview window.

SyncTeX is an essential feature of front ends, but it is not completely accurate. A short search afterward may be needed to find the precise point where changes should be made.

If a project has a root file and included chapter files, clicking on a point in the source for a chapter takes you to the corresponding point in the preview window. Clicking on a point in the preview window activates the source window for the appropriate chapter. If this window is not open, it will be opened automatically. Then you are taken to the appropriate spot in that chapter's source.

This sync feature is a useful way to open appropriate chapter files. If a project with a root named Galois and various chapter files exists, open Galois.tex to see the root source and the full Preview. Scroll to an interesting point in the Preview and sync from one of its points. The appropriate source file will open automatically.

There is one tricky feature of SyncTeX. The information in the synctex file created by the typesetting engines contains full paths to files. So if a folder containing a LaTeX project is sent to someone else or moved to another location, synctex will no longer work and Command-click will do nothing. To fix this, just typeset again. That creates a new synctex file.

Chapter 11

Organizing Windows in TeXShop

11.1 Introduction

A user working on a typical TeX project will have several windows open at once: the source window, the preview window, and the console. The number of open windows will increase if a project has a root file and individual chapter files. If several projects are open at once, many windows will be open.

Some TeXShop users cope with these multiple window by working with a large monitor, and I'll confess that I use that strategy. Other users prefer a different solution. There are essentially four ways to configure TeXShop for multiple window use:

- Use multiple windows in the normal way
- Switch to single window mode, with the source and preview in a single window
- Use multiple windows with tabs
- Use single windows with tabs

Users should select one of these strategies and stick with it. We have only discussed the first option so far. It is time to introduce the other methods.

11.2 Single Window Mode

Open a project which has one source window and one preview window. With one of these windows active, go to the TeXShop Window menu and select the item "Use One Window".

The two windows will be merged into a single window with the source on the left and the preview on the right. The first time this item is used, this window will be small and awkwardly placed on the screen. Resize it and place it in a convenient place. From then on, all single mode windows will appear in this location with this size.

If you prefer the preview on the left and the source on the right, go to the Preview tab in TeXShop Preferences and reset the appropriate item.

To return to ordinary mode, activate the single window and select "Use Separate Windows" in the Window menu.

The commands listed so far work best for isolated moments when it is convenient to switch one or more projects to single window mode temporarily. But if you close a project in single window mode, and later reopen it, it will appear in standard mode. If you configure TeXShop so it remembers windows that are open when it quits and reopens them when it restarts, then a source and preview in the same window when TeXShop quits will reopen in two separate windows with unusual sizes. They must be resized before being used.

In short, if you use a mixture of standard mode windows and single window mode windows, you may run into unusual behavior when quitting and restarting.

These problems do not occur if you always work in single window mode. To configure TeXShop for that behavior, select the option to "Use One Window" when "Opening Source and Preview" in TeXShop Preferences under the Preview tab. Then all documents will open in single window mode and restoring windows after quitting and restarting will work as expected. In short, you will have completely converted to One Window Mode for everything.

There are just two cautionary notes. First, projects with a root file and separate chapter files do not work in single window mode, so do not switch to single window mode if you use that facility. And second, the File menu's "New" command always produces a single source window (because there is no file to preview at first). So start new documents by writing a skeleton source file, typeset that file, and then switch that project to single window mode with the Window menu's "Use Single Window" command. Do the same thing if a colleague sends you the source for a document, but no pdf.

11.3   Multiple Windows with Tabs

Several years ago Apple introduced support for tabs in windows. Recently a user, Szilágyi Zsolt, wrote that TeXShop has a problem with tabs, and he sent a movie showing the problem in action. A few days later he solved the problem himself. I looked at the movie anyway, and Zsolt was doing things with tabs in TeXShop that I never dreamed possible. So I asked how he did them and was told that in Apple's System Preferences under the

General tab, there is an item labeled "Prefer tabs in full screen when opening documents." Here the words "full screen" appear in a pull-down menu whose other choices are "never" and "always". Selecting "always" greatly improves the way tabs work in TeXShop.

Next I discovered that some applications have their own similar preference. This makes it possible to improve tab support in these applications without forcing it upon the full system. For example, Safari has an item labeled "Open pages in tabs instead of windows" with possible values "Never", "Automatically", and "Always". The choice "Automatically" causes Safari to follow the global preference item in System Preferences.

TeXShop now has a similar preference setting. The item is at the bottom of the Source tab in TeXShop Preferences, with the same choices as Safari.

When separate source and preview windows are used:

- Set the new preference to "Always" and close all windows. Open a document from disk. A source window and a preview window appear. Open another document. The source and preview for the new document are added as tabs to the existing source and preview windows. We still have only two windows. Open a third document, perhaps this time one based on a root file with various chapter files. A tab for the root file appears in the source window, and a tab for the preview appears in the preview window. We still have only two windows.

- Select one of the source tabs and typeset. A console window appears; when typesetting ends, the appropriate preview tab is activated. Select a different source tab and typeset again. Rather than creating a new console window, a tab is created in the existing console; after typesetting ends, the appropriate preview tab is activated. We still have only three windows: source, preview, console.

- Try sync from source to preview or preview to source. These work and activate the appropriate tab in the destination of the sync.

- Select the log file for a document. A new window appears containing the log. Change to a different document and show its log file. A tab appears in the log window for the new log. We only have four windows: source, preview, console, log.

- Open a document in the Help menu. A new window appears containing the document. Note that the document does not appear as a tab in the source or preview windows. This is deliberate since help pages are often used in conjunction with TeX tasks and both need to be visible. We don't want to clutter the source and preview windows with documents which aren't source or preview items. Each Help document gets a separate window because multiple items aren't often used simultaneously and some of the items are displayed by other programs like Apple's Preview and not even governed by the new TeXShop preference.

- Select the menu item "About TeXShop." It is displayed in a separate dialog, as we would hope.

- In the preview window, open the Drawer. Then switch to another tab and notice that the Drawer disappears. That is because each tab controls its own drawer. Switch back to the original tab and notice that the drawer opens again. Activate drawers for two different preview tabs. Notice that when switching from one tab to another, the contents of the drawer change to match the selected tab.

- Select a tab for a document containing a root file and various chapter files. Sync from the preview to the source. Notice that the appropriate chapter file opens and a new tab for it is created. Using sync in this way, you can obtain tabs for all chapters currently being edited.

So far, things work in the expected manner. We end with a couple of cases where surprising things happen.

- Suppose we have source and preview windows with three tabs for three projects. Activate one of these tabs and select "Use One Window." A new window is created containing both source and preview panes, but only for the active tab, not for all three projects. If we select "Use One Window" again, another of the documents is converted to single window mode, but this time that single window becomes a tab in the single window for the first document. Selecting "Use One Window" a third time converts the remaining document, and now at last we have what we expected to get immediately, namely a one-window object with three tabs for the three documents. Unwrapping these windows using "Use Separate Windows" works the same way, unwrapping one document at a time.

- The surprising behavior of tabs and "Use One Window" is actually useful because it solves a problem with full screen mode. This mode is essentially useless if you are a "separate window mode" person, because it is impossible to write TeX if you only have access to the source, or only have access to the output. But suppose you are writing a very tricky portion of a document and would like to concentrate only on it without distractions. Convert the document to a single window with "Use One Window." Then click the green button to make this window full screen. You now have both source and text on the screen, which is exactly what you need. Moreover, you can typeset because when you do that, the console becomes a floating window on top of the screen. Use it to fix errors and then close it. Continue editing. When you are done, click the green button to return to a single window, and then select "Use Separate Windows" to return your document to tabs where it started.

- You can also move two different source tabs to tabs in a single window and then convert this window to full screen. This makes it possible to work on two separate documents at the same time in full screen mode.

- There is only one really bad piece of news for people who use separate source and preview windows. If you have a document with a root file and various chapter files, tabs work fine as long as you stay in this mode. But converting such a document to single-window mode and then taking it full screen will completely fail. Single windows contain ONE source file and ONE preview file. If your project also has chapter files and you try to convert it to single window mode, you end up with a single window for the root source and the preview output, and then lots of source windows for the individual chapters. This isn't something I can fix.

So you have to make a choice. If you like to split your projects into a root and chapters, fine. This decision is completely reasonable. Tabs may make your screen less cluttered; give them a try. Never convert projects with a root and various chapter files to one window mode, and never use full screen mode for this kind of project. This is actually how I often work.

The other choice is to write books with just a single source file. Many famous authors work this way. I used to use ROOT-CHAPTER, but recently I've written even longer documents with just a single source. In that case, the full TeXShop interface is available to you.

11.4   Single Window Mode and Tabs

If you work exclusively in single-window mode and want to use tabs:

- I recommend activating the preference item "Opening Source and Preview: Use One Window" at the bottom of the Preview Tab in TeXShop Preferences. Then documents immediately open in one window mode.

- Although the Window menu has items "Use One Window" and "Use Separate Windows", there are only two circumstances when these are helpful. The first occurs when you start a new document. The second occurs when someone sends you a TeX source document without an associated pdf. In both cases, create or open the source file. Typeset. Now you have both windows. Activate one of the two and select "Use One Window".

- Set the new tab preference to "Always" and close all windows. Open a document from disk. A window opens containing the source and the output of the document. Open a second document. A new tab appears in the window, showing the source and output of the second document. We still have one window. Open a third document. Another tab appears and our window contains all three documents.

- We do not get one set of tabs for the preview portion and a separate set for the output portion. Instead we only get one set of tabs because tabs select entire documents.

- Select a tab and typeset. A console window appears and the new output appears in

the window. Select a different source tab and typeset again. Rather than creating a new console window, a tab is created in the existing console. We still have only two windows: project and console.

- Sync works fine.

- Select the log file for a document. A new window appears containing the log. Change to a different document and show its log file. A tab appears in the log window for the new log. We only have three windows: project, console, log.

- Open a document in the Help menu. A new window appears containing the document, so the help document does not appear as a tab in the document window. This is deliberate since help pages are often used in conjunction with TeX tasks and both need to be visible. Note that each Help document gets a separate window.

- Select the menu item "About TeXShop." It is displayed in a separate dialog, as usual.

- Open the drawer. Then switch to another tab and notice that the Drawer disappears. That is because each tab controls its own drawer. Switch back to the original tab and notice that the drawer opens again. Activate drawers for two different tabs. Notice that when switching from one tab to another, the contents of the drawer change to match the selected tab.

- So far, things work in the expected manner. There is one surprise. Suppose we still have three tabs. Activate one of these tabs and select "Use Separate Windows" Two new windows are created containing the source and preview for the active tab, not for all three tabs. If we select "Use Separate Windows" again, another of the tabs is converted source and preview files, this time as tabs in the source and preview windows. So we work one project at a time, rather than converting all tabs at once. However, this is a task that is rarely done if you use one-window mode.

- Select a window with tabs. Click the green button to convert the window to full screen mode. You can still typeset because the console will appear as a floating window over the screen, and can be closed to give a full screen view. Now tabs allow you to work on several projects at once in full screen mode.

- There is only one bad piece of news. If you work in one-window mode, you cannot work with projects that have a ROOT file and then various CHAPTER files. But you couldn't do that before tabs either. Many authors of books keep the full source in a single file.

11.5   Obsolete Tab Commands

Before the changes described above were present, TeXShop had a few other features dealing with tabs. These features are now obsolete and should not be used.

In TeXShop Preferences under the Typesetting tab, there is an item labeled "If Sync Opens a New Window" and then a check mark "Open as Tab in Root Window." Do not check this.

Version 3.82 and 3.84 introduced magic comment lines dealing with tabs. Read the Changes document for details. These methods are untested with the modern tab support and should not be used.

```
% !TEX useTabs
% !TEX useTabsWithFiles
% !TEX tabbedFile{chapter1/Introduction.tex} (One)
```

Chapter 12

Macros

TeXShop users can write and execute AppleScript commands using the Macros feature of TeXShop. But what is Applescript? Let me quote Apple's *Introduction to AppleScript Language Guide*:

> AppleScript is a scripting language created by Apple. It allows users to directly control scriptable Macintosh applications, as well as parts of macOS itself. You can create scripts—sets of written instructions—to automate repetitive tasks, combine features from multiple scriptable applications, and create complex workflows.
>
> A scriptable application is one that can be controlled by a script. For AppleScript, that means being responsive to interapplication messages, called Apple events, sent when a script command targets the application. (Apple events can also be sent directly from other applications and macOS.)

Macros and the Macro Editor are by Mitsuhiro Shishikura. Default Latex macros were created by Mitsuhiro Shishikura and Hirokazu Ogawa. Default Context macros were created by Hans Hagen.

I should confess that I find AppleScript difficult to use. The language was designed to be "conversational", but this creates ambiguities, at least for me. The current AppleScript expert for TeXShop is Michael Sharpe, who wrote a great document about it named *Notes on Applescript in TeXShop*. His document is provided as an item in the TeXShop Help Menu.

An example will explain why AppleScript is useful. Around 2009, Ramon Figueroa-Centeno wrote three crucial applescript commands named "Program", "Encoding", and "Root". These are available near the top of the Macros menu, and I use them every day and recommend them to all TeXShop users. Each creates a magic comment line. For instance,

the "Program" command displays a list of all active engines. Select one of these, and the corresponding magic line similar to

```
% !TEX program = xelatex
```

is written at the top of the source, replacing any similar existing line if necessary. Magic comment lines are picky about spaces and capitalization of engine names, but the Macro handles all those details automatically.

The Macros feature of TeXShop has been stable for many years, so the material below is taken directly from the TeXShop Help Panel.

12.1 Introduction

TeXShop Macros and the Macro Editor were written by Mitsuhiro Shishikura. The available Macros depend on the typesetting engine chosen. If this engine is LaTeX, then a series of macros suitable for LaTeX will be displayed. If the engine is ConTeXT, then macros for ConTeXt will be displayed. When TeXShop is first installed, it has macros for only these two engines; the LaTeX macros will be displayed for all engines except ConTeXt.

Macros come in two kinds. Some insert strings into the TeX source file; these macros are similar to buttons in the Latex panel. Other macros run Applescript scripts.

Macros are stored in

```
~/Library/TeXShop/Macros/Macros_Latex.plist
```

and

```
~/Library/TeXShop/Macros/Macros_Context.plist.
```

If these files are missing when TeXShop runs, default macro files are created there. These files are ordinary text files, so it can be opened and inspected with TeXShop. This is usually not necessary because the files are best manipulated with the Macro editor.

The Macro editor modifies macros stored in Macros_Latex.plist and Macros_Context.plist when one of these typesetting engines is chosen. But if another typesetting engine like TeX is chosen, then the Macro editor will replace the default Latex macro set with a set defined by the user for TeX, storing the results in Macros_Tex.plist and leaving Macros_Latex.plist unchanged.

When you define useful Macros and wish to give them to others, simply open the Macro Editor and choose "Save selection to file..." in the Macro menu. This will create a file in any location you wish. To distribute your macros, send this file.

To add macros created by others to your list of macros, open the Macro Editor and choose "Add macros from file..." Then use the Macro Editor to arrange the macros as you desire.

New items, new submenus, and new separators can be created by the buttons at the bottom left. A little practice illustrates the basic behavior of the editor. Submenus can be created to any level.

Any particular item can be assigned a keyboard equivalent using the buttons on the right side of the menu. The result will immediately appear in the outline view. However, these assignments cannot duplicate key combinations already used by TeXShop menus. If they do, the new keyboard shortcut will be ignored.

12.2   Default Macros

The best way to understand default macros is to examine their definition with the Macro Editor. For example, consider the macro titled "Begin/End". Suppose you wish to use an environment like the theorem environment. Type the word "theorem" and select it. Then choose the "Begin/End macro. The word "theorem" will be replaced with the text

```
\begin{theorem}
\end{theorem}
```

with the cursor placed on the line between this pair.

Now examine the begin/end macro code:

```
\begin{#SEL#}
#INS#
\end{#SEL#}
```

Text in the macro will be inserted into the source file. Each occurrence of the string #SEL# will be replaced by the text selected when the macro was invoked. If no text was selected, #SEL# will be replaced with an empty string. The cursor will be placed at the end of the inserted text unless the text contains the string #INS#, in which case the cursor will be placed at that location.

Using this knowledge, it is easy to understand and modify the default macros.

12.3   Rearranging Macros

Open the Macro Editor and examine the macro menu on the left. It is shown in an outline view similar to the view of the file system obtained by choosing the middle button of the View tab in the Finder. To rearrange items, drag them from one spot to another. Notice

that items can be placed at different levels by sliding left and right. To rename an item, select it in the outline view and type a new name in the field at the top right.

12.4   Applescript Macros

If a macro begins with the string

```
--AppleScript
```

or

```
--AppleScript direct
```

then the resulting applescript code will run when the Macro is chosen. For example, consider the macro titled "View pdf with Acrobat". Choosing this macro when the source file is active (and the pdf file has been created) will start Adobe Acrobat Reader and open the output pdf file in that program. The corresponding applescript code reads

```
--AppleScript direct

  tell application "Acrobat Reader 5.0"
     activate
     open POSIX file #PDFPATH#
  end tell
```

Commands beginning with "–AppleScript direct" are run directly by TeXShop. During the time this applescript is running, TeXShop's event loop is not active. Consequently, the script cannot call TeXShop to perform an action which might require user input. For example, it cannot ask TeXShop to run LaTeX, because if the LaTeX file has an error, the console will appear and wait for user input, but such input would not be recognized.

Commands beginning with "–AppleScript" are run by a small auxiliary program in the TeXShop application bundle. Such commands can ask TeXShop to perform actions which might require user input, because in that case the auxiliary program will temporarily halt but TeXShop will remain active.

Thus the syntax "–AppleScript direct" is appropriate for routine macros, but "–AppleScript" may be needed for fancy ones.

12.5   Included Applescript Commands

TeXShop comes with an extended collection of Applescripts by Will Robertson, Claus Gerhardt and others. Some of these scripts automate workflow when a series of typesetting

commands must be issued in sequence. By copying and modifying the scripts, users can construct scripts appropriate for their own workflow.

In this section, we will describe some of these scripts.

Column Macros, Insert Reference, Open Quickly

These macros are by Will Robertson. The first provides a very convenient way to construct an arbitrary matrix or table. The second searches through the current file for

```
\label{...}
```

commands and then pops up a list from which you may insert a reference label, wrapped in an (optional) customizable LaTeX command. The final macro allows you to rapidly open any file in the directory of the current source file.

Paste Spreadsheet Cells

This macro (by Alan Munn) provides cut-and-paste functionality from spreadsheet applications into your LaTeX source, using various popular table styles. Select a range of cells in Excel or other spreadsheet application, and copy them to the clipboard. Within TeXShop, choose Paste Spreadsheet Cells in the Macros menu. You will be asked to choose a table style. The styles available are the following:

- **cells**
  converts the clipboard text simply to latex table cells

- **simple**
  creates a simple tabular environment around the cells

- **booktabs**
  assumes the first line is a header, and places "toprule", "midrule" and "bottomrules" in the appropriate places within a tabular environment

- **longtable**
  assumes the first line is a header, creates the appropriate longtable header information and places the cells in a longtable environment

Note that the script modifies the contents of the clipboard to contain the LaTeX formatted text, and not your originally copied cells. This means that if you need to change cell formats, you must re-copy the cells from your spreadsheet before choosing a different cell format to paste.

Convert to Mac, Convert to Unix, Convert to Windows

In the early days, teletype terminals were used for communication. The ascii character set still has remnants from those days; for instance 0x07 rings the teletype bell. The character 0x0a was a line feed which turned the carriage to the next line and the character 0x0d was a carriage return which pulled the carriage back to the start of the line. To get a line feed, one used the sequence 0x0d 0x0a.

After the age of teletypes ended, computer manufacturers selected different subsets of these characters to indicate a line feed. In the Unix world, 0x0a was used, in the Windows world 0x0d 0x0a was used, and in the old Macintosh Classic world 0x0d was used. Apple's guidelines for Mac OS X state that programs should be able to automatically open files using any of these conventions. Most programs including TextEdit, TeXShop, etc., follow these guidelines. In TeXShop, new files are created using Unix conventions, but if you load a file created with Mac OS Classic and then add extra lines, old portions of the document will be saved with the Classic convention and new sections will have the Unix convention.

These line feeds cause no trouble until you send a file to a friend using a different operating system. Many editors now understand multiple line feed conventions, so often you'll have no problems. But if you do, use the scripts above. Suppose the source file for a document is named MyFile.tex. Then "Convert to Mac" will create a new file named MyFile_Mac.tex with the same source code and Macintosh line feed conventions. "Convert to Unix" and "Convert to Windows" work the same way. All of these scripts call a binary program by Craig Stuart Sapp named "flip" in /Library/TeXShop/bin.

Other Scripts: Bibliography

Processing a file with a bibliography requires multiple typesetting operations. First Latex is run to create an .aux file. Then Bibtex is run and uses this file to create .bbl and .blg files. Latex is run again to add the bibliography to the document. Latex is run a final time to update the references to the bibliography in the text.

The "Bibliography" command does all of these things one by one. First it saves the file. Then it runs latex->bibtex->latex->latex. Finally it updates the preview display.

htlatexc, htlatexr

Tex4ht is a TeX program which converts a latex document into a web page. The source can be a standard Latex file and can include eps illustrations. The end result is an html page and a large number of gif files. The script command "htlatexc" saves the source documents, runs htlatex, and opens the resulting html file in Safari. Thus it behaves like a new TeXShop typesetting command, except that it was constructed by a user without

waiting for new TeXShop code! (Note: In 2023, TeXShop was modified to directly typeset with TeX4ht and display the result in an internal HTML Preview Window rather than Safari. See a later chapter for details.)

When using these scripts, it is not necessary to use the package tex4ht since this package will automatically be loaded. The script htlatexc calls htlatex without additional options. The script htlatexr calls it with the option "-r", which tells the program to recreate any .gifs which already exist.

pdfselectc

This script runs the shell script pdfselect to create pdf files containing only certain selected pages of a document. When it is run, a dialog appears asking for the number of pdf files to be created. Suppose we answer 3. Next a dialog appears asking for the range for the first document. Suppose we answer 5:8. A dialog appears asking for the range of the second document. Suppose we answer 10. A dialog asks for the range of the third document. Suppose we answer 20:30. We will then obtain three documents, one containing pages 5 - 8 of the original, one containing page 10, and one containing pages 20 - 30.

mpostc, mpostcpl, latex-makeindex-mpost

The script mpostc runs mpost and then pdflatex; the script mpostcpl runs pdflatex, and then mpost, and then pdflatex again.

The script latex-makeindex-mpost saves the source and runs pdflatex, makeindex, mpost, and pdflatex again, opening relevant log files in the process.

12.6   Defining AppleScripts

AppleScript syntax is a subject all its own. To learn more about it, read one of several books on the subject. If you installed the developer tools which come with Mac OS X, there is an online book about AppleScript in the Developer folder.

When TeXShop interpretes an applescript macro, it first replaces any string #FILEPATH#, #PDFPATH#, #DVIPATH#, #PSPATH#, #LOGPATH#, #AUXPATH# with the complete path name of the source tex file, pdf file, dvi file, ps file, log file, or aux file respectively. Similarly, the strings #INDPATH#, #BBLPATH#, and #HTMLPATH# are replaced with the complete path name of the ind file, bbl file, or html file.

In addition, any string #NAMEPATH# is replaced with the complete path name of the source tex file minus its extension, and any string #DOCUMENTNAME# is replaced with the display name of the current document. This last replacement is somewhat subtle; it gives the title of the document as shown at the top of the source window. If a document was

saved with the "hide extension" box checked, #DOCUMENTNAME# will contain only the document name. But if the document was saved without checking "hide extension", #DOCUMENTNAME# will contain the document name and extension. This information can be used to locate the calling document for Applescript code as follows:

```
tell document #DOCUMENTNAME# of application "TeXShop"
  latex
end tell
```

There are at least two ways to write Applescript commands. Applescript can run shell commands, so after preliminary processing with an applescript, a shell command can be called to do the actual work. TeXShop comes with several examples of this technique; some of these examples will be explained in a later help section. Applescript commands can also call built-in TeXShop commands and thus work directly. A later section will give examples of this technique.

TeXShop understands the following commands:

```
typeset
latex
tex
context
bibtex
makeindex
metapost
typesetinteractive
latexinteractive
texinteractive
contextinteractive
bibtexinteractive
makeindexinteractive
metapostinteractive
taskdone
refreshpdf
refreshtext
goto line
```

The first seven commands call TeXShop typesetting routines. These commands typeset continuously without stopping at errors. The next seven commands also call TeXShop typesetting commands, but this time if there is an error, the user is allowed to interact with the console. When a typesetting command is called, control returns to the applescript immediately without waiting until the operation is complete. The "taskdone" call returns NO if typesetting is still running, and YES when it is done. The calls "refreshpdf" and

"refreshtext" cause pdf and text documents to display the latest version of their files on the screen. The "goto line" command tells the editor to select a given line; for example:

```
tell document #DOCUMENTNAME# of application "TeXShop"
  goto line 37
end tell
```

12.7 Dialogs

AppleScript can create dialogs and act on user input. For example, insert the following script and test its behavior.

```
-- AppleScript <br>
-- Use a dialog to enter user defined information into an applescript.
-- There are two feedbacks possible 'text' and "choice of buttons".
-- The returned information can be used to define corresponding variables.
-- The number of buttons may not exceed three.
-- In the three examples below it is shown how to use text return,
-- button return, and both.

activate
display dialog "Test dialog: type something below" default answer "Hello World!"
   buttons {"A", "B", "C"} default button "B"
set theText to (the text returned of the result)
display dialog "Test dialog: type something below" default answer "Hello World!"
   buttons {"A", "B", "C"} default button "B"
set theButton to (the button returned of the result)
display dialog "Test dialog: type something below" default answer "Hello World!"
   buttons {"A", "B", "C"} default button "B"
set {theText, theButton} to
  {the text returned of the result, the button returned of the result}
```

One command in this example, "activate", requires comment. When TeXShop is asked to run an AppleScript, it calls a separate program to run the script. That program, Scriptrunner, is in the TeXShop bundle. Any dialogs created by the script will be displayed by Scriptrunner rather than by TeXShop. The "activate" command brings Scriptrunner to the front so dialogs will appear on top of other windows.

12.8 Writing Complete Applescripts

If a TeX project contains a bibliography, a sequence of typesetting commands must be run to update the bibliography. Latex is run first to create an .aux file. Bibtex is run to create

.bbl and .blg files from this .aux file. Latex is run again to add the bibliography to the document. Latex is run a final time to update the references to the bibliography in the document.

Your projects may contain similar sequences of typesetting commands. It is possible to use applescript to automate these sequences.

To see how to do that we'll examine the OtherScripts: Bibliography command which comes with TeXShop. Here is the body of that command:

```
--Applescript

set fileName to #FILEPATH#
if fileName is equal to  ""
  activate
  display dialog "Please save the file first"
       buttons {"OK"} default button "OK"
  return
end if

set frontName to #DOCUMENTNAME#
tell application "TeXShop"
  save document frontName
end tell

tell document frontName of application "TeXShop"
  latexinteractive
  repeat
    delay 2
    if taskdone
      exit repeat
    end if
  end repeat

  bibtex
  repeat
    delay 2
    if taskdone
      exit repeat
    end if
  end repeat
```

```
    latex
    repeat
      delay 2
      if taskdone
        exit repeat
      end if
    end repeat

    latex
  end tell
```

The first line of this command indicates that this is an AppleScript macro. The next lines check #FILEPATH#, a parameter which gives the full path to the tex source. This parameter is an empty string when a new document has been created and not yet saved. In that case the user is asked to save the document and the script ends.

The next line tells TeXShop to save the document. Notice that we use #DOCUMENT-NAME# to refer to the document in question.

The remaining commands run latexinteractive, bibtex, latex, and latex. Recall that control returns to applescript immediately after calling a typesetting command before the typesetting job is over. The repeat loop tells the script to check whether the job is complete before running another typesetting task. The line "delay 2" causes applescript to pause rather than continually asking if the task is done and thereby slowing the entire computer down.

12.9   Writing Command Scripts

We will use the AppleScript "pdflatexc" by Claus Gerhardt to illustrate the principles involved in calling a Unix shell script from an Applescript. This particular shell script isn't very interesting; it adds the location of the TeX binary files to the $PATH variable and calls pdflatex to typeset.

The shell script itself is independent of TeXShop and can be run from the Terminal by typing "pdflatexc myfile.tex" provided the directory holding pdflatexc is in the search path for binaries. Here is that shell script:

```
#!/bin/tcsh
# pdflatexc
# Claus Gerhardt
#
# Usage
# pdflatexc filename.tex
```

```
    set path= ($path /Library/TeX/texbin /usr/texbin /usr/local/bin)]
    pdflatex --shell-escape "$1"
```

Of course you are likely to write a more complicated script which performs several operations in sequence. That is when these techniques become useful.

The AppleScript used to call this shell script is more interesting. Here it is:

```
--Applescript
-- Apply only to an already saved file.
-- Claus Gerhardt, Nov. 2003

set scriptPath to (do shell script "dirname " & "~/Library/TeXShop/Scripts/ex")
set scriptPath to scriptPath & "/setname.scpt"
set scriptName to POSIX file scriptPath as alias
set scriptLiB to (load script scriptName)
tell scriptLib
    set frontName to setname(#NAMEPATH#,#TEXPATH#)
end tell

set fileName to  #TEXPATH#
set n to (number of characters of contents of fileName)
set fileNamequoted to quoted form of fileName
set baseName to do shell script "basename " & fileNamequoted
set m to (number of characters of contents of baseName)
set dirName to quoted form of (characters 1 thru (n - m - 1) of fileName as string)

set shellScript to "cd " & dirName & ";"
set shellScript to shellScript & "~/Library/TeXShop/bin/pdflatexc  " & baseName
do shell script shellScript

tell document frontName
    refreshpdf
end tell
```

Ignoring the introductory comments, the first seven lines of this script are a magic recipe by Claus Gerhardt to save the source file and find the name of the document in question, setting "frontName" to this name. This recipe uses a compiled script named "setpath.scpt" in `~/Library/TeXShop/Scripts` to do all the hard work. Careful reading shows that these lines find the path `~/Library/TeXShop/Scripts/setname.scpt` and call this script with parameters #NAMEPATH# and #TEXPATH#.

In many AppleScripts all of this could be accomplished in an easier way using the commands

```
set frontName to #DOCUMENTNAME#
tell document frontName of application "TeXShop"
     save
end tell
```

However, Gerhardt's script has two advantages. First, his script can be called when the front document is a log file, say /Users/koch/Examples/myfile.log, that is, in a case when the front document is not the document which should receive later commands. Second, if the file has never been saved, Gerhardt's script returns an error when trying to save, while the "save" command would cause TeXShop to hang after it put up a save dialog (see the help document Writing Scripts with TeXShop Typesetting Commands for details.)

The next six lines of the script define the variables dirName and baseName. If the source file is "/Users/koch/This directory/Stuff/myfile.tex", dirName is " '/Users/koch/This directory/Stuff' ", including the single and double quotation marks, and baseName is "myfile.tex." A lot of this work is required so spaces can occur in the names of folders. As always, tex does not allow spaces in the final file name.

The next three line call the shell script. The result is the same as typing "cd dirName; /Library/TeXShop/bin/pdflatexc baseName" in a Terminal, although, to be absolutely precise, dirName would have to be replaced by its unquoted form, i.e., if we use the example above, by '/Users/koch/This directory/Stuff' including the single quotes because of the space in the name of one directory.

Shell commands in AppleScript are issued in the form

```
do shell script "cmd input"
```

If one wants to combine several shell scripts, it is better to write the command in an equivalent form

```
do shell script "cmd " & "input"
```

Notice the space behind cmd and the quotes. The ampersand is a binary concatenation operator, i.e.,

```
"cmd " & "input" = "cmd input"
```

When the shell is called via an applescript, the default working directory is the root directory. The command

```
do shell script "cd " dirName
```

changes the directory to the directory specified in dirName; dirName is already quoted so that additional quotes are not needed.

If one would like to keep the working directory and issue further commands, then these commands may not be stated in the form "do shell script", since then a new shell would be invoked. In the terminal one would separate consecutive commands by semicolons, in AppleScript one does it by concatenating ";", i.e.,

```
do shell script "cmd(1) " & "input(1)" & ";" & "cmd(2) " & "input(2)"
```

This is done in the above example: cmd(1) = cd, "input(1)" = dirName, cmd(2) = ~/Library/TeX/bin/pdflatexc - calling the shell script -, and "input(2)" = baseName.

The lines

```
<div id="source">
set shellScript to "cd " & dirName & ";"
<br>set shellScript to shellScript & "~/Library/TeX/bin/pdflatexc  " & baseName
```

are simply a convenient way to concatenate this sequence of commands and input into one variable.

The final three lines update the Preview window.

To be sure, several of these lines are complicated, but these lines can be copied without change into new scripts.

Chapter 13

Editing Aids

TeXShop has several methods to speed the entry of source text in the editor. Each of these methods is optional, and each is customizable. Some users will ignore the methods (I'm one of them because I write slowly), some users will use one or more methods in the default form we provide, and some users will spend time customizing each to suit their exact entry style.

13.1   Emacs Mode

There is a famous text editor named **emacs**. This editor was written before computers had mice, so it had elaborate commands to move the cursor. There were keystrokes to move forward one character, back one character, forward one word, back one word, forward one sentence, back one sentence, etc. Additional keystroke commands were available for copy and paste. The great disadvantage of emacs was that the editor came with a two page table of instructions and took weeks to master. The advantage is that it allowed rapid entry of text without lifting a hand to reach the mouse. (Modern versions of emacs accommodate mice.)

The Cocoa text editing code used by TeXShop and many other programs on the Macintosh was originally written by engineers at NeXT; many of these engineers used emacs. So they added a vast number of emacs keystroke shortcuts to the editor. I don't know the history of this code, but I like to imagine that the emacs keystrokes were added secretly without telling Steve Jobs. Most modern Mac users are completely unaware of this feature.

For a small sample, hold down the control key and type the letter "f". Notice that the cursor moves forward one character with each keystroke. Replace "f" by "b" and the cursor moves backward.

Herbert Schulz has a useful website with many items for TeXShop users: `https://herbs.github.io`. One of these items is a folder with Key Binding documents and samples. Don't be misled by the folder title; much of the information in this folder is about emacs. The folder has a link to a site at Harvard with very extensive information, but sadly that site is gone. However, the material is still available at `https://github.com/jrus/cocoa-text-system#readme`. Using the green Code button, download the ZIP file and decompress. The resulting folder has several files including three crucial html files.

Emacs keystrokes were completely customizable and the shortcuts for the Cocoa editor are similarly customizable. This is done by placing a file named DefaultKeyBinding.dict in `~/Library/KeyBindings`. The structure of this file can be found with Google searches; some sites contain sample customization files. Note that this DefaultKeyBindings.dict file has nothing to do with TeXShop's Key Bindings, and instead is about customizing the Cocoa support for emacs features.

Essentially, the DefaultKeyBindings file is a list of keystrokes and calls to procedures in NSTextView, the Cocoa editing code. For example, one line might read

```
"^f" = "moveForward:";
```

Here ^f stands for Control-f and moveForward is the Cocoa procedure which moves the cursor one character forward, documented as follows:

```
- (void)moveForward:(id)sender;
```

A professor of mathematics named David Wang asked me to add a keyboard shortcut to TeXShop which moved the cursor past the next dollar sign. I wrote back that the appropriate way to do that would be to use the emacs key bindings system. It turned out that Wang knew about and used the emacs cursor moving shortcuts. But unfortunately, searching for a string is not a procedure that the Key Bindings system can use. So I added four calls to TeXShop:

```
- (void)moveForwardTo$:(id)sender;
- (void)moveForwardTo$$:(id)sender;
- (void)moveBackwardTo$:(id)sender;
- (void)moveBackwardTo$$:(id)sender;
```

These calls move the cursor from its current position to just after the next \$, just after the next \$\$, and so forth. Using the calls, the required shortcuts can easily be obtained. Below is a sample DefaultKeyBinding.dict file. If you want to test the feature, create this file and place it in /Library/KeyBindings. Then push the option key and f or b to move to an appropriate \$, or push the control key and f or b to move to an appropriate \$\$.

```
/* ~/Library/KeyBindings/DefaultKeyBinding.dict */

{
    /* Sample Math Mode bindings */
    "~f" = "moveForwardTo$:";
    "~b" = "moveBackwardTo$:";
    "^f" = "moveForwardTo$$:";
    "^b" = "moveBackwardTo$$:";
}
```

The above DefaultKeyBinding file is not appropriate for permanent use. For one thing, it shadows other commands already available in the default system. Most users should just ignore this new feature, which is only provided for users who have already customized the Key Bindings system. If they want to add the new item, they will need to edit their DefaultKeyBinding.dict file and choose appropriate keystrokes to add the new search.

Note that DefaultKeyBinding.dict adds keyboard shortcuts to *all* editors which use the Cocoa code, not just TeXShop. This file is read when a program first starts. It turns out that commands like "moveForwardTo$" not in a program's code base are ignored. So, for example, the above sample does not harm Apple's TextEdit.

User-defined edit commands used by Key Bindings cannot have parameters. Thus I could not expose general procedures like

```
moveForwardTo: (String *)s;
moveBackwardTo: (String *)s;
```

However, these two procedures are now in TeXShop, and the four procedures listed earlier just call them. This means that if you would like to write a keyboard shortcut which searches for a different string like "begin" or "end" and moves the cursor appropriately, you will need to add appropriate routines to TeXShop and recompile. But writing these routines would be trivial because you can just call my generic routines.

I'd be happy to add such special routines to TeXShop upon request, so they remain available after program updates. Just write.

13.2 Key Bindings

This section is about the TeXShop *Key Bindings* feature, which is independent of the previous Emacs material.

The idea behind Key Bindings is simple. Some rarely-used keys can be reprogrammed in TeXShop to enter something else, and this "something else" can be several characters rather than just one.

For example, a quotation on a typewriter would be written

```
He replied "that is not the point".
```

But when typeset by LaTeX, this input would become "that is not the point" rather than the desired "that is not the point". Key Bindings can reprogram the " character to input the correct symbols to start a LaTeX quote and the correct symbols to end that quote with the cursor positioned in the middle. If text is selected in the editor and then " is typed, Key Bindings can insert LaTeX quote symbols before the selection and after it.

There is an item in TeXShop Preferences under the Edit tab to turn Key Bindings on or off when documents are first opened. To be frank, I leave it off because I find it confusing to type a key that is supposed to produce one symbol and have it produce something else. But if you can get around that hangup, turn it on.

There is also an item in the TeXShop "Source" menu to toggle Key Bindings on or off. So if you use Key Bindings but suddenly need an unusual character that has been remapped, turn Key Bindings off, type the character, and turn it back on. If this situation occurs often for you, there is an optional tool for the Source Window toolbar which toggles Key Bindings on and off.

Finally there is another item in the TeXShop "Source" menu called "Edit Key Bindings File ..." This brings up an editor listing keys in a column on the left and the corresponding output in a similar column on the right. Extra items can be added, and existing items can be edited or removed.

Note the special symbols #SEL# and #INS#. If there is selected text when a remapped key is pressed, #SEL# will be that selection in the resulting output. The insertion cursor will be placed at the spot #INS# in the resulting output.

Although many items are remapped by the default Key Bindings, my guess is that only a few items are used, and users aren't even aware of the others. This seems to do no harm.

If you are one of the few users who wants to rewrite the Key Bindings list, it might be best to take a screen snapshot of the existing list first. Then if an item is accidentally mangled or deleted, it can be added again. However, there is an easy way to get back to the default list if editing breaks things. Go to ∼/Library/TeXShop and drag the folder "Keyboard" to the desktop. The next time TeXShop starts, it will recreate this folder with default contents. The folder contains a single file named "autocompletion.plist". Despite the name, this is the list of Key Binding modifications.

13.3 Command Completion

13.3.1 Introduction

The Command Completion feature was written by Greg Landweber. Command Completion in TeXShop allows you quickly enter commands and environment structures into your source document. It also allows you to easily move from one argument for a command to another using built-in menu commands. Command Completion is always on; there are no Preference settings or menu commands to turn it off. But it is unlikely that a casual user will accidentally trigger it.

In a TeXShop document, type

```
\begin{th
```

and then press the Escape key. TeXShop knows several standard LaTeX constructions which begin with these characters, and it cycles through them with each push of Escape. One of the elements in the cycle is the characters just typed; the other possibilities are

```
\begin{thebibliography}
\begin{theindex}
\begin{theorem}
```

However, TeXShop does not just finish the command word; it adds more. For instance, for the "theorem" choice it writes

```
\begin{theorem}
•
\end{theorem}•
```

and the first of the two stars is selected. The user can immediately type the theorem, and the first keystroke will replace the initial star. When the theorem text is complete, type control-command-F to jump to the second star and continue with the remaining source. This move can also be done by holding down the Option key and typing Escape.

13.3.2 Herbert Schulz and Command Completion

I've been too lazy to learn Command Completion. The expert on that system (and several other aspects of TeXShop) is Herbert Schulz. This section of the manual has been copied word for word from one of his documents. Schulz wrote a much larger and more detailed document as well, *Command Completion for TeXShop.* That document and other support material is in `~/Library/TeXShop/CommandCompletion`.

In the "Source" menu there is a submenu CommandCompletion/Marks with the items: Next Mark, Previous Mark. These items have keyboard shortcuts control-command-F and control-command-G. A regular user of CommandCompletion will remember these keyboard

shortcuts and thus easily enter source, jumping from star to star, without lifting a hand from the keyboard.

Some users prefer to use Tab rather than Escape for triggering operations. TeXShop has a Preference setting under the Source tab to make this change.

It takes practice to learn which commands have completions and which do not. For example,

    \begin{

has no completions because many TeX commands start with "begin" and it makes little sense to list them all. However

    \begin{t

has many completions, and

    \begin{theo

only has one.

While the list of abbreviations is fairly extensive, just learn a few that you need all the time and slowly pick up others as you need them. Press the trigger key multiple times to get the same commands and environments with differing numbers of options.

Some commands are abbreviated with the first few letters of the command. Others are created with a sort of nickname. The following are just a few samples of each type.

13.3.3   Completing Using First Few Characters

On a new line type `\newc` and then press `Esc` to get:

`\newcommand{`•`}{`•`}`

while successive presses of `Esc` give

`\newcommand{`•`}[`•`]{`•`}`

and finally

`\newcommand{`•`}[`•`][`•`]{`•`}`

all with the first "Mark" (•) selected.

You need only start to type the new command's name to replace the selected •. Then select the next mark by using the `Source` → `Completion` → `Marks` → `Next Mark` (`Ctl-Cmd-F` (or, alternatively, `Opt-Esc`) menu item and enter the next argument, etc.

You can complete many commands by starting to type them and pressing the Escape key. Variations on the commands with differing numbers of optional arguments are generated by additional presses of Escape. One example: typing `\sec` on a new line and then the Escape key produces

`\section{`█`}`

while a second press of Escape gives

`\section*{`█`}`

the *-variant of the command and a final press of Escape gives

`\section[`█`]{•}`

with the optional argument.

13.3.4   Abbreviations

In addition to command completion there also exist many abbreviations for commands. The principal difference is that an abbreviation is a mnemonic, a short name, rather than the start of a command name. There are many abbreviations for different environments; all of them start with a 'b'.

For example typing `benu` and then `Esc` at the beginning of a line will produce the complete enumerated list environment:

`\begin{enumerate}`
`\item`
█
`\end{enumerate}•`

as you might expect; the final Mark is there so it is easy to skip to the end of the environment using the `Next Mark` menu item. You can add an additional item to the list by typing `ite` and press `Esc` at the start of a new line to get and additonal

`\item`
█

inserted at that spot. Additional handy environment abbreviations are `bite`, `bali`, `bfig`, `btabl` and `btab` for the `itemize`, `align`, `figure`, `table` and `tabular` environments respectively; see the full list in the documentation.

Sectioning and font variation commands also have abbreviations. Typing `sec` and pressing `Esc` on a new line produces

`\section{`█`}`

while additional presses of `Esc` give the variations

`\section*{`▉`}`

and

`\section[`▉`]{•}`

in turn. Additional sectioning abbreviations are `ssec`, `sssec`, `par` and `spar` for `sub-section`, `sub-sub-section`, `paragraph` and `sub-paragraph` respectively.

The abbreviations for font changing commands can come in-line with text so you should use them with a leading \; \tt, \sf, \sc, \em, \bf followed by `Esc`, give `\textttt{`▉`}`, `\textsf{`▉`}`, `\textsc{`▉`}`, `\emph{`▉`}` and `\textbf{`▉`}` respectively.

13.3.5   Comments

It is easy to remember the arguments for commands that are used fairly often but forget them for those rarely used; these are the perfect candidates for comments. An example is the order of the arguments for the `\rule` command; type `\rul` and the Escape key to get

`\rule{`▉`<width>`▉`}{•<height>}`

and an additional press of Escape gives

`\rule[`▉`<lift>`▉`]{•<width>}{•<height>}`

the version with the optional argument. Another example is the `wrapfigure` environment, from the `wrapfig` package, which has multiple versions with differing numbers and positions of optional arguments. To see the variations with the comments type `\bwr` on an empty line and press Escape to get:

`\begin{wrapfigure}{`▉`<placement: r,R,l,L,i,I,o,O>`▉`}{•<width>}`
`•`
`\end{wrapfigure}•`

with the versions with optional arguments on succeeding presses of Escape key.

13.3.6   Other Environments

Environments that aren't built into the `CommandCompletion.txt` file can always be added if you use them a lot but there is an alternative for occasional use. Built into the completion algorithm is a way to complete environments. First press `\b` and `Esc` to get `\begin{`, enter the environment name and the closing `}` and then `Esc` again; the closing `\end{}` with the corresponding environment name will be generated on a separate line.

13.3.7 When Typing \ is Difficult

Some keyboard localizations make it difficult to type \ directly; e.g., it takes multiple keystrokes to do so using the French keyboard localization. Hope isn't lost! In most cases an abbreviation or start of command doesn't have to start with a \ but rather any 'white space character' (i.e., the start of a fresh line, a space or tab). So instead of

`\sec`

and the trigger key to produce

`\section{`■`}`

you can use

`sec`

at the start of a line and the trigger key will produce the same completed command.

Similarly, writing `tt` and Escape will give `\texttt{`■`}` since it is preceded by a space character. However `` `tt `` will *not* work since the `tt` *isn't* preceded by a 'white space character'; in that case you will have to use `` `\tt ``. The simplest way to make that easier is to create a macro that does nothing by insert a \ and assign it to a simple `Cmd` based keystroke.

13.3.8 Editing the Command Completion File

The location ∼/Library/TeXShop/CommandCompletion contains a file named Command-Completion.txt. This is the actual list of command completion elements defining the system. TeXShop has a built-in Command Completion editor, so consulting the file CommandCompletion.txt is never necessary. To use the built-in editor, select the menu item Source/CommandCompletion/Edit Command Completion File.

This is not the place to explain how to add items in detail; consult the larger Schulz document for that. Studying and copying items in the existing file will be enough to get started. But some questions will arise immediately because the file contains a large number of • symbols.

Recall the menu Source/CommandCompletion/Marks. This menu has three commands

```
Next Mark
Previous Mark
Insert Mark
```

But if the Option key is pushed, the items change to

```
Next Mark (Del)
Previous Mark (Del)
Insert Mark
```

and if the Control key is pushed, the items change to

```
Next Mark
Previous Mark
Insert Command
```

Each of these items has a keyboard shortcut. The Schulz document explains how these items can simplify entry of new command completion elements using the editor.

The creation of the existing Command Completion list was the joint work of several people. A few users have created their own versions of the file, and thus their own systematic ways to remember and name command abbreviations. Some of these are in ∼/Library/TeXShop/CommandCompletion.

Chapter 14

Automatic Saving; Finding Documents on Disk

14.1 Automatic Saving

Automatic Saving is a powerful feature introduced by Apple in Lion, macOS 10.7. TeXShop uses this feature, which is entirely implemented by code in Cocoa written by Apple. Only a single line of TeXShop source code mentions it.

With Auto Save, you can essentially forget about saving files. Files being edited will automatically be saved every few minutes. Only changes are saved, so there is minimal disk activity and saving is completely unnoticeable.

You do not need to save files before typesetting because this will happen automatically. If you quit a document or shut down the computer, all files will automatically be saved. If you copy a file, or drag the entire file to mail, or do similar operations, the file will automatically be saved and users will get the latest version.

Since the Macintosh is only saving changes, it has a complete record of all the versions of the document. Theoretically, you could go back to earlier versions and reconstruct the document as it was yesterday, or last month. As we will see in a moment, this isn't just theoretically possible — it is actually easy.

You might wonder if the file containing a document has all the old versions hidden away somewhere. Could I send someone the TeX source of a letter of recommendation which currently says "Paul is a creative student", but whose initial version said "Paul hasn't had a new idea in years"? Happily, no. Files on disk only contain the latest version.

Before AutoSave, files that had been edited were marked as "dirty" with a black circle

in the red close button at top left of the window. Now edited documents are marked by appending the word "- Edited" to the document's name in the window's title bar. The file menu still has a "Save" command. If you save in this way, the "- Edited" notice goes away. Apple sometimes calls this operation "Saving a version".

What happens if you decide to experiment, get confused, end up with a source document that makes no sense, and then discover it has been automatically saved out from under you, and the previous version of the file is gone? Since *undo* works across file saves, just undo the bad changes.

When I give talks about TeXShop at TeX User Group Conferences, users of Linux and Windows like to make fun of features of the Macintosh. So when I talked about Automatic Saving, I was sure the Linux users would warn about the danger of losing data. To my surprise, their actual comment was "we've had that for years!" A user has never complained to me about losing data due to automatic saving and I cannot imagine living without it.

Just in case, TeXShop has a hidden feature to turn AutoSave off. The hidden preference is

```
defaults write TeXShop AutoSaveEnabled NO
```

Using this preference is strongly discouraged. I never test TeXShop with AutoSave disabled.

14.2   Revert To

Iff you work on a document for several hours and then notice that a crucial paragraph or illustration is now gone, there is an easy way to recover. In the File menu, select the item "Revert to" and select the option "Browse All Versions." The Macintosh screen then switches to a Time Machine view showing the current document on the left and a receding stack of earlier views on the right. You can leaf through these earlier versions, noting the date and time that each was created. All of these views are alive, so you can copy and paste between versions, or restore an earlier version. This works even if you have not activated Time Machine. So it is a sort of "personal Time Machine just for TeXShop".

See the picture at the top of the next page for an illustration of "Revert To".
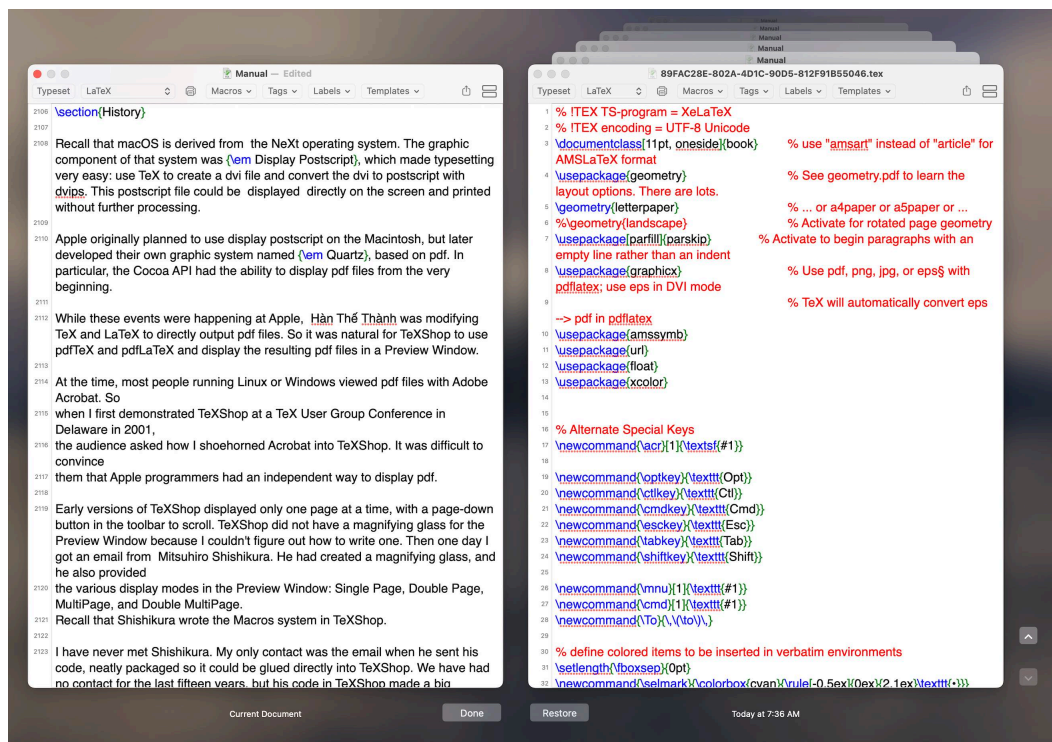
Figure 14.1: Revert To

14.3 Lock

Often users have a folder of old papers which they haven't edited in a long time, but refer back to from time to time. If you open such a document and click the title, a dialog will open allowing you to "lock" the document. When a document is locked, it cannot be saved or changed. If a document is locked, clicking on the title and unlocking the document makes it possible to edit again.

14.4 Finding the Disk Version of a Document

In TeXShop and most other programs on the Mac, you can click the title of a window while holding down the command key and see a complete path to your document, listed as a series of entries in a drop down menu. By selecting an item in this menu, you can directly go to that spot of the file system.

Chapter 15

Preview Window

15.1 History

Recall that macOS is derived from the NeXt operating system. The graphic component of that system was *Display Postscript*, which made typesetting very easy. Use TeX to create a dvi file, convert the dvi file to postscript using *dvips*; display the postscript file.

Apple originally planned to use display postscript on the Macintosh, but later developed their own graphic system named *Quartz*, based on pdf. In particular, the Cocoa API had the ability to display pdf files from the very beginning.

While these events were happening at Apple, Hàn Thế Thành was modifying TeX and LaTeX to directly output pdf files. So it was natural for TeXShop to use pdfTeX and pdfLaTeX and display the resulting pdf files in a Preview Window.

At the time, most people running Linux or Windows viewed pdf files with Adobe Acrobat. So when I first demonstrated TeXShop at a TeX User Group Conference in Delaware in 2001, the audience asked how I shoehorned Acrobat into TeXShop. It was difficult to convince them that Apple programmers had an independent way to display pdf.

Early versions of TeXShop displayed only one page at a time, with a page-down button in the toolbar to scroll. TeXShop did not have a magnifying glass for the Preview Window because I couldn't figure out how to write one. Then one day I got an email from Mitsuhiro Shishikura. He had created a magnifying glass, and he also provided the various display modes in the Preview Window: Single Page, Double Page, MultiPage, and Double MultiPage. Recall that Shishikura wrote the Macros system in TeXShop.

I have never met Shishikura. My only contact was the email when he sent his code, neatly packaged so it could be glued directly into TeXShop. We have had no contact for the last

fifteen years, but his code in TeXShop made a big difference.

The Shishikura code was used until Apple introduced macOS 10.4, Tiger. In that system, Apple made available a framework named PDFKit, which worked more or less the same as Shishikura's, providing the same display modes and the like. I replaced his code with PDFKit because it is important for future developments to keep up-to-date with Apple technologies. The magnifying glass has gone through many iterations as version after version broke when Apple made API changes. Finally I talked directly to the author of PDFKit at WWDC and found a stable technique.

In the early days of macOS, strange bugs would sometimes surface in Apple's pdf display technology, but the Apple engineers were quick to fix these bugs. Very rarely these problems can surface even today. Users should be aware of a trick to help locate these bugs. Most pdf software on the Macintosh uses PDFKit, including Apple's Preview and Skim. However Adobe Acrobat Reader and Adobe Acrobat use Adobe's own software to display pdf. So a bug which occurs in TeXShop, Preview, and Skim, but not Adobe Acrobat Reader, is a bug in PDFKit. A bug which occurs in all four of these programs is a bug in the software which created the pdf file. And a bug which occurs only in TeXShop — perish the thought.

15.2   Display Modes

The Preview window can display a pdf as one long scrollable document; this is called MultiPage Mode, and is the recommended mode. It can also display two pages side by side in a long scrollable document. This is called Double MultiPage Mode. It can display one page at a time, called Single Page Mode, or two pages at a time, called Double Page Mode. Menu items in the Preview menu move to the Previous Page, or to the Next Page. Items in the Preview Toolbar also perform these tasks.

As usual, items in TeXShop Preferences under the Preview tab select the behavior to use when documents first start, but the mode for a particular document can later be changed using an item in the Preview menu.

The modes showing two pages side by side are controlled by a TeXShop Preference item under the Preview tab, which allows users to select "Pages Flow from Left to Right" or "Pages Flow from Right to Left". Select the first item for languages written from left to right and the second one for languages written from right to left, that is, Arabic, Hebrew, and Persian. By default, the first page is shown by itself, followed by pairs of double pages. This is called *Book Mode* by Apple because most books start with an odd number of preliminary pages, followed by the regularly numbered pages. Therefore the pairs of numbered pages correspond to the pairs of visible pages when a book is opened at random.

There is a hidden preference item for those very rare users who do not want that isolated starting page. To activate it, open Terminal and type

```
defaults write TeXShop DisplayAsBook NO
```

In addition, two new special comment lines are provided so the property can be set on a document by document basis. These overrule the defaults and turn book display mode on or off for that document:

```
% !TEX bookDisplay
% !TEX standardDisplay
```

In Japan, text can be written horizontally or vertically. When it is written vertically, pages should appear from right to left. Therefore we also provide special comment lines to set "displaysRTL" on a document by document basis:

```
% !TEX PageDirectionL2R
% !TEX PageDirectionR2L
```

15.3 Magnification

The magnification of the Preview Window can be set to "Fit to Window", "Fixed Magnification", and "Actual Size". The recommended choice is "Fit to Window", which adjusts the magnification of the window so the text exactly fits side-to-side. On large monitors, it is convenient to adjust the window so it shows exactly one full page of output. On smaller monitors, adjust the window so it is wide enough to read easily; only a portion of each page will be shown, but scrolling easily reveals the remaining parts of the page.

The other choices are "Fixed Magnification" and "Actual Size". In a sense, these choices are the same. The first allows the user to select the amount of magnification, which is set in TeXShop Preferences under the Preview tab by an item named "Fixed Magnification Amount". This item can be reset for an individual window using the "Scale" item in the window toolbar. The "Actual Size" item behaves the same way, except that the computer now selects the magnification amount based on the actual size of pages in the pdf document.

Note that "Fit to Window" and "Fixed Magnification" are orthogonal choices. If you select "Fit to Window" in TeXShop Preferences, then the "Fixed Magnification Amount" just under it in Preferences will be ignored, because the magnification is actually set by the size of the window. If you open a window in "Fit to Window" mode, and then modify the magnification with the "Scale" tool in the toolbar, the magnification mode will silently change to "Fixed Magnification Mode" so that resizing the window will no longer change the magnification. To restore "Fit to Window" mode for that window, reset the "Magnification" method in the Preview menu.

If you use "Fit to Window" mode, but want to temporarily magnify the image so a portion can be studied in detail. I recommend using an Apple gesture rather than the Scale tool.

Click in the window, place two fingers on the track pad, and spread them to magnify or bring them together to shrink. When you shrink back to the original "fit to page" magnification, the display will snap back to this configuration. Note, however, that this gesture changes to "Fixed Magnification" mode and resizing the window will not change its magnification. To restore "Fit to Window" mode for that window, reset the "Magnification" method in the Preview menu.

A very small number of people use TeX to create intricate pdf diagrams which they then view in TeXShop for long periods of time at very high magnification. TeXShop is not really designed for that sort of work flow and if bugs occur, I recommend searching for other appropriate tools.

15.4   A Mysterious Preference Item

TeXShop Preference items for the Preview Window are mainly found under the Preview tab. This page also contains items for Single Page mode, and for the HTML Preview window. Single Page mode has already been discussed in this document, and the HTML Preview window will be covered later.

At the top of the second column of the Preview tab, there is an item labeled "Change Page Buttons" with a checkbox titled "Do not scroll". What does this item do? I haven't got the slightest idea.

At the start of this chapter, I explained that the original TeXShop used a preliminary version of Apple's PDF code to display pdf documents, together with extensions by Mitsuhiro Shishikura. Some of that old code is still in TeXShop, used to display graphic files and dvi files, as explained later in this document. But most of the code has been replaced by code which calls Apple's PDFKit. The old code is in a source file called MyPDFView, and the new code is in a source file called MyPDFKitView, and for years and years I have only edited the MyPDFKitView file.

I looked at the code in Preferences which controls the mysterious "Change Page Buttons" item. It controls a global preference which is only consulted by the code in MyPDFView. Whatever this code did was not ported over to MyPDFKitView during the transition to PDFKit in Tiger. So you can ignore that preference setting.

15.5 Preview Mouse Tools

The Preview toolbar has an item determining the behavior of the mouse in the window. There are five choices:

- **Scroll**
  scroll the pdf document in directions allowed by the scroll bars

- **Select Text**
  select text in the pdf for copying and pasting into another text document

- **Magnify**
  a small magnifying glass

- **Magnify Large**
  a larger magnifying glass

- **Select Pdf**
  select a region of the pdf, for copying and pasting into another graphic program

An item in TeXShop Preferences under the Preview tab selects the default tool when the Preview window first opens.

The two magnifying glasses offer more options than first imagined. Click and hold down the mouse at a spot to magnify the region near that spot. Double click and hold down the mouse to magnify a larger region, and triple click and hold down the mouse to magnify the complete window. While the mouse is down, move it to magnify different regions of the window.

Hold down the command key, or option key, or control key to increase the magnification by various amounts.

Hold down the shift key and the command key, option key, or control key to decrease the magnification by various amounts.

The "select pdf" tool offers more options than first imagined. Select any portion of the pdf document. This can contain text, illustrations, or both. Copy using command-C and then paste into any document which accepts pdf data. For instance, open Preview and select "New from Clipboard" to open the selection in Preview.

Drag and drop also works with graphic selections. For instance, click and hold down the mouse inside the selection and then drag everything into Apple's Keynote to copy the information to a slide. Dragging the selection to the desktop creates a file containing the pdf information, with a default name. Rather than dragging and dropping the selection, select the item "Save Selection to File.." in the Preview menu to name the corresponding file and the location on disk where it is saved.

An item in the Misc1 tab of TeXShop Preferences determines the type of graphic data created for selections. By default, pdf data is created, but various types of tiff, jpeg, and png data are also possible. This choice can also made in the Preview menu for an individual selection using the "Copy Format" menu.

15.6   Back, Forward, Rotation

The toolbar has two arrows pointing up and down for scrolling by a page, and two arrows pointing left and right for moving through the history of a document. These left and right arrows act like the corresponding arrows in a browser. If you use the "Goto page" tool to go from page 3 to page 5, and then page 10, and then page 25, the left arrow will take you back to page 10, and then to page 5, and at that point the right arrow will take you back to page 10.

The Preview menu also has menu items to rotate individual pages clockwise or counterclockwise.

15.7   The Drawer and Two Search Fields

The Preview Window has an associated Drawer, which can be toggled visible or invisible with an item in the window toolbar or the first item in the Preview menu. This drawer appears on the right side of the window if there is room there, and otherwise on the left side of the window. It's contents are divided into two pieces. The top pieces shows an outline of the document, listing chapters, sections, and subsections if available. Clicking on an item takes you to that point in the actual Preview Window This top portion is blank if the pdf document does not contain an outline. Outlines are automatically created by the hyperref package in LaTeX.

The bottom of the drawer is a search field. Type an entry in the search field and a list of appearances of the entry will be created below the entry. The page where the entry occurs, and some of the surrounding text will be shown. Click on an entry to be taken to that spot.

The Preview Window has an alternate search tool in its toolbar. Enter an entry in this field and the first appearance of the entry will be selected. Type command-G to go to the next entry and shift-command-G to go to the previous entry. If the edit window is active, command-F will go to its Find Panel or Find field. If the Preview window is active, command-F will go to the search field in its toolbar.

15.8 Following Links and URLs

The package hyperref can create links from one portion of a document to another. These links are active in the Preview window, so a reference to a theorem can be followed to the action theorem, a reference to the bibliography can be followed to the actual item, and so forth. In addition, url's are active, so clicking a url will open the corresponding page in Safari.

Hovering over various links will bring up a small view of the linked text. For instance, hovering over a section number in the table of contents will display the beginning of that section in the text.

15.9 Too Much Magnification is Bad

One user liked to increase the default magnification significantly in the Preview Window, and in addition use the magnifying glass for extremely close inspection of fine details. This caused TeXShop to run out of memory and crash. TeXShop now protects itself by refusing to activate the magnifying glass if the default magnification is too large.

To understand the problem, it is useful to know how the magnifying glass works. When a user asks for the glass, TeXShop glues an extra view on top of the Preview window's content, but this view is transparent so the user does not notice that it is there. Then TeXShop creates an offscreen image of the entire active page being displayed on the screen. Next TeXShop reads the data from this offscreen image which is contained in a small rectangle centered on the mouse location, and draws this data to a larger rectangle in the transparent view. The result is a magnified view of the portion of the screen under the mouse; this magnified image is in the mostly transparent view rather than in the original Preview page.

If the user moves the mouse, TeXShop erases the transparent view and repeats the procedure with the new mouse location. The magnified image appears to move to a new location, but the uncovered data need not be redrawn.

If the Preview window has a large default magnification, then only a small portion of the window is visible on the screen. But the offscreen image of the active page will contain the full page in intricate detail, taking a gigantic amount of memory. Hence the out of memory bug.

Chapter 16

Printing

To print a TeX or LaTeX document, select "Print" in the File menu. A standard print dialog will appear, allowing you to print everything or selected items, and set several parameters depending on the capabilities of the printer. This can also be activated by typing command-P.

To print the TeX source for a document, select "Print Source" in the File menu. Users do not often perform this task, so it has no keyboard shortcut.

Bruno Voisin called my attention to new support in LaTeX for PDF files, added as of the June, 2022 release. He sent the following illuminating source file. The source creates a document with three pages. The first and third pages contain text formatted in the standard way, but the second page is in landscape mode showing a wide picture. If this output came from a book, all three pages would be standard pages and the user would rotate the book to look at the picture. But when the author is creating the pages on a computer screen, the second page should be rotated for easier viewing. This source file causes the second page to be appropriately rotated. If you want to try this example, supply your own graphic file.

```
% !TEX program = XeLaTeX
\DocumentMetadata{}
\documentclass[12pt]{article}
\usepackage{graphicx}
\usepackage[figuresright]{rotating}
\ExplSyntaxOn
\AddToHook{env/sidewaysfigure/end}{\pdfmanagement_add:nnn{ThisPage}{Rotate}{90}}
\ExplSyntaxOff
\begin{document}
```

```
Some text.
\begin{sidewaysfigure}
\centering
\includegraphics[width=\textheight]{YourPicture.jpg}
\caption{Then a nice rotated figure.}
\label{fig-stprof}
\end{sidewaysfigure}
\clearpage
And more text.
\end{document}
```

When TeXShop prints pdf files containing rotated pages, all pages are printed in portrait mode and information is lost on rotated pages. The solution to this problem is easy. Before printing, comment out the three lines starting with "ExplSyntaxOn" and typeset. Then no page needs to be rotated in the printed output.

One problem remains. Suppose you are creating a document which most users will read on the screen, but a few will print. Should pages with sideways-figures be rotated or not? This is a problem I cannot solve, since the fix must be in the software your readers use to display pdf, not the software you use to create it.

Chapter 17

The Console

A special tab in TeXShop Preferences contains preference items for the console. The font used by the console, the console position, and other items can be set there.

Many typesetting engines format output text to the console with lines of fixed length. A setting allows the console to be resized arbitrarily, or only vertically. Some users like to temporarily resize it arbitrarily and adjust the size to be slightly wider than this conventional fixed width. Then they reset the preference to only allow vertical resizing. This makes it impossible to accidentally change the width.

A final setting controls the placement of the console. One possibility is for it to appear in front of all other windows during and after typesetting. Another possibility is to bring it forward, typeset, and then push it behind the edit and source windows if there is no error.

Chapter 18

HTML, Interactive TeX4ht, PreTeXt

18.1 Html Preview Window

Starting with TeXShop 5.00, a new preview window is available in TeXShop, able to display html source as a live web view. The window is provided because several TeX and LaTeX engines produce html output. For the moment, we ignore these engines and examine the window.

Select the item "Show HTML Window" in the Preview menu. A default web document will open with links to to several active web sites. Click the last link to see an illustration, and click the TUG link to confirm that the window can access live content. When the page first appears, it may be small and in an awkward location. Resize it and move it to a reasonable spot, perhaps where the Preview Window usually appears. Select it and choose the item "Save HTML Window Position" in the Preview menu. The window will then always open with this size and position.

In TeXShop Preferences under the Preview tab, the item "Home URL" allows you to set the web page that opens by default when the window is selected.

Although the window contains a full featured web view, the toolbar items are minimal. Our window only provides Back and Forward buttons, a field to enter a new URL, and a search field. The search field currently does nothing. You already have Safari and perhaps other browsers; TeXShop has no obligation to provide a duplicate.

When actually using the Html Window, you will almost never call "Show HTML Window." What is the purpose of the window?

18.2 Html Files

TeXShop has a large "Changes" document, available in the Help menu. "Changes" is clearly a pdf document because when it is resized, the text shrinks or enlarges rather than reflowing. However, this document is actually written in html, and that html version is available on the TeXShop web page as `https://pages.uoregon.edu/koch/texshop/changes_3.html`. Notice that this web document reflows when it is resized. To create the pdf form, I print the document and then ask that the resulting pdf be saved in a file.

Until recently, I wrote Changes source using TeXShop and opened the html file in Safari to check for mistakes. This was tedious; for one thing, Safari opened the document at the beginning and I would have to scroll down to find the section currently being written.

But now this and similar html files can be previewed directly in TeXShop. Check to see if **html.engine** is one of your active engines. If not, go to `~/Library/TeXShop/Engines/Inactive/html` and move a copy of the file html.engine to the active folder `~/Library/TeXShop/Engines`. Then find an arbitrary html file; I'll use Changes.html. Html files begin with a mandatory first line. Just below this line, add the comment shown below.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" \\
        "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">

<!--
% !TEX TS-program = html
-->
```

Since this addition is an html comment, it is ignored on the web. But TeXShop sees the line inside, which tells it to typeset using the html.engine. No actual typesetting is done by this engine; instead, it just opens the html file in the new Html Preview Window. On the left side of the screen we have the source in an edit window, and on the right side we have the resulting web view.

Type new content on the left and then type command-T. The new content immediately appears on the right and no scrolling is needed because the revised html opens scrolled to the correct spot. What a time saver!

If you create complicated web pages, use special software designed for that purpose. But for small tasks like the Changes document, TeXShop has become an adequate authoring tool.

18.3 Help File for Html Commands

I'm no html expert, so when writing Changes I often have to use Google to find appropriate commands. The TeXShop Help menu now has a document named **HTML Commands** to

make this easier. This opens a window with a list of tasks and appropriate html solutions. But it has got to be the ugliest help document ever, because it is written in html.

Notice, however, that the top of the document contains an instruction to typeset using the html.engine. You are supposed to open this source and then typeset to get a readable document. Try that now. The web version is easy to read and explains, among other things, how to embed images, movies, and UTube videos in a web page. In particular, if you right click on a UTube video in a browser like Safari, a contextual menu appears allowing you to copy code which can be pasted directly into an html document and thereby provide that video for viewers of your page. I found a lecture by John Maynard, one of the four Field's Prize winners at the International Congress of Mathematicians for 2022. It is fun to watch this video for the depth and clarity of his mathematics. Here is the code provided for this video, which is Greek to me:

```
title="13. Large gaps between primes in subsets - James Maynard
(University of Oxford) [2017]" frameborder="0"
allow="accelerometer; autoplay; clipboard-write;
encrypted-media; gyroscope; picture-in-picture" allowfullscreen></iframe>
```

When the html help document is edited in TeXShop; the changes will automatically be saved. So when you need a command not listed, look it up using Google and then add that example to the help file. The initial document provided in TeXShop doesn't list many commands because I don't know much html, but if you expand it, it will gradually become more useful.

The actual source for the help document is contained in a new directory `~/Library/ TeXShop/HTML`, and the source is named "Help.html". This folder can also contain support files. It currently contains an image "topdot.jpg" used by Help.html and a few other support files, and you can add similar support files. When TeXShop is updated, Help.html and the various support files will not be touched, because you may have edited these items. But the HTML folder also contains a subfolder named "Inactive" and that folder will be updated. So more extensive help files may be provided in the future, which you can merge with your own additions to the default file. I encourage users to contribute better initial Help.html files for this Inactive folder.

18.4   TeX4ht

TeX4ht is a program written by Eitan M. Gurari. It uses a standard LaTeX source file as input, but outputs an html file instead of a pdf file. In 2009, Gurari was preparing a talk for the TeX User Group conference about an extension which outputs braille, when he suddenly died. His TeX4ht project has been taken over by other volunteers and is now being very actively developed by Michal Hoftich.

Check to see if **TeX4ht.engine** is one of the active engines. If not, go to `~/Library/TeXShop/Engines/Inactive/TeX4ht` and drag the engine TeX4ht.engine to the active folder `~/Library/TeXShop/Engines`. The TeX4ht folder also contains a subfolder titled Example. Make a copy of this entire folder and drag it to your home directory or another directory used for TeX projects. Inside this folder is a LaTeX file named Sample.tex. Open it in TeXShop. A Source Window and a pdf Preview Window will open. Look carefully at the source file and notice that the first line is a magic comment saying to typeset using the TeX4ht engine.

Typeset. Now alongside the pdf Preview Window there is an Html Preview Window. The pdf window displays the output when typeset by pdfLaTeX and the Html window displays the output when typeset by TeX4ht. This makes it easy to compare the results. Resize both windows. They behave differently. If you have a trackpad, resize both window contents with a finger gesture. Both resize. In particular, finger gestures can magnify the html image. Notice that the mathematical equations are clear and precise in both versions.

Each document contains an active link at the end. Click these links. The link in the PDF Preview Window causes Safari to open and display the new page. But the link in the HTML Preview Window opens the linked page directly in this window. The Back arrow can take you back to the original content.

Add extra material to the source and typeset again. Notice that both windows are immediately updated. Neither window jumps to the top of the document when reloaded. For HTML, this is a major advantage over the old method of opening the output in Safari and clicking the reload icon when changes are made.

Math4ht has many parameters controlling its operation. Originally it created a large number of small pictures for mathematical equations, and the web page displayed these pictures. It can also output MathML and many modern browsers can interprete these commands. An even nicer approach is to render mathematics using MathJax. This is the approach recommended by the American Mathematical Society and produces very acceptable typeset equations. Users will want to create their own engines to explore the advantages of these various approaches. The TeX4ht engine uses MathJax, but an alternate engine is also provided in Inactive/TeX4ht which uses MathML.

18.5   Historical Intermission

The material below, copied directly from the Changes document, tries to put this new feature in a historical context.

The TeX User Group (TUG) was founded in 1980, shortly after the appearance of TeX. It publishes a journal, TUGboat, three times a year and creates the distribution TeX Live,

which works on almost all computer platforms. TeX Live is created in collaboration with similar TeX User Groups in other countries. TUG holds an annual meeting each year, sometimes in the U.S. and sometimes abroad.

In 2001 after MacOS X was released, I was invited to speak at the national TUG conference in Delaware by Wendy McKay; TUG offered to pay my way. I knew almost nothing about TUG, but imagined a conference of 2,000 users. We would meet in a large auditorium and listen to talks on solving common typesetting issues. Perhaps 100 Mac people would be there, excited to hear anything about MacOS X. When I got to Delaware, I discovered that there were 45 people at the conference and almost every one was on the program. They paid their own way. I sheepishly refunded the travel money back to TUG. There were only three or four Mac folks, but they all became significant in my life.

Wendy worked for Jerrold Marsden at CalTech. Marsden was a leading authority in classical mechanics, and is the author of many well known mathematical textbooks. He used a Macintosh to write mathematics very rapidly, and invited TeX experts to CalTech to solve TeX problems. Among these was Ross Moore from Australia, who was also at the Delaware conference. He knows more deep facts about LaTeX than anyone I know. Hans Hagen, author of ConTeXt, was also at Delaware.

Delaware had a special honored guest, Hàn Thế Thành. His pdftex program added code to Knuth's TeX which output pdf files rather than dvi files. His project was absolutely crucial for TeXShop because otherwise I'd have to make a dvi viewer, far beyond my capabilities. Thành came from Vietnam and his pdftex work was part of his PhD thesis from Masaryk University, Brno, Czech Republic. He talked about that thesis at the conference, beginning with a detailed image of a page from the Gutenberg bible. This remarkably beautiful page had absolutely straight right margins. But when you measured with a ruler, they turned out not to be straight; instead commas and periods were pushed slightly into the margin, creating the illusion of a straight side. In his thesis, Thành added that ability to pdfTeX and pdfLaTeX.

I began my talk in Delaware by installing TeX onto a new MacOS X machine using Gerben Wierda's installer. In the process, the Finder crashed and I spent ten seconds restarting it. After the talk, a participant said to me "I'm not interested in your program, but it is amazing that you could restart the Finder without rebooting the machine."

I have been to many TUG conferences. In Hawaii, the organizers pulled us out to look at the sunset because Mercury was visible close to the horizon. A few days later they pulled us out at noon because on that one day at that one time, the Sun was directly overhead. In a recent conference at Palo Alto, I met my collaborator Yusuke Terada, from Japan, for the first time. Shortly after we met, I looked over his shoulder at another participant, and asked Terada "do you know who that is?" It was Donald Knuth.

Already in Delaware, there were talks about XML. The speakers directly typed XML code

during their talks, and the indentation grew and grew until it was two pages right of the original margin. It took me several years to understand the goal of these talks. XML is easily parsed by computers and LaTeX is not. Authors submit papers to publishers already formatted in a personal way, but the publishers need to reformat these articles to conform to the standard of a journal or book series. This can be done automatically by computers if the paper is in XML, but requires work by hand for articles in LaTeX. All well and good, but I stopped paying attention to XML talks.

There were also pessimistic talks by famous figures in the TeX world, predicting that TeX would be dead in another five years. Most of these talks were from England, and I gradually realized that many speakers had jobs in the Open University system. For these jobs, they didn't want to produce static articles, but rather interactive documents allowing students to input information and try experiments. Their students had portable machines and worked remotely. HTML web documents were much more useful than static LaTeX documents. Eventually I stopped listening to these talks as well. It is now twenty years later and LaTeX is very much alive.

In my little corner of the world, TeX has created a revolution. Our mathematics department used to employ three secretaries to type mathematical manuscripts. They are gone. Today if you go into a mathematician's office, there will be a computer running LaTeX and a chalk board. Research is done with these two devices and immediately typeset and sent to collaborators in Europe. All of our PhD students use TeX for quizzes and exams, and when we hired younger hot shots as Assistant Professors, they required graduate students to submit homework written in LaTeX. If research articles in some other field must be in Microsoft Word, that's their problem, not mine.

Then Covid hit. Luckily I'm retired, but I watched my mathematical colleagues switch to remote teaching with only a week's notice. I don't know how they did it. Gradually I realized that those pessimistic talks contained an important message.

Some of my colleagues at the University of Oregon switched to PreTeXt to write lecture notes; I'll describe that project in a moment, but suffice it to say that one goal of that project is to allow authors to write interactive material which can be accessed over the web. Authors write using XML. Perhaps I should have listened to those talks after all.

Here's my "dream setup for the future." A faculty member would write lecture notes in LaTeX. These notes would be typeset by an engine which outputs both pdf and html. The pdf document would contain the course in final polished form, so students could look back at the clear logical direction of the lectures. The html document would contain interactive material: movies, questions with student input, etc., so students would be actively involved, even if working remotely.

The HTML Preview window has been added to support such software, and to encourage new software which makes it easier for authors to include interactive material in their

mathematical documents.

18.6   TeX4ht and Interactive Documents

After the first version of TeXShop with an HTML Preview Window appeared, I communicated via Karl Berry with Michal Hoftich, and then directly with Michal. Recall he is the author of TeX4ht. Essentially I asked if "interactive" documents can be written using TeX4ht. That boiled down to the following concrete questions:

- Can TeX4ht source code be written which sends some material only to the pdf document, and other material only to the html document?

- Can the material for the html document be written directly in html?

- Writing mathematical equations in html is difficult. MathML is extremely clumsy if users have to write it by hand. If material for TeX4ht can be written in html, can the mathematical portion of this material be written in standard LaTeX?

- If TeX4ht can accept source written directly in html, can that source contain movies and UTube videos? Can it contain interactive Sage programs? Can it contain graphing programs where the user gets to enter the formula of the function to be graphed? Can the document contain three dimensional graphics which can be rotated directly by the user?

The amazing answer to these questions is yes, yes, yes, and yes.

There is no need to explain these answers in detail, because TeXShop contains a demonstration program which illustrates all of these features. Using a TeXShop menu item, go to `~/Library/TeXShop/New`. This directory contains a subdirectory named **Demo**. Drag a copy of this folder to your home directory or where you keep standard LaTeX document folders.

First read the document **Preparing-for-the-Demo** in this folder. Follow the simple instructions to make sure TeXShop is correctly configured for the demo.

Next open **Fourier.tex** in the **Fourier-for-TeXShop** folder. This document contains lecture notes on Fourier Series which I happened to have around the house. Typeset the document to convince yourself that it is absolutely standard LaTeX, illustrations and all. Leaf through the pdf to notice that there are many mathematical formulas.

The first line of this document tells TeXShop to typeset using pdflatex. Change the word **pdflatex** to **TeX4ht**. Typeset.

The result will be *two* Preview Windows, one for the pdf form of the document and one for the html form of the document. Notice first that the mathematical formulas in the html

version are crisp and clear. Resize this window and notice that the text reflows. Resize the pdf window and notice that the magnification changes and the text does not reflow. Thus standard Preview is showing a pdf document and Html Preview is showing an html document.

Skip to section 9 of the two documents. Notice that this section has a different name in the pdf and the html versions, and the contents are completely different.

Scroll down to the end of section 9 in the html document and notice that it contains some standard mathematical formulas, very clearly typeset. It will turn out that they are written in the html section of the document using standard LaTeX.

Scroll up to the subsection named **Calling Sage**. A few lines below is a command

```
plot(sin(x), (x, 0, 2*pi))
```

Push the Evaluate button to get a plot of this function. The plot was performed by the open source program Sage, acting from a remote server which provides free access.

The plot command for this function can be edited by the user. Change it to

```
plot(sin(x) + cos(2 * x), (x, 0, 2*pi))
```

and plot again.

Just below this example is a 3D plot. Evaluate that. Grab the graph and rotate it on the screen.

It is possible to run complete Sage programs. Look at **A Calculus Course Experiment** and push Evaluate. We obtain a demonstration of Riemann sums. Notice that many items can be changed by the reader: the function, the number of divisions, the method of computing the height, etc.

The Demo folder contains a file named **Interactive-TeX4ht-Documents**. Read this along with the Fourier demo to understand the details of how the sample was constructed, and thus the answers to my four questions for Michal Hoftich.

18.7 PreTeXt

At the 2014 TUG conference in Portland, Oregon, I heard a talk by Robert Beezer of the University of Puget Sound on Mathbook XML. A former PhD student of mine named Tom Judson was converting a textbook he had published into the new interactive XML format. I talked to Beezer, but the conversation soon switched to a description of how Beezer and Judson met, which was via very serious bicycling in France.

A couple of years later, I found myself in a group of alumni of the University of Oregon, wearing suits and talking about possible donations. As rapidly as possible, I retreated to the mathematics building where people wearing jeans talked about mathematics. One of the faculty members there asked if I knew anything about PreTeXt, a way of writing interactive mathematics that he and other faculty were using. I didn't, but I looked it up on the internet. Turns out, Beezer's project had been renamed.

PreTeXt is a fascinating project. The central idea is that authors enter formatting commands using a special form of XML. But mathematics is entered in standard LaTeX, so writing documents is quite straightforward and not nearly as verbose as documents entirely in XML. The source is then translated into other formats using XSL. In this way, PDF, HTML, EPUB and other versions can be created automatically from the source. All of this is described very clearly on the web site https://pretextbook.org

In 2019, I spent a considerable amount of time revising TeXShop so it would be useful in this project. I added some new engines for PreTeXt, provided appropriate syntax coloring for XML documents, and made other changes. A description of all of this is in the folder `~/Library/TeXShop/Engines/Inactive/PreTeXt`. This folder contains several typesetting engines for PreTeXt, including an engine which shows the PreTeXt source in the Source Window, the pdf output in a Preview Window, and the HTML output in an HTML Preview Window. To get started, read the document **PreTeX with TeXShop** in this PreTeXt folder.

An enormous advantage of PreTeXt over similar projects is that Robert Beezer puts substantial effort into the interactive elements of the language. He organized a small group of authors who have published mathematical textbooks and are now converting these textbooks into interactive web documents. Their very practical demands have driven much of the recent work.

The PreTeXt project makes initial steps very easy. The folder `~/Library/TeXShop/Engines/Inactive/PreTeXt` contains a small shell script called updatemathbook.sh. Place this script in your Documents folder and run it in Terminal. It will download a folder named mathbook from Beezer's site. This contains all the software to use PreTeXt, together with many example folders. Since PreTeXt is an active project, updating is useful and easy. Just rename the old mathbook and then run the script again.

Inside mathbook is a folder named examples, which has many subfolders. Among these are sample-article and sample-book. The document **PreTeX with TeXShop** mentioned earlier carefully explains how to typeset these two projects in TeXShop, and provides appropriate typesetting engines for them. In particular, the book example uses portions of Tom Judson's Algebra Book, and thus is a real example rather than just a small taste.

18.8 Special Features of TeXShop for html, xml, and ptx Files

PreTeXt source files have extension .xml or .ptx. They are text files containing a series of tagged entries, just like html. TeXShop can open and edit html, xml, and ptx files, and when it does, it alters its behavior slightly, as indicated below.

18.8.1 Syntax Coloring

The old syntax coloring methods are used for LaTeX embedded in these files, but in addition five extra colors are used for the tags. These colors, and all other TeXShop colors, can be adjusted in TeXShop's Themes Preferences, to be described in a later chapter.

The "Comment" color is for xml comments and the "Tag" color is for xml tags. Some tags contain extra information; for instance, the minimal.xml file contains

```
<section xml:id="section-textual">
```

In this example, xml:id receives the "Attribute" color and section-textual receives the "Value" color. Finally, some rare commands have the form &lt and &amp and in these cases, lt and amp are given the "Escape" color. Here &lt represents < and &amp represents &.

18.8.2 The Tags Item in the Toolbar

In LaTeX mode, the Tags menu lists all

```
\chapter, \section, \subsection
```

commands in a source file, identified by associated text. Selecting an item takes the user to that location in the source. The Tags menu works the same way in xml mode, but this time it lists all

```
<chapter>, <section>, <subsection>
```

locations, with associated text. Indeed, the command recognizes eleven distinct tags. Since this is a large number, it can cause very long scrollable Tags menus. So under the Misc2 tab in TeXShop Preferences, an item lists all eleven possibilities and allows the user to select the items that create entries in the menu. It is possible to mark your own important spots in the source using the command

```
<!--!mytext-->
```

Here "mytext" can be replaced by any desired string. These locations will be listed as tags, with accompanying text. It is not necessary to remember the syntax; just select a

location in the source and choose the Macro "PreTeXt Personal Mark" to insert appropriate text.

18.8.3 Keyboard Tricks with Tags

Two new keystroke commands are available to aid in text entry. Here is the first. Suppose <hello> is an xml tag. Double click in the "hello" while holding down the option key to select all text between this tag and its associated </hello>. This action understands xml comments and will ignore text inside these comments. This technique also works in reverse; double click on "hello" in </hello> while holding down the option key and TeXShop will find the associated key <hello> and select all text between these tags.

The second keystroke command is called "Close Tag" and is initiated by a TeXShop menu item of the same name. The command has a keyboard shortcut option-command-period for easy entry. This command will search for an unclosed tag like <hello> and close it with </hello> at the cursor spot. The command understands xml comments and ignores text in such comments.

This command is often used by PreTeXt authors, who insert a new tag and then click on the appropriate closing spot and type option-command-period to insert its close. Close Tab works by searching backward until it finds an open tag that has not been closed. Note that "the rules of xml" limit the spots where commands can be legally closed. If a user tries to close a command at an illegal spot, a different closing tag will almost always be entered at that spot. When that happens, back up and think carefully about what you want to do.

For example, consider the text below, where <hello> has no associated closing tag. Notice that the initial <hello> can be completed immediately after it occurs, or after the ending </sage>, or after the following </p>, but not elsewhere. If you try to close it after A.rref(), Close Tag backs up and finds that <input> has not been closed, so it closes that rather than <hello>.

```
<section>
   <hello>
      <sage>
         <input>
         A = matrix(4, 5, srange(20))
         A.rref()
         </input>
         <output>
         [  1  0 -1 -2 -3]
         [  0  1  2  3  4]
         [  0  0  0  0  0]
         [  0  0  0  0  0]
         </output>
      </sage>

      <p>This is extra text.</p>

</section>
```

18.8.4 Command Completion

A new completion dictionary is provided for html, xml, and ptx files, listing phrases likely to occur in a PreTeXt document. When an xml file is opened, TeXShop will automatically switch to this dictionary, and the menu command "Edit Command Completion File" will open this dictionary.

Consult the document **PreTeX with TeXShop** in `~/Library/TeXShop/Engines/Inactive/PreTeXt` for details about using and creating entries in this xml version of the command completion dictionary.

Chapter 19

ConTeXt

Recall that LaTeX is ordinary TeX used with the LaTeX macros written by Leslie Lamport. ConTeXt is TeX used with an entirely different set of macros written by Hans Hagen around 1990. Hagen works for an educational publishing company in the Netherlands, and that company needed to produce books with distinctive and colorful styles rather than the standardized documents produced by LaTeX. As Wikipedia wrote in their ConTeXt article,

```
ConTeXt may be compared and contrasted with LaTeX, but the primary thrust
of the two are rather distinct. ConTeXt from the ground up is a typography
and typesetting system meant to provide users easy and consistent
access to advanced typographical control - important for general-purpose
typesetting tasks. The original vision of LaTeX is to insulate the user from
typographical decisions - a useful approach for submitting e.g. articles
for a scientific journal.
```

As ConTeXt developed, it faced limitations in the underlying TeX engines, and Hagen was also instrumental in a related project to create LuaTeX, a modern version of TeX with full support for Unicode, modern fonts, and precise control over typesetting. Today many LaTeX users typeset with LuaTeX rather than TeX or pdfTeX, and that engine is a standard part of TeX Live.

Both ConTeXt and LuaTeX changed rapidly during the thirty years from 1990 to 2020, and the yearly updates of TeX Live had a hard time catching up. Since ConTeXt is independent of the various style packages and fonts of standard TeX, the ConTeXt support group developed a special distribution containing only ConTeXt, LuaTeX, and the files needed to support them. ConTeXt users typically use this distribution for ConTeXt projects and TeX Live for other TeX projects. The two distributions easily coexist. See the text below

for details.

Nicola Vitacolonna is responsible for the next few developments, first calling some new features in ConTeXt to my attention and then helping me support them in TeXShop. Send thanks to him.

The binaries in the TeX Live distribution of TeX are updated once a year. All other style files, class files, and the like are updated daily. This approach is reasonable because the binaries are quite stable, while the auxiliary files are constantly being updated by thousands of contributors.

ConTeXt is a special case because its author, Hans Hagen, is constantly making improvements. So the copy of ConTeXt in TeX Live is often out of date. Luckily, ConTeXt users have access to a beautiful wiki called the ConTeXt Garden; see `https://wiki.contextgarden.net/Main_Page`. This page has a link titled "Install ConTeXt and start typesetting" which leads to a page with easy ways to install a complete ConTeXt system on MacOS, Windows, GNU/Linux, FreeBSD, and OpenBSD. The MacOS portion has binaries for both Intel and Arm. The distribution contains everything needed to typeset, including a recent copy of LuaTeX, but it is quite small. It can be installed in a user's home directory.

ConTeXt users should certainly use this distribution, and reconfigure TeXShop to point to it. However, this makes life difficult for users who write both ConTeXt and ordinary LaTeX documents. A recent version of TeXShop fixes this problem. In TeXShop Preferences under the "Engine" tab, the "Path Settings" item at the top has a new item named "Alternative Distribution." Set this item to the path of your ConTeXt distribution, which depends on the location you picked to install it. Then add the magic line below to the top of your ConTeXt source files:

```
% !TEX useAlternatePath
```

These ConTeXt documents will be typeset by the ConTeXt Garden distribution, while ordinary LaTeX documents will continue to use TeX Live.

Other users may find this feature useful in special circumstances. The "Alternate Distribution" preference item can point to any TeX distribution and the magic line can be used in any source file. However, the magic line only affects typesetting "engines", and not the built-in calls to TeX, LaTeX, BibTeX, or Make Index. Recall that these engine files are just shell scripts. When such a script is called, it is called with three arguments:

- "$1": the full path to the source file
- "$2": an optional parameter determined by the magic line **!TEX parameter** in the source, or just a space if no such line is given

- "$3": the full path to the binaries in the distribution; this is usually /Library/TeX/texbin but points to the ConTeXt binaries if the line **!TEX useAlternatePath** is in the source

Existing engine files don't use "$3", so they need to be revised if you want to use an alternate path with them.

Putting this all together, here is an engine file which calls the Magic Garden version of ConTeXt:

```
#!/bin/tcsh

setenv PATH "$3"\:$PATH
mtxrun --script context --autogenerate "$1"
```

Jerome Laurens wrote synctex and it is used for all typesetting engines in TeX Live. Laurens adds code to the engines which generate a synctex file. He also provides C code for front end developers to process the synctex file and provide the key information to display a sync location.

But Hans Hagen eventually rewrites everything for ConTeXt, and several years ago he replaced Laurens' code with his own version to generate the synctex file in ConTeXt. This was painful for me because Hagen used Lauren's original design of synctex and ignored a later modern version. So TeXShop had to include two versions of the C code for front end developers, one for all engines except ConTeXt and a second just for ConTeXt. A magic line

```
% !TEX useOldSyncParser
```

was added for ConTeXt users, telling TeXShop to use the original Lauren's code rather than the latest version to interprete the synctex file in ConTeXt.

Recently Hagen also wrote code to interprete the synctex file for ConTeXt users. Now TeXShop has code to call this ConTeXt interpretation code. To use it, ConTeXt uses should replace the previous magic line with

```
% !TEX useConTeXtSyncParser
```

This is a welcome improvement, because now Hans both generates the synctex file for ConTeXt users and also provides the code to interprete the file. Consequently, he is free to make any further changes without requiring changes in the front ends.

In 2023, further changes were made to the ConTeXt distribution in TeX Live and MacTeX and to the ConTeXt distribution available through the ConTeXt Garden. To cope with these changes, a new engine is provided in `~/Library/TeXShop/Engines/Inactive/ConTeXt-2023` called ConTeXt-2023. Use this engine if you are using TeX Live 2023 or

MacTeX-2023, and if you are running macOS Monterey or Ventura or later. If you are using an earlier operating system, read the document in the ConTeXt-2023 to modify this engine appropriately.

How should a ConTeXt user react to these changes. I recommend

- Go to the ConTeXt Garden site and install the latest version of ConTeXt in your home directory

- Add the ConTeXt engine to `~/Library/TeXShop/Engines`. Use the engine ConTeXt2021 if you are using an older version of TeXLive or MacTeX, and the engine ConTeXt-2023 if you are using the 2023 or later version of TeX Live or MacTeX.

- Add the magic lines

  ```
  % !TEX TS-program = ConTeXt-2023
  % !TEX useAlternatePath
  % !TEX useConTeXtSyncParser
  ```

  to the heading of your ConTeXt source files if you are using the ConTeXt Garden distribution. Omit the middle line if you are using the distribution in TeX Live and MacTeX.

- Add the line

  `\setupsynctex[state=start,method=min]`

  to your ConTeXt source file; this line tells ConTeXt to generate the new synctex information.

Chapter 20

Using an External Editor

20.1   Basics

TeXShop can be configured for an external editor in several ways. Some features below will be useful and others not depending on how your editor works. In the future, your editor may come with configuration instructions.

There is a menu item named "Open for Preview..." When this item is chosen, you will be asked to select a .tex source file. But only the associated pdf preview window opens. If the source has not yet been typeset or if the pdf is out of date, TeXShop will typeset the file as it is opened. You can open your source file in any editor. When it is time to typeset, save the changes in your editor, switch to the preview window, and typeset. In this mode, TeXShop never opens or modifies the source file. It simply passes this file to the Unix TeX or LaTeX process.

The preview window can be configured to contain a typesetting button. If such a button is not there, use "Customize Toolbar..." in the Windows menu to add one.

For users who prefer external editors most of the time, there is a preference item called "Configure for External Editor." When this preference is chosen, the "Open" and "Open Recent..." menus open tex source files in the above manner for editing with an external editor. Moreover, the "Open for Preview..." menu becomes "Open for Editing..." and opens the source file in TeXShop's internal editor for those rare occasions when the internal editor is desired. The "Open" and "Open Recent..." menus can also open jpg, tiff, ps, pdf, dvi, log, and other files; selecting the new preference does not change this behavior.

Some editors are able to call Unix typesetting commands directly. A new preference item, "Automatic Preview Update", has been added to improve your experience with these editors. When this item is active and a .tex file is opened using "Open for Preview", the pdf

preview display automatically updates whenever the pdf file is rewritten. Thus you can typeset with an external editor and the preview window will automatically show changes. The preference also applies to .pdf files opened in TeXShop, and if TeXShop is configured to use an external editor, the preference applies to .tex files opened with "Open".

The following applescript commands have been added to TeXShop so editors can call TeXShop directly:

```
typesetinteractive
texinteractive
latexinteractive
contextinteractive
bibtexinteractive
makeindexinteractive
metapostinteractive
taskdone
refreshpdf
open_for_externaleditor
```

The first seven commands activate TeXShop typesetting commands. Applescript will return immediately after these calls without waiting for typesetting to complete. The "taskdone" call can be used to test completion; it returns NO while typesetting is being done, and YES when it is finished. Each of these calls refreshes the preview window at the end of typesetting. If your editor calls TeXShop to typeset, the "Automatic Preview Update" preference can be turned off. If the external editor modifies the pdf output directly, it can call "refreshpdf" to refresh the pdf display. This is only needed if "Automatic Preview Update" is off. Finally "open_for_externaleditor" opens a tex file by calling "Open for Preview".

20.2 Preliminary Version: Using Sync with an External Editor

Jesús Soto uses TeXShop with an external editor, TextMate, and requested that SyncTeX in TeXShop be given the ability to sync from the Preview window to the corresponding source in TextMate. I absolutely, positively, refused to honor this request. But I couldn't help thinking about it. In the end I relented.

TextMate installs a command line program in /usr/local/bin which is able to control some of the features of the editor. For instance, the command

```
/usr/local/bin/mate --line 50 /Users/koch/Syllabus.tex
```

opens Syllabus.tex in TextMate (if not already open) and highlights line number 50. This

makes it easy to support syncTeX from TeXShop to TextMate because the standard TeXShop routine to implement syncTeX can still be used until the last moment when a line in the Source Window would be highlighted; this final step can be replaced by a call to "mate".

Therefore TeXShop can sync from the Preview window to TextMate in the standard way: click on a spot in the pdf display while holding down the command key. As a bonus, Goto Error works from the Console window if the user is using TeXShop to typeset. Click this button to be taken to the first error, and click again and again to cycle through the first few errors. Of course TeXShop must be in "external editor" mode to use these features; It is also necessary to activate the features with a hidden preference:

```
defaults write TeXShop TextMateSync YES
```

If you use a different external editor and want to implement the feature, you must first turn it on using a hidden preference:

```
defaults write TeXShop OtherEditorSync YES
```

This default deactivates the TextMateSync preference mentioned earlier. Instead whenever TeXShop wants to "sync from pdf to editor" or "Goto Error", it calls a shell script in /usr/local/bin named "othereditor". When it calls this script, it sends two parameters to the call: $1 contains the line number (as a string), and $2 contains a full path to the desired tex source file. The othereditor script then calls the external editor using its version of the "mate" program.

But there is no "othereditor" script in /usr/local/bin. The user needs to create this script. The script must be named "othereditor" with no extension, and have its execute bit turned on. The script should call whatever binary is used by the desired editor to open the file $2 if it is not already open, and jump to line $1.

For example here is the complete "othereditor" script file for TextMate:

```
#!/bin/tcsh
set path = ($path /Library/TeX/texbin /usr/texbin /usr/local/bin)
/usr/local/bin/mate --line $1 $2
```

A final word of warning. Since this code is preliminary, I haven't worried about paths or file names containing spaces. Avoid these spaces for now.

20.3 More on External Sync

In the previous section, we activated "Goto Error" and "Sync from Preview to Editor" for TextMate. We did that by calling an external script, which had to be located in /usr/local/bin, had to have its execute bit set, and had to be named "othereditor". This

script received two inputs, $1 and $2. The first variable contained a line number of the source window and the second variable contained a full path to the file to be displayed by the editor. When called in this way, the script should cause the editor to open the source file if it is not already open, scroll to the line number, and mark that line in some fashion. Activating this feature also required that TeXShop run in external editor mode, and that the hidden preference item "OtherEditorSync" be set to YES.

The actual script would be written by the authors of the external editor, because only they know how to ask their editor to perform this action. The script might contains an applescript command, or perhaps activate a third party program which communicates with the external editor. In a couple of cases, the script is already known because these editors communicate with other front ends. Below are scripts for TextMate, and for BBEdit. Thanks to Max Horn for these.

TextMate:

```
#!/bin/sh
/usr/local/bin/mate --line "$1" "$2"
```

BBEdit:

```
 #!/bin/sh
/usr/local/bin/bbedit "$2:$1"
```

Both of these scripts ultimately call binary programs in /usr/local/bin provided by the editors: mate for TextMate and bbedit for BBEdit. So they require that the editors install their associated binaries. In TextMate, for instance, there is a preference item which causes TextMate to install the mate program.

TeXShop 4.25 also allows syncing the other direction, from the editor to TeXShop. This time the editor must call TeXShop, giving it three variables, $1, $2, and $3. The first is the line number in the source file which the user clicked. The second is the character position in the line where the user clicked. Both are strings representing integers, so values might be 625 and 35. The final variable is a full path to the source file being displayed by the editor. This could be **/Users/koch/deRham/Chapter2.tex**.

For this to work, TeXShop must be in External Editor mode and the hidden preference OtherEditorSync must be set to YES.

Notice that TeXShop will then open the synctex file and do all processing to finish syncing. Ultimately it will scroll to the corresponding spot in the pdf Preview Window, and mark that spot.

The Editor can call TeXShop to perform this operation in one of two ways. First, there is a shell script which makes the call. Here is that script. Copy this text, say in TeXShop, and save it to a file named "ExternalSync" with no extension, and with its execute bit set.

I recommend putting the file in /usr/local/bin. The location and name of the file are not important, provided the editor knows which name and location the user picked.

```
    #!/bin/bash

MyShellVar=$1
MyShellVas=$2
MyShellVat=$3

osascript  &lt&ltEOD

tell application "TeXShop"

activate
set the front_document to the front document
set MyAppVar to $MyShellVar
set MyAppVas to $MyShellVas
set MyAppVat to "$MyShellVat"

tell front_document
sync_preview_line theLine MyAppVar
sync_preview_index theIndex MyAppVas
sync_preview_name theName MyAppVat
return 0
end tell

    end tell
    EOD
```

Notice that this shell script receives three variables, and passes them on to an applescript which then runs and connects with TeXShop. Some editors may be able to run applescript commands directly. They can ignore the shell script and copy and run the applescript command.

20.4 Irrelevant Comments

The following discussion is a sign of my ignorance rather than a sign of my knowledge. You should skip it. I keep it in case someone wants to enlighten me.

When I was an undergraduate, my teacher claimed that Niels Bohr would say after a conversation "We clearly don't understand this theory; let's write a paper about it." In that spirit, let me write about communication between independent programs on the Mac.

In Unix, each program runs as a separate process with an independent address space, so direct communication is not easy and usually involves the Unix kernel.

One form of communication occurs when a program starts another program. It can then provide that second program with an unlimited number of parameters. GUI programs on the Mac are actually standard Unix programs in disguise, so they can be opened from the Terminal and additional parameters can be prescribed at that time. TeXShop doesn't use that ability, but it certainly calls programs like TeX, MakeIndex, XeTeX, etc., providing each with additional parameters. Some editors can also typeset; they call other programs in this way.

Shell scripts are a special case of this. When such a script first starts, it can retrieve the parameters used to call it, as $1, $2, $3, and so forth. Cocoa has a special class called NSTask for starting other programs, so communicating when new programs start is easy to implement on the Mac, and in particular communicating by starting shell scripts is easy.

Applescript is another form of communication between independent programs. Indeed, a major use of applescript is to make programs "scriptable", so they can be controlled by other pieces of code. Deep down inside, applescript implements special methods to make this communication between independent tasks possible.

There are, however, two problems with applescript. The first is that it can be insecure, and Apple has started to address this problem by allowing programs to refuse to accept applescript communication. It is reasonable to fear future additional restrictions. The second problem with Applescript may be my fault. I don't grok the language, and it doesn't grok me. So I waste hours and hours trying to make Applescript do something trivial, like pass more than one variable in a procedure.

Notice that TextMate and BBEdit have another method of communicating with external programs. Each has an independent program in /usr/local/bin, which can ask the main program to perform certain tasks. I looked at the TextMate code. It called low level Unix socket commands; I admired it in the same way that I admired a 21 year old pianist who performed Liszt's Hungarian Rhapsody Number 6: fantastic, but something I could never do.

I once went to a NeXT developer conference, maybe the only one they held. At that conference, NeXT introduced "Distributed Objects", an easy way for independent programs to communicate on a NeXT. First, special "Connection Objects" established a connection between two programs. After that, one computer could run objects on the second machine and receive results back. The two programs could be running on the same machine, or different machines on a single ethernet network, or machines half way across the world. In the conference, NeXT showed slides containing the code necessary to establish such a connection. Each slide contained eight or ten lines of code.

For a time, distributed objects were also in Cocoa. I looked up the code recently. Every single routine in the API was deprecated.

However, there is a replacement, called XPC Services. I looked briefly at the code. It may be that this is the correct way to provide interconnection, particularly if AppleTalk has further restrictions. But someone else will have to take the time to decipher how it works.

20.5   More Recent Interaction Methods

Nicola Vitacolonna is responsible for the remaining features for external editors, to be explained below.

To support users who prefer an external editor, a new folder has been added to `~/Library/TeXShop` named **ExternalEditorScripts**. This folder will be updated with each new version of TeXShop, just as `~/Library/TeXShop/Engines/Inactive` is updated now. The new folder contains scripts contributed by users for various editors, each in a folder with the editor's name. TeXShop contains scripts by Michael Sharpe for BBEdit, and scripts by Nicola Vitacolonna for MacVim.

TeXShop contains additional commands making it easy to interact with external editors. These commands are documented below, but the text is quite technical. In practice, this information can be used to write scripts which automate the interaction, and these scripts will then end up in the ExternalEditorScripts folder. In the end, users can interact with TeXShop without understanding the details below.

If an external editor is used for ConTeXt work, TeXShop does not open the source file and thus cannot see the magic lines introduced in the previous chapter. So a method is needed for external editors to pass this type of information to TeXShop. We provide new applescript commands for this purpose.

Eight new applescript commands have been added to TeXShop:

```
StandardBinPath
AlternateBinPath
SyncRegular
SyncConTeXt
SyncWithNoEditor
SyncWithTextMate
SyncWithOtherEditor
SyncRootName
```

These commands return nothing and have no parameters. The commands apply to the front window in TeXShop. The commands are only needed when an external editor is used;

otherwise use the magic lines explained earlier.

The first line says that typesetting commands are in the standard location, `~/Library/TeX/texbin`. The second command says that the Alternate Binary Path should be used to find typesetting commands. By default, the first statement is active.

The third command says that sync commands for the active window should use Jerome Laurens' code; the fourth command says that sync commands for the active window should use the new ConTeXt sync. By default, SyncRegular is active.

Earlier, we introduced two two hidden Preference items]:

```
defaults write TeXShop TextMateSync YES
defaults write TeXShop OtherEditorSync YES
```

These items still work, but are no longer recommended. Instead, use the next three applescript commands. The command "SyncWithNoEditor" tells TeXShop that the editor does not communicate at all. The command "SyncWithTextMate" says that TextMate is the external editor and TeXShop should use the commands built into this program. The command "SyncWithOtherEditor" says that some other editor is being used and TeXShop should call a shell script to communicate with that editor, as described below and also in the previous sections.

The "SyncRootName" command is rarely used and will be explained later.

To clarify the use of these commands, the following applescript command tells TeXShop to typeset using the version of ConTeXt installed in the alternate location, to sync using the new ConTeXt methods, and to interact using the "other editors" method:

```
tell application "TeXShop"
     set the front_document to the front document

     tell front_document
          AlternateBinPath
          SyncConTeXt
          SyncWithOtherEditor
          return 0
     end tell
end tell
```

This script can be made into a TeXShop applescript macro. Recall that TeXShop has a menu item to call Macros, so the Macro can be called even if TeXShop is in external editor mode. Call it just once when you work on a ConTeXt source, and after that the TeXShop typeset command will use the alternate ConTeXt distribution, will sync using the new ConTeXt sync methods, and will communicate using the "other editors" protocol.

If your editor supports applescript and allows users to add additional commands, you can build this command into the editor rather than creating a macro. If your editor does not support applescript but can call shellscripts, you can easily embed the applescript in a shell script:

```
#!/bin/bash

osascript  &lt&ltEOD

tell application "TeXShop"
   set the front_document to the front document
   tell front_document
      AlternateBinPath
      SyncConTeXt
      SyncWithOtherEditor
      return 0
   end tell
end tell
EOD
```

From now on, we concentrate on the actual communication between TeXShop and the external editor.

Your editor may already support sync from preview to source. When TeXShop syncs in that direction, it finds the location of the click in the preview window and processes this information using either Jerome Lauren's original sync code or the new ConTeXt Sync code. It ultimately produces two pieces of information, the line number of the corresponding text in the source file, and a full path to the source file. This full path is required when source documents include other files, so the correct file will be opened and displayed. The remaining issue is communicating those two parameters, "$1" = line number and "$2" = source path, to the editor, so it will open "$2" if it is not already open and then select line "$1".

As it happens, TextMate contains a small file named "mate" which it can install in /usr/local/bin. If mate is called with two string parameters "$1" and "$2", it opens the file with path "$2" if it is not already open, and then selects line "$1". If the command SyncWithTextMate has been called, then TeXShop will call this "mate" program and configuration is complete. Incidentally, this call is also used when a TeXShop user selects "Go To Error".

If a different external editor is being used and the command SyncWithOtherEditor has been called, then TeXShop behaves in more or less the same way. Instead of calling "mate" in /usr/local/bin, it calls a binary named "othereditor" in /usr/local/bin. It calls this with

the same two parameters "$1" and "$2".

The catch is that someone else has to write this "othereditor" file, which must have its execute bit set. However, it may be trivial to write this file for widely used editors, provided they have provided the necessary hooks. Below are two such scripts, one for TextMate and one for BBEdit. The TextMate script is provided to show that SyncWithOtherEditor also works for it; the BBEdit script shows that another famous editor supports the required communication.

For TextMate:

```
#!/bin/sh
/usr/local/bin/mate --line "$1" "$2"
```

For BBEdit:

```
#!/bin/sh
/usr/local/bin/bbedit "$2:$1"
```

If you use some other editor and discover an easy way to write an "othereditor" file for it, please communicate that discovery to the TeX on OS X mailing list or in some other way.

Finally, we must implement sync the other way from source file to preview. In that case, it is up to the editor to recognize a click in the source while the command key is down, and to supply TeXShop with three pieces of information: "$1" = the line number clicked in the source file, "$2" = the character index of the click in that line, and "$3" a full path to the source file. If the source file clicked was just the original file which TeXShop opened in external editor mode, then it already knows "$3". But if the user's projects involve included or input files, then the third parameter is essential to tell TeXShop where to find the sync information.

The author of your editor must supply the code to convert a click into these three pieces of information. The editor should then call TeXShop, giving it the three parameters "$1", "$2", "$3". TeXShop expects the information to come as an applescript command. Below is that command. Of course the editor has to supply the strings "linenumber", "indexnumber", and "full path to source file":

```
    tell application "TeXShop"
        activate
        set the front_document to the front document

        tell front_document
            sync_preview_line theLine "linenumber"
            sync_preview_line theIndex "indexnumber"
            sync_preview_line theName "full path to source file"
            return 0
        end tell
    end tell
```

If your editor prefers to call a shell script, it is easy to embed the applescript in such a script:

```
    #!/bin/bash

    MyShellVar=$1
    MyShellVas=$2
    MyShellVat=$3

    osascript  &lt&ltEOD

    tell application "TeXShop"

        activate
        set the front_document to the front document
        set MyAppVar to $MyShellVar
        set MyAppVas to $MyShellVas
        set MyAppVat to "$MyShellVat"

        tell front_document
            sync_preview_line theLine MyAppVar
            sync_preview_index theIndex MyAppVas
            sync_preview_name theName MyAppVat
            return 0
        end tell

    end tell
    EOD
```

]Note that before using the above call, you must tell TeXShop the information it cannot

find from magic lines:

- The editor, either SyncWithTextMate or SyncWithOtherEditor

- The sync method, either SyncRegular or SyncConTeXt

- The binary location, either StandardBinPath or AlternateBinPath

Before asking the author of your editor to write the required code for syncing from editor to preview window, you may wish to test that it works. To simplify your life, you might suppose that all of the source code is in the main file. In that case, it isn't necessary to pass the full path to the source file, and so the last line of the applescript command which reads

```
sync\_preview\_name theName "full path to source file"
```

can be replaced with an applescript command with no parameter:

```
SyncRootName
```

Chapter 21

Themes

TeXShop uses customizable colors for syntax coloring, for various window backgrounds, and for other purposes. In the original version of TeXShop, users could modify these colors with a fairly baroque set of preference items. Then in macOS Mojave, Apple introduced "Lite Mode" and "Dark Mode" and an entirely new set of colors was required. At that point the old color preferences were ripped out of TeXShop and replaced by the Themes tab in Preferences. While previously many colors could only be changed using obscure hidden Preference settings, now all color choices are available in the Themes tab.

The Themes portion of Preferences is shown on the next page. On the right are all colors currently set by TeXShop. Some items have an obvious meaning and others will be explained shortly. A full set of such choices is called a "Theme". TeXShop allows users to create as many themes as they like. These themes are listed in three pulldown menus on the left: Lite Mode Theme, Dark Mode Them, Theme to Edit. The first menu sets the theme used on all systems below Mojave, and the theme used in Lite Mode on Mojave and above. The second menu sets the theme used in Dark Mode on Mojave and above. The final menu sets the theme which Preferences is currently editing.

TeXShop is shipped with several themes, including "LiteTheme" and "DarkTheme". These are the default themes for Lite Mode and Dark Mode. As explained later, there is a way for users to rename or remove Themes known to TeXShop. But TeXShop will always replace "LiteTheme" and "DarkTheme" and use them if other required themes are missing.

Gary Gray contributed GLG-Dark for Dark mode. Gray then tweaked this theme, and ended up with a dark theme that was was so my better than mine that I ended up using it as the default and renaming it DarkTheme. So Gray lost credit, but gained users. Thanks.

There is no distinction between themes for Lite Mode and themes for Dark Mode. Thus

nothing prevents a user from setting both Lite Mode and Dark Mode to the same theme.

After editing a theme, push "Cancel" or "OK" to end a preference session. If "Cancel" is pressed, the edited colors will not be saved and the Lite Mode and Dark Mode themes will return to choices before opening the Preference Pane. If "OK" is pressed, the edited colors will be saved and Lite Mode and Dark Mode themes will change to their new values.
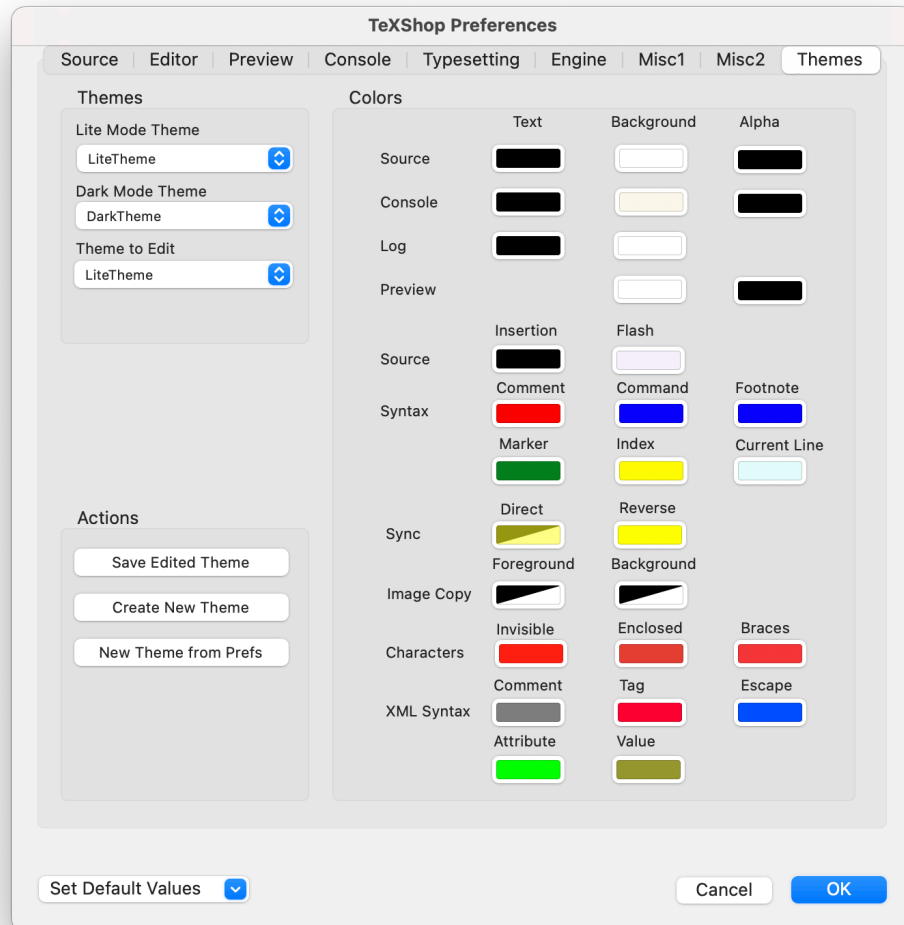

Figure 21.1: Theme Preference Items

Some users may want to edit several different themes during a session. When these users are finished editing their first theme, they should press "Save Edited Theme." This will

save the changes for that theme *permanently*, even if the entire session is ended using the "Cancel" button. Repeat the process for other themes.

To create a new theme, first change "Theme to Edit" to obtain reasonable starting colors for your new theme. Then push "Create New Theme" and fill in the resulting dialog with a title for this theme. Do not use spaces in this title. The new theme will become the "Theme to Edit" and you can begin changing colors.

One final item in the left column must be explained, but it is essentially obsolete. When the theme panel was first introduced, some users had spent time using the original color preference system to revise the default colors, only to have that system ripped out of the program. Their color preference choices, however, still remained in the Preferences data base. The button "New Theme from Prefs" created a new color theme using these old preference choices. There is just one other use for this button which someone might conceivably still use. When the old color preference system existed, some people on the internet developed color themes for TeXShop and made them available as shell scripts which reset various TeXShop color setting preferences. These shell scripts still work, but they no longer affect the appearance of TeXShop. After running such a script, you can use "New Theme from Prefs" to convert the old scripted theme to a regular new Theme.

Recall that various TeXShop items which users can customize are set in `~/Library/TeXShop`. This folder has various subfolders. There is a new folder in `~/Library/TeXShop` named "Themes". This folder contains very small ".plist" files describing the various Themes in TeXShop. If you create a theme you like, give it to others by putting its plist file on the Internet. To install a new theme of this kind, just drop its plist form in the Themes folder.

You can also remove Themes you no longer use by removing their plist files from the Themes folder. Avoid removing themes being used for Lite Mode or Dark Mode (although TeXShop should react gracefully when it runs into this situation). As explained earlier, the themes LiteTheme and DarkTheme will be recreated if they are removed.

When a theme is selected for editing, TeXShop colors will temporarily be reset to those colors. Revising colors is then interactive; as soon as colors change in Preferences, they will also change in TeXShop's Source and Preview windows.

Finally we explain the actual color choices. At the top of the Theme Panel, the Foreground and Background colors can be set for the Source, Console, and Log Windows, and the Background color can be set for the Preview Window. For each of these except the Log window, transparency can also be set. In color theory, alpha value sets transparency. Transparency settings bring up a full Color Well, but the colors of these items are ignored and only the alpha values of the choices matter.

Why can we set the Background Color of the Preview Window, but not its Foreground

Color? That is because pdf files created by TeX have an invisible background color. When such a file is printed, the result is black characters on the actual paper. If the pdf had a background color, then the printer might print a grayish wash over the entire page, which is clearly undesirable. So TeXShop can independently set the background of the Preview Window, and it will appear except when covered by actual letters. Note that when such a document is printed, the background color setting of Preview will be ignored because this information is not in the pdf file.

Illustrations are a different matter. Most illustrations set the background color, and thus the background color of Preview will not appear behind illustrations.

However, the text color is set by TeX itself, and cannot be reset by TeXShop without rewriting the pdf file itself. The only way to change the color of the text is to set it in the LaTeX source for the document.

After these items, we see colors of the source insertion cursor and its flash. The Insertion cursor is the flashing vertical bar that is seen as we enter text. Flash refers to a feature that many users will not turn on. If a closing parenthesis like) or } or] is typed and it does not match an existing open parenthesis, the edit window can be told to flash its entire background briefly as a warning. Note that the default color of this flash is subtle.

Next come six colors used for syntax coloring of TeX source. In the default scheme, comments are red, TeX commands starting with a backslash are dark blue, footnotes are blue, markers, the parentheses setting off arguments of commands, are green, and index settings are yellow.

It is possible to ask that the background of the active line be colored, and "Current Line" sets that color. After this come the direct and reverse colors used to indicate a SyncTeX match.

The colors "Invisible Chars, Enclosed Chars, Braces" are used for some features introduced by Yusuke Terada; see the menu item "Show Invisible Characters" and the item "Parens Targets & Highlight Color" in the Source Tab of Preferences, and the items "Show Invisible Characters" and "Parens Matching Settings" in the Editor Tab of Preferences. The items "Image Copy Foreground, Background" refer to rather obscure colors, used when copying a portion of the pdf file in unusual formats.rather than as pdf, jpeg, tiff, or png.

An item in the Source menu asks the editor to "Show Invisible Characters". The Invisible color is the color used for these characters.

The Enclosed and Braces characters are used for the items set in the top right of Preferences under the Editor tab.

Finally the XML Syntax colors are colors used when TeXShop syntax colors html, xml, and ptr files, as described in the chapter on Html, TeX4ht, and PreTeXt.

To be completely honest, one color is not set in the Themes panel. It is the color of the
optional block cursor, set in the Misc1 tab of Preferences. This will be discussed in a later
chapter of this Manual.

Chapter 22

Additional TeXShop Features

22.1 Experiments

This feature was requested by Wendy McKay.

The item "Experiment..." in the Edit menu is available when the source window is active. The item allows users to experiment with short, but complicated, fragments of TeX before copying the source into the main document.

When the item is chosen, a panel appears. Type a TeX fragment into the panel, say

```
$$\sqrt{x^2 + y^2}$$
```

Push the Typeset button at the bottom of the panel, and a second panel appears showing the result of typesetting the fragment. The fragment can contain anything: a displayed formula, ordinary text, several pages of mixed material. To typeset, TeXShop creates a new source file with the header of the current document up until "begindocument", the new fragment, and a final "enddocument" This also works in a project with a root file. In that case the contents of the root file up until "begindocument" are used.

Both panels have close buttons. The ESCAPE key will also close panels when they are active. Although the two panels do not have resize buttons, they can both be resized. TeXShop will remember the new sizes and locations and use them the next time "Experiment.." is selected. The font in the source panel will be the default TeXShop source font. The keyboard shortcuts "command +" and "command -" work in the source panel to enlarge the text if desired. Key bindings and command completion are available in the source panel, but with one caveat. Command completion uses the tab key in the panel even if it uses the escape key for regular source, since the escape key in a panel closes the panel.

The "Experiment..." feature requires a latex-like engine. It will not work with ordinary plain tex. The source panel's Typeset button looks at the main source window's toolbar to determine a typesetting engine., and also uses the the magic comment line mechanism if available at the top of the source window. If "Plain TeX" or "Context" is selected, nothing happens. If "bibtex" or "make index" are chosen, pdflatex is used. Obviously "pdflatex, xelatex, and lualatex" can be used. The panel will try to use any user-defined engine selected, but some such engines may fail if they don't expect latex-like code or don't output pdf.

The preview panel understands mouse scroll commands and trackpad gesture commands to scroll and resize. If you close a panel during work and later reopen it, the contents will be remembered. But the contents are lost when quitting TeXShop. It is assumed that panels will be used for short fragments of work; when the user is satisfied, they will transfer the source to the main document using copy and paste. Panel contents are not auto-saved and cannot be manually saved except via the copy mechanism.

Each document has its own source and preview panels, so if you have multiple documents open, you could also have multiple source and preview panels open, leading to a confusing mess. I expect users to exercise common sense and only experiment with one fragment at a time. One way to avoid confusion would be to hide the panels when a document becomes inactive. I didn't want to do that because a user constructing a complicated example might want to temporarily open a second document and copy source from that document into the panel as a starting point.

After this item was introduced, it was mentioned on the web site `tex.stackexchange.com`, and Denis Bitouzé suggested an improvement. Following his suggestion, if a selection of source is made first and then the menu is chosen, the selection is automatically copied into the experiment window. If no selection is made and instead the cursor is simply positioned by clicking at a point, then the Experiment window opens with its previous contents. Thus if a user carefully edits an equation, closes the experiment window, and then decides on a final change, the contents can be brought back for another edit.

Typesetting for "Experiment" uses the Preference settings for pdflatex and simpdftex latex. If the user's pulldown menu is set to latex, the three choices of 1) pdflatex, 2) tex + dvi, 3) personal script are obeyed. This should make the command work for TeX in Japan.

22.2 Splitting Windows

The toolbars for the source and preview window each contain an icon containing two rectangles on top of each other. This item "splits the window" into two independent parts. This makes it possible to examine one portion of the document while writing a separate portion. The Window Menu contains an item "Split Window" which does the same thing

and has a keyboard shortcut.

If the option key is down when a window is split, it splits vertically rather than horizontally.

22.3   Tags Toolbar Item and Menu; Labels Item

The toolbar for the Source Window has an item labeled Tags. This menu displays a scrollable list of all chapters, sections, and subsections in the document. Select an item to be taken to that spot in the source file.

It is possible to add your own tag to a location. Include at line at that location of the form

```
%: tag_name
```

where "tag_name" can be any sequence of words.

The menu takes at least the following commands (and more if an xml file is being used):

```
part
chapter
section
subsection
subsubsection
paragraph
subparagraph
macro
environment

frame
slide
wideslide
notes
```

The item "Tags Menu in Menubar" under the Editor tab in TeXShop Preferences causes a duplicate of the Tags menu to appear in the menubar.

22.4   Latex Panel, Matrix Panel

The Window menu has two items labeled "LaTeX Panel" and "Matrix Panel". Each causes an appropriate panel to open.

The LaTeX Panel floats above regular source windows in TeXShop and lists a large number of mathematical symbols. Activate a source window and click to position the Insertion Cursor. Then click on a symbol in the panel and the corresponding LaTeX code will be inserted at the indicated position in the source.
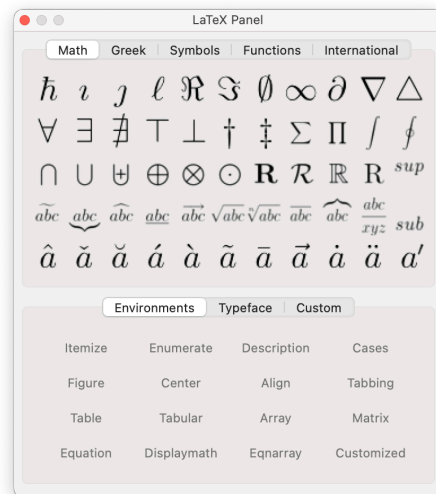
Figure 22.1: Latex Panel

The Matrix Panel is similar, but more complicated. It creates matrices and tables. The number of rows and columns can be set. Experiment with other entries to see how they work.
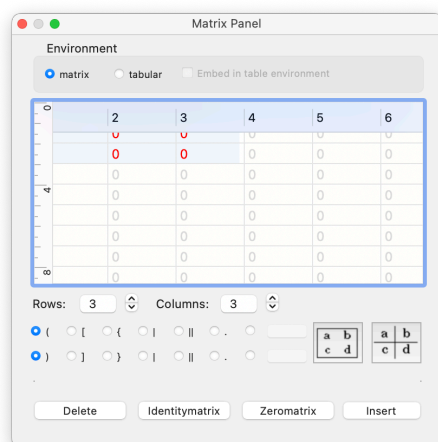
Figure 22.2: Matrix Panel

The Labels item on the toolbar works just like the Tags item, except that it lists labels in the Source Code.

22.5 Unicode Panel

The TeXShop Editor is fully Unicode capable, so unusual scripts can be entered and typeset. Sometimes it is useful to know the particular Unicode characters being typed, and for that purpose Terada Yusuke provided a Unicode Panel. Select a portion of text, and then select "Character Info.." in the Edit Menu. The next page shows this panel examining a piece of Arabic text in a source file to be typeset with XeLaTeX.
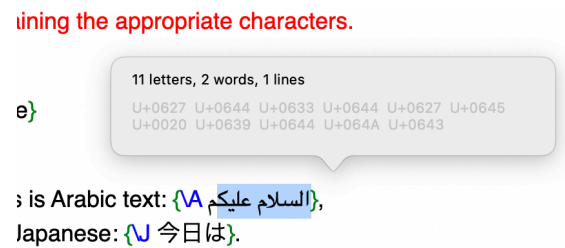
Figure 22.3: Arabic Unicode

22.6 Show Invisible

The Source menu has an item labeled "Show Invisible Characters". The specific invisible characters shown is determined by a setting in the Editor tab of TeXShop Preferences. Below is a short section of source with this item applied to it.
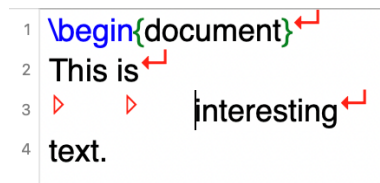
Figure 22.4: Invisible Characters

This particular example has three line feeds and two tab characters.

22.7 A Magic Line for Beamer and Other Slide Packages

Users often write mathematical slides using LaTeX. I highly recommend Beamer for that purpose. It is easy to use and produces beautiful slides.

I usually recommend the MultiPage Display Format for the Preview window, in which the pdf document can be scrolled from beginning to end. But when slides are produced and shown, it is preferable to use the Single Page mode and page down from slide to slide. The magic comment line

```
% !TEX pdfSinglePage
```

causes the document to display in Single Page Mode.

22.8   Counting Words in a Document

The menu item "Statistics" in the Edit menu counts the number of words, lines, and characters in the currently active source file. This is roughly the number of such items in the typeset document because the software is designed not to count LaTeX command words. The statistics panel calls "detex", a command line program in TeX Live, to do the actual counting.

22.9   Useful Items in the TeXShop Menu

Check for Updates

TeXShop uses a third party open source framework called *Sparkle* to handle updates. This item indicates if an update is available. If so, it lists the features of the latest update, and offers to install it. If you accept, the program is updated and restarted, returning to its state just before the update happened. Sparkle is used by many other TeX related GUI programs, including BibDesk, LaTeXiT, and TeX Live Utility.

Open ~/Library/TeXShop

As mentioned often in this manual, important configuration files are in this folder. Since `~/Library` is hidden by Apple, it can be difficult to access the location. Using this item makes the task easy.

Select TeXShop as Default TeX Editor

TeXShop contains a file named Info.plist which the Finder examines to discover features of the program it needs to know. One of these features is a list of filetypes that TeXShop can open and edit. The most important of these are files are TeX source files with extension ".tex".

Unfortunately, many other programs can also edit these files. The Finder more or less randomly picks a program to open files with extension ".tex" when they are double clicked. It is not unusual to install another program and discover that such files no longer automatically open in TeXShop.

The standard solution is to select a ".tex" file in the Finder and then select the menu item Get Info. A dialog opens with many pieces of informaion, and one is a pull down menu listing the program that will open the file if it is double clicked. Choose TeXShop in this menu. Below the menu is an item reading "Use this application to open all documents like this one." Select that item.

However, there is a much easier way to tell that Finder that ".tex" files should open in TeXShop. Just select the menu "Select TeXShop as Default TeX Editor".

Chapter 23

Other Miscellaneous Features

23.1 Items in the TeXShop Help Menu

The TeXShop Help menu contains several items, including a number of pdf documents. Note that the Print command works with these items, so it is easy to get a hard copy by just printing the document. The print dialog has a command which creates a new pdf file instead of printing, so it is also easy to produce an independent pdf document from any of these pdf items.

23.2 Help with Style and Class Files

The TeXShop Help Menu contains three items which provide information about TeX, LaTeX, and other items in the TeX Live Distribution:

TeX Live Documentation

The TeX Live distribution comes with a massive amount of documentation. This menu item opens a page of links to various important items in that documentation.

Show Help for Package ...

Most binaries and packages in TeX Live have associated documentation. Selecting this item opens a small dialog in which you can type the name of a binary or package. Usually this item is in small letters. TeXShop then calls a program in TeX Live to open the documentation in appropriate programs, often in Preview or Safari. Try this by searching for documentation for pdftex, xetex, beamer, hyperref, and other interesting software.

The search tool which this item calls can be set in TeXShop Preferences. By default it is

```
kpsetool -w -n latex tex
```

Open Style File ...

This menu command opens a small dialog in which you can type the name of a style file or class file. TeXShop will then open the file itself.

23.3   Opening Other Files

]TeXShop can open most files for editing. This facility has been provided so users can read TeX-related files with extensions ".log", ".aux", etc.

TeXShop can open files with extensions .ins, .dts, .sty, .cls, .mf, .def, .fd, .fdd, .ltx, .clo, .ctx, .bst, .bib, .engine, .htx, .sk, .dtx, .dry, .dn, .abc, .ly, .texi, .Rnw, .asy, .akt, .bbx, .cbx, .lbx, .mp, and others. It can also open any text file.

Files with extensions .jpg, .tif, .eps, or .pdf are opened as graphic files.

TeXShop can also open .dvi and .ps files. In these cases, it converts the file to pdf and displays this pdf file.

23.4   Arabic, Hebrew, Persian

These languages are written from right to left.

There is an item under the Editor tab in TeXShop Preferences called *Arabic, Hebrew, Persian.* When this item is checked, any line of source text which contains a character from one of these languages in the first three characters will be right justified, while other lines will be left justified.

Consequently lines starting with a TeX command in English will be left justified, even if later parameters are written in one of the languages, but source lines in any of these three languages will be right justified.

I was once shown the source for a college physics textbook in Hebrew. All along the left margin there were TeX formatting commands in English. But all along the right margin there were sentences in Hebrew.

Incidentally, there is a hidden preference setting which can be set by typing the following command in Terminal:

```
defaults write TeXShop RightJustifyIfAnyKey YES
```

If this preference is NO, then the previous behavior will hold. If it is YES, then any line with any character at all in one of these languages will be right justified.

23.5 Localizations

TeXShop has been localized for English, French, German, Italian, Spanish, Portuguese as used in Brazil, Chinese, Japanese, and Korean, Three other languages were briefly supported, Dutch, Portuguese for Portugal, and Romanian. The program can be run in these three languages, but many commands will be in English.

23.6 Redefining Menu Shortcut Keystrokes

Many TeXShop menu items have keyboard shortcuts. Eventually these shortcuts became a nightmare as users requested more and no more shortcuts were available. Some shortcuts are really essential, like command-T for typesetting and command-+ and command– for changing the size of fonts. Others are used by many but not all users, and some are used by only a few.

Nowadays I ignore shortcut requests, letting users modify the shortcuts themselves if desired. There are two ways to do this.

One method is provided by Apple in System Preferences under the Keyboard Module. The details which follow apply to Ventura and later, but similar methods are available for earlier systems. Select the Keyboard module and push the item *Keyboard Shortcuts...* On the left side, select *App Shortcuts*. In the resulting dialog, push the plus key. A new dialog appears. On the top line in the pull-down menu, select TeXShop. TeXShop may not be in the resulting list; in that case select *Other* at the end of the menu and then choose TeXShop. On the second line, type the exact name of the menu command you want to work with. On the final line, type the desired shortcut. Press *Cancel* or *Done* to end the session.

Before Apple created this Preference Item, TeXShop provided a way for users to modify its shortcuts. This method still exists, but it is seldom used. In a pinch, you can try it.

To redefine keyboard menu shortcuts used by TeXShop, open the file `~/Library/TeXShop/Menus/KeyEquivalents.plist` with TeXShop, read the comments at the top of the file, and edit the file to redefine selected menu shortcuts. Be sure to edit and save in UTF-8 format if you use Unicode characters.

23.7 The Block Insertion Cursor

The macOS cursor can take many shapes: an arrow for pointing, a hand for scrolling in multiple directions, etc. In the editor, it becomes a blinking vertical line between letters, called the *insertion cursor.* Apple provides routines allowing developers to modify the appearance, size, and color of many of these shapes. Indeed in Apple's System Preferences, the Accessibility panel allows *Users* to change the size, outline color, and fill color of cursors. But the insertion cursor is special and these preference items do not change it. Also there are no developer routines to modify the insertion cursor.

A user with vision problems asked that TeXShop provide a block insertion character. Some rare programs like Terminal provide such a cursor, but no Cocoa routines mention it, and modifying Apple's NSTextEdit class to provide one would be a daunting task. On the other hand, I lost the central vision in one eye several years ago, so I was sympathetic.

TeXShop has two related features which make using the insertion cursor easier for many users. The first is called *Highlight Current Line.* This feature can be turned on in the Source menu for the active document, and can turned on when documents first open in TeXShop Preferences under the Editor tab. When this feature is on, the line containing the insertion character is given a pale blue background. This color can be revised in Themes, where it is called a Syntax color named "Current Line".

Some authors like to issue a line feed after each source line, while other authors only issue a line feed at the end of paragraphs. So the line with the pale background may be a short single line, or a full paragraph.

For further emphasis of the insertion cursor itself, a second item named *Use Block Cursor* is available. This item is really a second mode for "Highlight Current Line", changing its default behavior. The two features cannot both be active at the same time.

To turn the block cursor on for all new documents, select it under the Misc1 tab in TeXShop Preferences. This will override the item "Highlight Current Line" in the Editor tab, which can be either on or off. To modify the block cursor using a menu item for the active document, toggle *Use Block Insertion Cursor* on or off in the Source Menu. Turning this item on will also turn on "Highlight Current Line". Thus if you later toggle the block cursor off, the current line will be highlighted instead.

The block cursor works by syntax coloring the backgrounds of the adjacent characters just before and just after the insertion line, while leaving the original insertion line to blink as before. It provides a color for this background which can be adjusted by a preference item in the Misc1 tab.

The block cursor cannot color the background of a character that does not exist. So if there is no character before the insertion character, only one character's background is colored,

and if there is no character after the insertion character, only one character's background is colored. In particular, when starting a brand new line, nothing is colored and only the original insertion character shows. When editing existing text, both sides show and the insertion character is highly visible.

The features of this block cursor can be modified in TeXShop Preferences under the Misc1 tab. The defaults described above seem best to me. I recommend that users try this cursor and decide for themselves if it is useful or distracting.

Chapter 24

Spell Checking

24.1 Introduction

TeXShop uses Apple's NSTextEdit Cocoa Framework to provide editing in the Source Window. In typical Cocoa style, TeXShop adds or modifies features but an astonishing list of features come directly from Apple without any TeXShop code. This includes obvious features like copy and paste, and subtle features like the switch to entry from right to left when writing in Arabic, Hebrew, and Persian.

In particular, spell checking comes from Apple. macOS uses an array of built-in dictionaries, depending on the user's language, to identify incorrectly spelled words. It controls the GUI structures which reveal these words to the user and allow them to be corrected.

Unfortunately, standard LaTeX commands are not in these dictionaries and thus are marked as misspelled. There are several approaches to fixing this problem. Before describing these approaches, let us discuss the standard ways spell checking can be controlled in Cocoa.

Apple provides three ways to spell check text, and TeXShop inherits these three methods. The methods are activated in TeXShop's Edit menu, and some are selected by TeXShop Preference settings.

The first of these items is titled *Check Spelling*, and has a keyboard shortcut "command ;". When this combination is pressed, the first misspelled word is highlighted. Each additional press causes TeXShop to jump to the next misspelled word and highlight it. This spell check command is thus a glorified search in which only misspelled words are found.

A second way to spell check is to activate the menu item *Correct Spelling Automatically*. If this menu did not exist, the feature would be controlled in Apple's System Preferences,

where it is on by default. When it is on, your computer becomes a giant iPhone, standing behind you and changing what you type into what it thinks you ought to have typed. Instead of turning it on, TeX users are likely to want to turn it off. Therefore I added the menu *Correct Spelling Automatically* to TeXShop, and by default it is off.

The final way to spell check is to use the menu item *Check Spelling While Typing*. This item underlines misspelled words as they are typed, and the user can then go back and correct these words. This is how I spell check in TeXShop.

24.2   Dictionaries

The Macintosh has many dictionaries; special commands choose the active dictionary. A "Dictionary" field in TeXShop Preferences under the Source tab has a pop-up menu which can be used to select the default dictionary. An unexpected feature is that dictionaries are listed using the ISO 639-1 and ISO 639-2 standards rather than the localized names shown in the "Show Spelling and Grammar" pane to be discussed later. These ISO values are easy to decipher. The preference menu also has an "Automatic" item in which the computer selects the appropriate dictionary based on the localization you are using.

When a new file is opened in TeXShop, it will be set to use the default dictionary. But this dictionary can be changed for that file by opening "Show Spelling and Grammar" in the Edit menu and selecting a new dictionary. In the small panel that opens, select a dictionary using the pull-down menu at the bottom.

A user who writes in English but corresponds with a French relative can easily do that when writing a note to that relative. A more unusual situation occurs if a user has several files open at once, some written in one language and some in another. Activate each source file by clicking on the text, and then select the dictionary for that file using the "Show Spelling and Grammar" panel. Click randomly on the various source files and notice that the dictionary field changes to the correct dictionary for each file.

Finally, if you intend to work on a file for an extended period of time and it does not use your default dictionary, the default dictionary for that file can be set with an instruction at the top similar to

```
% !TEX spellcheck = de-DE
```

This particular document will then open with spelling set to German. But the "Show Spelling and Grammar" panel can later be used to switch dictionaries temporarily, in case a German letter contains an English quotation which needs to be spell checked.

After working on these features, I mentioned them to a user who told me in a disappointed tone that he really just wanted to return to the days when TeXShop entirely ignored the

"Show Spelling and Grammar" panel and let it "do its thing." There is a special hidden preference for that user:

```
defaults write TeXShop OriginalSpelling YES
```

24.3   Spell Checking LaTeX Commands

This section is about code sent to me by Neil Sims, the Head of the Department of Mechanical Engineering at The University of Sheffield. It concerns the problem that most LaTeX commands are marked as misspelled. Thanks to Sims, TeXShop can now handle this problem — for some users — while using the standard Apple spell checker and standard Apple dictionaries.

What is the mechanism Sims uses to turn off spell checking? I wish I had thought of it. The text in the TeXShop source window is an "attributed string." This means it is an ordinary (often very long) string, with an additional data structure associated with the string that lists attributes like "text color" and "background color" for selected ranges of the string. Sims noticed that one of the available attributes is "do not spell check this selection." So Sims added lines to TeXShop's syntax coloring code which prevent the Mac from spell checking TeX commands and certain other words. This means in particular that the feature only works if syntax coloring is turned on.

In TeXShop Preferences under the Source tab, there is a box of selections labeled *Spell Checking.* These items are off by default. Leave them off if you use cocoAspell or any spell checking method except "Check Spelling While Typing."

The first spell checking item turns off spell checking for all TeX command words like:

```
\documentclass, \usepackage, \begin, \alpha
```

The second item is more complicated and will be explained in the next paragraph. The third item turns off spell checking inside comments. Some users may write little essays as source comments and prefer to leave spell checking on for them.

Many TeX commands have optional parameters [...] and mandatory parameters {...}. The entries inside these parameters can also be specialized TeX words. That is true of the first example below, and it is annoying if the spell checker marks "parfill" and "parskip" as misspelled. On the other hand, in the second example the parameter is a user-supplied string which ought to be spell checked.

```
\usepackage[parfill]{parskip}
\emph{This remark has been made before}
```

The second item in the *Spell Checking* box of TeXShop Preferences turns on a somewhat crude method of handling both kinds of parameters. When this item is on, most TeX command parameters are spell checked. But TeXShop has an internal list of certain TeX commands whose parameters contain specialized words, and for these it turns off spell checking for the first two parameters (if they occur in the source).

This internal list, incidentally, is exactly the list marked for special handling by cocoAspell. But cocoAspell is more intelligent, and can mark which parameters to spell check and which to skip.

There is a hidden preference setting to extend the list of specialized TeX commands. The first command below adds one more element to the existing array of user supplied exceptions. The second command erases the array so the user can start over. Note that neither command affects TeXShop's default list of special commands.

```
defaults write TeXShop UserCommandsWithoutParameterSpellCheck -array-add "\\verbatim"
defaults write TeXShop UserCommandsWithoutParameterSpellCheck -array
```

There are some minor glitches with Sim's method. When a document is first opened, all TeX commands will be marked as misspelled. But a single click in the edit window will fix this problem. Similarly, while source is being typed, some commands may be marked as misspelled, but the mark will be removed when RETURN is pressed. Unfortunately, the new attribute is totally ignored by the "Check Spelling" search item, so it will not help when you go through the manuscript word by word looking for misspellings.

24.4   CocoAspell

Coco-Aspell is an alternate spell checker by Anton Leuski. Leuski's project allows users to replace Apple's own dictionaries with new dictionaries that can be made "LaTeX aware", while still using all the Apple spelling facilities. Leuski made the project open source recently; see https://github.com/leuski/cocoAspell. I think of the project as the "gold standard" for TeX-aware spell checking, and if you use it, then the *Spell Checking* preference items discussed in the previous section should be turned off.

Coco-Aspell can be difficult to install, and the latest commit to the open source project was August 5, 2017. Leuski understandably stopped working on the project and nobody has stepped up to take it over. Users must then use the dictionaries supplied with cocoAspell, rather than dictionaries by Apple or others. Below are two documents from 2019, a short description by Koch, and a longer description on installing the code by Schulz. These are the latest documents I possess on CocoAspell.

24.5 Cocoa-Aspell Usage (Koch)

cocoAspell is a spell checking service which attaches to the build-in spell checker from Apple. It knows about LaTeX and does not mark LaTeX keywords as misspelled. Since it is not notarized, it cannot be provided in our install packages; obtain it directly from the author's web site.

The spell checker installs a Preference Pane named "Spelling" in Apple's System Preference. After running the cocoAspell installer, open this pane and configure it appropriately.

The Pref Pane requires macOS El Capitan or higher, because Apple changed the requirements for Preference Panes just before El Capitan. The cocoAspell pane meets all current Apple standards.
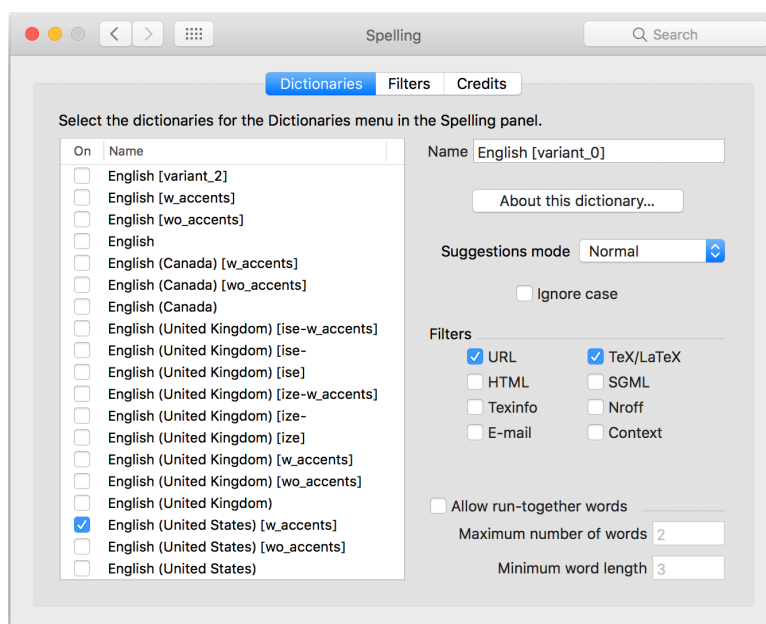
On the left side, select those dictionaries which should appear in the Mac interface. For each selected dictionary, turn on "Latex" and possibly "Context". Notice that all dictionaries are variants of English. If you use another language, go to the web page to obtain and install an appropriate dictionary for that language.

Now in the TeXShop Edit menu, show "Show Spelling and Grammar." A panel will appear, shown below. At the bottom of this panel, select a cocoAspell dictionary instead of a built-in dictionary. Once it is selected, LaTeX keywords will not be marked as misspelled. Your editor should remember this choice across reboots.
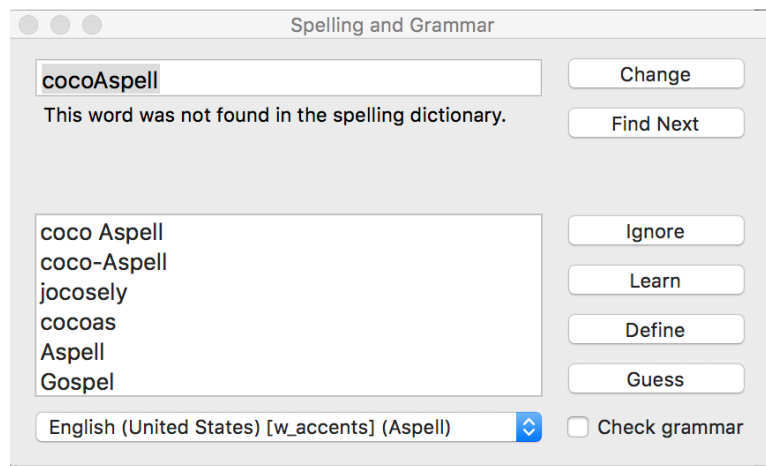
24.6 Coco-Aspell Installing(Schulz)

24.6.1 Introduction

`CocoAspell` is a LATEX-aware Spell Checker that operates through Apple's `Spell Checking` System. Version 2.5, for `macOS 10.11` (`El Capitan`) and later, is available at <`http://people.ict.usc.edu/~leuski/cocoaspell/`> or as part of `MacTeXtras` available at <`http://www.tug.org/mactex/mactextras.html`>.

If you've never installed `cocoAspell` before *and* you are using `OS X 10.13.x` (High Sierra) (or later?) the install may partially fail; the dictionaries are not completely built under that OS version (or later?). There is a fix, provided by Thaddeus Perala, but it involves using the Command Line (`CL`) tool called `make`.

24.6.2 First Check for `make`

First run the command

```
make --version
```

in Terminal. If `make` is already installed you will get a short paragraph that looks like

```
GNU Make 3.81
Copyright (C) 2006  Free Software Foundation, Inc.
This is free software; see the source for copying conditions.
There is NO warranty; not even for MERCHANTABILITY or FITNESS FOR A
PARTICULAR PURPOSE.

This program built for i386-apple-darwin11.3.0
```

which means all is well and you can continue with the fix for the `cocoAspell` installation below.

If it comes up with a dialog box you don't have the `CL` tools installed. To install the `CL` tools click on the `install` button in that dialog box. Note: unless you *really* want to use `XCode` in the future don't click on the `XCode` button since that is a very large download and application while the `CL` tools take very little space. Once those tools are installed run

```
make --version
```

in Terminal again and you should get the short paragraph given above.

24.6.3 Building the `cocoAspell` Dictionaries

Once you know you have the `make` tool installed simply run the two commands

```
cd "/Library/Application Support/cocoAspell/aspell6-en-6.0-0"
sudo make install
```

in Terminal. You will have to give your administrator password to run the second command. After some output the prompt should return and you can open the `cocoAspell Spelling` pane in `System Preferences` to finish your configuration.

Chapter 25

Japan

For reasons I do not understand, a series of very important TeXShop code contributions have come from Japan. The OgreKit Find Panel is a project by Isao Sonabe used by several open source programs. It provides support for regular expressions. The Macro support comes from Mitsuhiro Shishikura, who also created the magnifying glass and the original Display and Magnifying Modes for the Preview Window. More recently, Yusuke Terada contributed the Character Info panel to obtain Unicode information, the invisible character support, several different ways to simplify entry of matching parentheses during text entry, and other features.

All of these people sent me code with only a few connection points, and then sent very detailed instructions. Ten minutes after receiving an email, their code would be connected up and working in my copy of TeXShop!

There are special problems when typesetting Japanese text, and I understand only a tiny fraction of what is required. I first heard of problems when Japanese students at the University of Oregon came to my office with bug reports about TeXShop. These reports made no sense, and seemed to involve the yen symbol. The students were emissaries rather than users and didn't have details.

Eventually I learned the story, which is amusing. When Apple originally sold Macs in Japan, they used the standard American keyboard. There was one problem. Even if the owner typed in English, a yen symbol was essential. So Apple used the backslash key, changing the symbol on the key top to a yen, and changing the backslash to a yen in all the fonts. This caused no problem in LaTeX because the input code was still ascii for backslash.

Then later, Apple grew more conscientious about unicode, and now their keyboards really output a Unicode Yen. Luckily, Donald Knuth had not hardwired the backslash character

into TeX, but had made that character configurable. Since the Japanese were used to using the yen symbol, they just recompiled TeX so commands started with yen. However, this required that all of the style and package files be slightly edited to also use a yen.

Because of the strong input from Japan, TeXShop has a few Preference items which only affect users in Japan. I do not understand these items. For instance, the list of invisible characters which can be shown contains a "Japanese space". The Misc2 tab has an item about pTeX (for Japan), and an item labeled "Automatic UTF-8-Mac to UTF-8 Conversion". The TeX Live Ghostscript installer also installs a few special files for CJK users. Other users can ignore these items.

Chapter 26

Updating TeXShop

TeXShop can be updated using the Sparkle mechanism described at the end of chapter 20. It can also be updated by going to the TeXShop web site, `https://pages.uoregon.edu/koch/texshop/texshop.html`. Finally, the latest version of TeXShop is available on CTAN, `https://ctan.org`. On the main page, search for "texshop" and go to the "package texshop".

After updating, there are occasionally final steps that only a user can do. If an active engine is updated, TeXShop cannot automatically update it because the user may have modified that engine. If new macros are introduced, TeXShop cannot add it to the Macros list because the user may have edited that list.

There is a document at the top of the TeXShop Help menu named *About This Release*. This document describes any final steps required when updating to the current release. Usually nothing is required. Users sould get in the habit of glancing at this document after each update.

The second item in the TeXShop Help menu is *Changes*, which lists all changes in the new update, and in previous updates. Read the top of this document to learn of new features after updating. This document is also on the TeXShop web site if you'd prefer to read about changes before updating.

Note that the TeXShop web site contains the complete TeXShop source code.

On very rare occasions, users will update and the new version will be unresponsive, or have massive bugs. If this happens, it is useful to reboot your machine and then try the update. This often fixes all problems. If not, the problem is almost always caused by a glitch in TeXShop Preferences; this can happen when new items are added to Preferences. (Even then, the vast majority of users have no problems.)

The solution is to rewrite the TeXShop Preferences data file from scratch. Do this in the following order:

- Quit TeXShop.

- Go to `~/Library/Preferences`, find the file *TeXShop.plist*, and drag this file to the desktop.

- Restart TeXShop.

- Reset any Preference changes you made have made previously.

If this step doesn't fix the problem and you want to return to your earlier Preference settings, you can do so using the saved copy on the desktop. But reinstalling this copy requires care because the system caches the Preference file for extra startup speed. Proceed as follows:

- Quit TeXShop.

- Replace the file "TeXShop.plist" currently in `~/Library/Preferences` with the old backup copy from the desktop.

- In Terminal type

      defaults read ~/Library/Preferences/TeXShop.plist

- Restart TeXShop.

Chapter 27

Left Over Items

27.1 Menus

Open from Stationery

This is the second item in the File menu. It provides an easy way to start a new document. Select the type of document you want to write and a new window will appear filled with the correct starting boilerplate code. This item is customizable using `~/Library/TeXShop/ Stationery`. To create or edit a piece of stationery, put the initial document in this folder with a name suggesting its purpose. Suppose this was Letter.tex. Create a second file named Letter.comment with a sentence explaining the purpose of the document. The comment will appear in the initial dialog when "Open from Stationery" is chosen, and a copy of the stationary will be created. When you typeset, you'll be asked to give a location where it will be stored.

Save This item appears in the File menu, but many users want **Save As ...** Hold down the option key and this new item will be added to the File menu.

Convert Tiff This item appears in the File menu. Pdflatex can accept pdf, png, and jpeg illustrations. In modern versions of TeX Live, it can also accept eps illustrations because the system automatically converts them to pdf during typesetting. This leaves tiff as an unsupported file type. This command converts tiff files to png format.

Line Number This command in the Edit menu brings up a small dialog. Type the line number of the source line you'd like to go to, and the command will take you there.

Close Current Environment

Suppose you have typed

```
\begin{itemize}
\item one
\item two
```

Select the menu item "Close Current Environment" and the following line will be added:

```
\end{itemize}
```

Close Tag Suppose you have typed

```
<start>
<bf> This is nice </bf>
```

Select the menu item "Close Tag" and the following item will be added

```
</start>
```

Toggle XML

When a source file with extension html, xml, or ptx is opened, TeXShop syntax colors it as an XML file rather than as a TeX file, and provides a different set of command completion items for the file. This item toggles this feature on or off regardless of the type of the file. Thus the syntax coloring of an XML file can be changed to ordinary TeX syntax coloring, and the syntax coloring of an ordinary TeX file can be changed to syntax coloring suitable for XML and html.

Save Source Position for Portable

Many users have a portable computer attached to a large monitor. They used the command "Save Source Position" to fix the size and position where source files open on the large screen. But when they travel, this extra room is not available. "Save Source Position for Portable" sets an alternate size and position for opening source files on the portable. TeXShop can distinguish the two cases, so source files then open in a reasonable position when traveling.

Save Preview Position for Portable

This commands behaves the same way for the Preview window.

Source ↔ Preview

If a source window is active, this command brings the corresponding Preview window to the front. Conversely if a preview window is active, the corresponding Source window

comes to the front.

Next Source Window

This command cycles through open source windows, bringing each successively to the front.

Previous Source Window

This command reverses the cycle direction.

27.2   Items in the Preference Pane

Distiller

TeXShop sometimes needs to convert a postscript file into pdf. This is usually done with Ghostscript, but users who did not install Ghostscript can use a distiller provided by Apple in macOS. In case one method is unsatisfactory, it can be useful to switch to the other temporarily.

On Big Sur and Later

In Big Sur (macOS 11), Apple introduced a new style for toolbar items. The new style only works on Big Sur and higher.

Since TeXShop runs on High Sierra (macOS 10.13) and above, it has to be able to draw toolbar items in both the old and new styles. This preference item allows users of recent systems to retreat to the old styles for toolbar items. I *strongly recommend* switching to the new style.

27.3   Modifying the Latex Panel

The Latex panel contains a "Custom" tab; up to sixteen user-defined buttons can be placed on this tab. Other buttons in the Latex panel cannot be changed, but the text inserted when the button is pressed can be changed.

To do so, edit the file `~/Library/TeXShop/LatexPanel/completions.plist` with TeXShop. The comments at the start of this file explain how to make these changes. Be sure to edit and save in UTF-8 format if you use Unicode characters.

27.4   Adding to the List of Files That Can Be Typeset

TeXShop can open a wide variety of files: tex, ltx, dvi, html, pdf, etc. The list contains many different kinds of text files, but only some of these files can be typeset. For other text files, the typeset button will be inactive, and the text will not be syntax colored.

Currently only files with the following extensions can be typeset:

```
tex, dtx, ins, sty, cls, rnw, def, ltx, mp, mf, bib, htx,
sk, skt, ly, Stew, lytex, ctx, bbx, cbx, gabc, gtex,
md, fdd, lhs, lua, xml, html, ptx, asy, lbx
```

There is a hidden preference to add file extensions to this list. For example, a user wrote me wanting to typeset TeXInfo files, which have extension ".texi". He had written an engine which called "texi2pdf" to do the typesetting; this binary is not in TeX Live, but it is available in MacPorts. However, the typeset button was inactive when he wanted to call the engine.

To add ".texi" to the list of filetypes which can be typeset, set a hidden preference in TeXShop using Terminal:

```
defaults write TeXShop OtherTeXExtensions -array-add "texi"
```

This adds texi to an internal hidden list of additional types. The command

```
defaults read TeXShop
```

will list all TeXShop Preference items, including hidden items, and their values, so the hidden list can be determined. To start over by erasing the entire list, enter

```
defaults write TeXShop OtherTeXExtensions -array
```

Chapter 28

Magic Comment Lines

Earlier sections listed "magic comment lines" to set the typesetting engine, encoding, root file, and other items. These lines are ignored by TeX, but seen by TeXShop. The lines should appear in the top 20 lines of a source file.

A small number of these lines are used routinely. Others are used in specialized situations. A few are obsolete. Here is a complete list.

28.1 Commonly Used Magic Lines

% !TEX program =
% !TEX TS-program =

```
Set the typesetting program used for this file, overriding all other ways
to determine the program. The words ``tex'', ``pdftex'', ``latex'', ``pdflatex''
indicate the built-in TeX + DVI, pdftex, LaTeX + DVI, pdflatex engines.
Otherwise use the exact name and spelling of the appropriate engine.
```

% TEX encoding =

```
Set the encoding used to open the file, overriding all other methods
to determine this encoding. Irrelevant if all files use UTF-8 Unicode.
Important if you deal with a mixture of encodings
```

% TEX root =

```
Point to the root file which includes this file. Can be a full path
or a relative path. Should appear at the top of all included or
input files, but definitely not at the top of the root file.
```

% TEX parameter = % TEX TS-parameter =

> Occasionally used. Equals the string sent to an engine as the
> second argument $2 of an engine command. If the magic line
> is missing, the second argument will be a single space.

% TEX spellcheck =

> The exact name of the Spelling Dictionary to be used with this
> source file. Only important when a user writes documents in
> multiple languages.

28.2   Magic Lines to Control Preview in Double Page Modes

% TEX PageDirectionL2R

> The choice for languages written from left to right. Pages flow
> from left to right.

% TEX PageDirectionR2L

> The choice for languages written from right to left. Pages flow
> from right to left.

% TEX bookDisplay

> The initial page stands alone, and then double pages begin.
> Since books start with an odd number of preliminary pages,
> this causes the page pairs to correspond to the two pages
> of an open book. This is the default behavior of the Preview
> Window.

% TEX standardDisplay

> The double pages begin immediately. Useful in Japan for
> certain special books.

28.3   Magic Lines for ConTeXt

% TEX useAlternatePath

> Tells the typesetting engine to find the binary program in
> the location "Alternate Path" set in TeXShop Preferences
> under the Engine tab. This is mainly used by ConTeXT,
> which provides an alternate distribution containing only
> files required by ConTeXt.

% TEX useConTeXtSyncParser

```
The author of ConTeXt, Hans Hagen, likes to rewrite
portions of the TeX infrastructure. He rewrote the
SyncTeX system for ConTeXt, and provided his own
file to parse the information in the resulting synctex file.
This line causes TeXShop to use that alternate parsing
method.
```

% TEX useOldSyncParser

```
This magic line is obsolete. Hans Hagen first wrote
his own code to create the synctex file output by Jerome
Lauren's SyncTeX, but he based his code on the original
version of SyncTeX rather than the current version. This
command  parsed that very old style file. This magic line
might well disappear in later versions of TeXShop.
```

28.4   A Magic Line for Beamer

% TEX pdfSinglePage

```
Sets the Preview Window's Display Mode to SinglePage
style even if it is set in preferences as MultiPage. When
creating and displaying slides, single page is the appropriate
choice.
```

28.5   Obsolete Magic Lines

% !TEX useTabs % !TEX tabbedFile(

```
These commands are described in the Changes document
and were created when Apple first introduced tabbed windows.
There are now better ways to use these windows described
in this manual. The old magic lines interfere with these
better methods.
```

% !TEX projectFile =

```
A very old method used to handle projects with a root file and
included chapter files. Hasn't been used in the last decade.
```

28.6 Pseudo-Magic Lines for Engine Files

Very recent versions of TeXShop introduce magic lines to be used when constructing engine files. These lines can appear anywhere in the engine, and can start at any position in a line. The first character of these lines is not a comment symbol because engines can be written in many languages, each with its own method of writing comments. But the magic lines should appear within appropriate comments so the engine itself does not see them.

Here are the magic lines:

```
!TEX-noPreview
!TEX-pdfPreview
!TEX-htmlPreview
!TeX-bothPreview
!TEX-noConsole
```

The standard behavior of TeXShop still applies if an engine has none of these lines. After executing all lines of the engine, TeXShop looks in the document directory to see if there is a file with the same name as the source, but extension ".pdf". If so, it opens this pdf file in the Preview Window, unless the Option Key is depressed.

If one or more commands are present, then

- if the line "!TeX-noPreview" is present, no preview window appears even if some of the other lines are also present;

- otherwise if the lines "!TEX-pdfPreview" and or "!TEX-bothPreview" are present, TeXShop searches the document directory to see if there is a file with the same name as the source, but extension ".pdf"; if so, it opens this pdf file in the Preview Window;

- if the lines "!TEX-htmlPreview" and or "!TEX-bothPreview" are present, TeXShop searches the document directory to see if there is a file with the same name as the source, but extension ".html"; if so, it opens this html file in the HTML Window;

- and if the line "!TEX-noConsole" is present, the console is not shown.

It is unlikely that this final line will be used with any engine except the one which directly opens html code without any intervening typesetting.

Chapter 29

Hidden Preferences

29.1 Introduction

A great many TeXShop Preferences can be set with the Preference Panel. But there are also "hidden preferences" for obscure settings that can only be changed with the command line. Changing these hidden items is discouraged, because it is easy to forget that they were set, buy a new machine, and find that the behavior of TeXShop is different.

Hidden Preferences are sometimes added when a new feature is introduced. If users don't like it, there is a hidden way for them to retreat to the previous behavior. Another use occurs when a bug is discovered that I don't know how to eliminate but do know how to mitigate. In that case, a hidden preference may be provided temporarily until the correct fix is discovered.

There is a way to discover all TeXShop Preferences and their current values. In Terminal, type

```
defaults read TeXShop
```

29.2 Preferences Created Before Yosemite

These preferences have existed for a long time. Most of them are still active, if seldom used.

<Obsolete Color Preferences>

Originally, many syntax colors, foreground and background colors, and other TeXShop colors could be modified using hidden preferences. These preference items still exist, but they now do nothing. Modern colors are determined by themes, which are stored as small files in `~/Library/TeXShop/Themes` rather than in the Preferences database.

With syntax coloring, comments are colored red, commands are colored blue, and the symbols $, {, and } are colored dark green. These colors could be changed. A color is determined by the red, green, and blue components of the color; each is a number between 0.00 and 1.00.

```
defaults write TeXShop commentred 0.0
defaults write TeXShop commentgreen 1.0
defaults write TeXShop commentblue 0.0

defaults write TeXShop commandred 0.0
defaults write TeXShop commandgreen 1.0
defaults write TeXShop commandblue 0.0

defaults write TeXShop markerred 0.0
defaults write TeXShop markergreen 1.0
defaults write TeXShop markerblue 0.0
```

The background color of the source window could be changed.

```
defaults write TeXShop background_R 0.42
defaults write TeXShop background_G 0.39
defaults write TeXShop background_B 0.77
```

The text color of the source window could be changed. This change requires that syntax coloring be on.

```
defaults write TeXShop foreground_R 0.42
defaults write TeXShop foreground_G 0.39
defaults write TeXShop foreground_B 0.77
```

The color of the insertion point in the source window could be changed.

```
defaults write TeXShop insertionpoint_R 0.42
defaults write TeXShop insertionpoint_G 0.39
```

```
defaults write TeXShop insertionpoint_B 0.77
```

The background color of the preview window could be changed. This background color will not affect printing.

```
defaults write TeXShop Pdfbackground_R 0.42
defaults write TeXShop Pdfbackground_G 0.39
defaults write TeXShop Pdfbackground_B 0.77
```

The transparency of the source, preview, and console windows could be changed. Here an alpha value of 0.00 is completely transparent and an alpha value of 1.00 is completely opaque.

```
defaults write TeXShop ConsoleWindowAlpha 0.75
defaults write TeXShop SourceWindowAlpha 0.75
defaults write TeXShop PreviewWindowAlpha 0.75
```

Version 2.38 of TeXShop introduced new parenthesis matching code and the ability to display invisible characters. The new code employs three colors: a temporary color assigned to parenthesis pairs (magenta), a temporary color assigned to the background of the text between these parenthesis pairs (yellow), and the color of invisible characters (orange). The first of these colors could be changed in the Preference Panel. The other two colors were controlled by hidden preference items:

```
defaults write TeXShop highlightContentRed 1.00
defaults write TeXShop highlightContentGreen 1.00
defaults write TeXShop highlightContentBlue 0.00

defaults write TeXShop invisibleCharRed 1.00
defaults write TeXShop invisibleCharGreen 0.50
defaults write TeXShop invislbleCharBlue 0.00

defaults write TeXShop ReverseSyncRed 1.00
defaults write TeXShop ReverseSyncGreen 1.00
defaults write TeXShop ReverseSyncBlue 0.00
```

Version 2.38 also marks matching parentheses with a yellow badge over the parenthesis which flashes briefly and then disappears. To turn the badge off:

```
defaults write TeXShop brieflyFlashYellowForMatch NO
```

\<Preferences for Line Spacing in the Source Window\>

The command below sets the line spacing in the source window. The default value 1.0 corresponds to standard single spacing, the value 10.0 corresponds to double spacing, and values outside the range $0.5 < value < 40.0$ will be ignored.

```
defaults write TeXShop SourceInterlineSpace 1.0
```

<Preferences for Trashing AUX Files>

Occasionally during typesetting, .aux files, .log files, and other files created automatically by TeX become corrupted and must be removed. TeXShop has a "Trash AUX Files" menu item and associated button in the console window. When either item is selected, TeXShop moves to the trash all files in the folder containing the source file with the same name as the source name, and extensions .aux, .bbl, .blg, .brf, .glo, .idx, .ilg, .ind, .ioa, .log, .log, .lot, .mtc, .mlf, .out, .pdfsync, and .toc. Other extensions can be added to this list, one by one. For instance, to add .dvi files to this list:

```
defaults write TeXShop OtherTrashExtensions -array-add "dvi"
```

To remove all added extensions and return to the original list:

```
defaults write TeXShop OtherTrashExtensions -array
```

Sometimes a more extensive cleanup is desirable. If the option key is held down while choosing "Trash AUX Files", TeXShop uses the

```
% !TEX root = ...
```

mechanism to find the root file. It then moves all files in the folder of this file and any subfolders of this folder to the trash if they have appropriate extensions, regardless of the names of the files. This behavior can be made the default behavior for "Trash AUX Files" without using the option key using the command

```
defaults write TeXShop AggressiveTrashAUX YES
```

<Preferences for Command Completion>

TeXShop supports command completion. Type the first portion of a complicated TeX command and push the escape key. The rest of the command will be entered into the source. The full command can be complicated, so TeXShop types "marks" within it; menu commands jump from mark to mark. These commands have shortcuts listed in the menu.

Since jumping to the next or previous mark is a common operation, TeXShop also allows users to type option-escape to get to the next mark and control-escape to get to the previous mark.

A Preference Panel item allows users to use "tab" instead of "escape" as the key which triggers command completion. In that case, the shortcuts which jump from mark to mark become option-tab and control-tab.

Occasionally, third party utilities make use of option-escape, control-escape, option-tab, and control-tab for other purposes. If TeXShop's use of these shortcuts interferes with these utilities, you can turn off these particular TeXShop shortcuts with a hidden preference:

```
defaults write TeXShop CommandCompletionAlternateMarkShortcut NO
```

\<Preferences for the Preview Window\>

When the preview window is updated after typesetting, it comes to the front. This behavior causes trouble for uses with an X11 editor running in Apple's X11 Window Manager. For these users, the behavior can be turned off via:

```
defaults write TeXShop BringPdfFrontOnAutomaticUpdate NO
```

TeXShop can be configured to automatically refresh pdf views when the pdf file changes. Once a second it examines the date and time when the pdf was last written to see if this information has changed. The time interval between these checks can be modified. To do this,

```
defaults write TeXShop RefreshTime 1.00
```

Automatic refresh for pdf views is useful if TeXShop is configured to use an external editor. In these cases, a .tex file is opened, but TeXShop only shows the associated pdf file. TeXShop also allows pdf files to be opened directly; this is useful for a brief glance at illustrations before embedding them in a TeX document. If you want pdf files opened for an external editor to be refreshed automatically, but pdf files opened for a brief glance to be left alone, issue the following command. (The default value is YES.)

```
defaults write TeXShop PdfFileRefresh NO
```

When TeXShop opens a .tex file for an external editor, it checks the dates of the tex and pdf files to make sure that the pdf output is up to date. If this output is not up to date or does not exist at all, TeXShop typesets the .tex file again. To turn off this behavior

```
defaults write TeXShop ExternalEditorTypesetAtStart NO
```

\<Preferences for the Preview Window Drawer\>

When the Preview window first appears, its drawer is hidden. A hidden preference changes this behavior so the drawer is visible when the Preview window first appears.

```
defaults write TeXShop PreviewDrawerOpen YES
```

<Preferences for Automatic Saving>

By default, automatic saving is on. I strongly recommend leaving it on, but it can be turned off via

```
defaults write TeXShop AutoSaveEnabled NO
```

Before the days of Automatic Saving, TeXShop could be configured to save a backup file whenever it saves or typesets a source file. To turn this on, type the following command. Change YES to NO to turn it off.

```
defaults write TeXShop KeepBackup YES
```

<Preferences for the Matrix Panel>

There is a hidden preference to set the default size of the matrix in the Matrix Panel:

```
defaults write TeXShop matrixsize 12
```

<Preferences for Server Problems>

Ulrich Bauer added a patch to make TeXShop perform better when used with a server if AutoSave is on. Many users thank us for this patch, but it caused problems for a few. To turn it off:

```
defaults write TeXShop WatchServer NO
```

<Preferences for the Tags Menu>

TeXShop adds lines which begin with the words

```
\section, \subsection, \subsubsection, \chapter
```

to the Tag menu. To turn this off

```
 defaults write TeXShop TagSections NO
```

ConTeXt users may want additional words to be recognized and added to the Tag menu. For instance, they want

```
\subsubsubsection, \subsubsubsubsection, \part, \title, \subject, \subsubject,
\subsubsubject, \subsubsubsubject, \subsubsubsubsubject
```

The following command adds these words

```
defaults write TeXShop ConTeXtTags
```

<Obsolete Preferences>

These items should never be used, but are listed for completeness.

Before TeXShop had a preference to set the font for the console and log windows, there was

```
defaults write TeXShop ScreenFontForLogAndConsole -bool YES
```

The following preference causes synchronization spots to be shown in the Preview window:

```
defaults write TeXShop ShowSyncMarks YES
```

TeXShop used to support a different set of source commands to determine the typesetting engine, file encoding, and root file; examples are

```
%&latex, %&encoding= UTF-8 Unicode, %SourceDoc ../Main.tex
```

This syntax was a poor choice on my part and has been changed. If you have a lot of old documents, you can temporarily turn this choice back on using the command below. This preference change should only be made in an emergency.

```
defaults write TeXShop UseOldHeadingCommands YES
```

The left and right arrow keys scroll left and right if the preview page is narrower than the total page width, but otherwise they page up and down. A hidden preference changes this behavior so the left and right arrow keys always page.

```
defaults write TeXShop LeftRightArrowsAlwaysPage YES
```

The console text reporting typesetting behavior and errors shows black text. The text can be made to switch to red after the first error with a hidden preference.

```
defaults write TeXShop RedConsoleAfterError YES
```

In the editing window, a soft line break occurs after words. This behavior can be changed to "no line break" = 0, "line break after words" = 1, or "line break after characters" = 2.

```
defaults write TeXShop LineBreakMode 1
```

TeXShop supports SyncTeX. If synctex fails, TeXShop falls back on the Search synchronization method. For users who would like to experiment with synctex by itself, there is a hidden preference to turn off this fallback behavior:

```
defaults write TeXShop SyncTeXOnly YES
```

Early versions of TeXShop considered backslash @ to be a single command which ended with the @ sign, so lines like the one below in style files would only be partly colored. This was later fixed.

```
\ifx\@@input\@undefined\let\@@input\input\fi
```

If you prefer the original method

```
defaults write TeXShop MakeatletterEnabled NO
```

<Preferences to Fix Temporary macOS Bugs>

The items in this section are now irrelevant and mentioned only for historical reasons.When TeXShop was released on Tiger, users ran into an annoying bug which caused the program to gradually slow to a crawl after several typesetting actions. This bug was fixed a couple of days after the release. The problem occurred when a new pdf file was loaded into the PdfKitView in the Preview window. According to Apple documentation, this should have released the previous data structure from memory. The release did occur, but it caused the program slowdown. The bug fix consisted of tricking the system into believing that the data structures were still being used so the system didn't try to release them.

Later a hidden preference was added to release the data if desired. The values of this preference were 0 to release the data on system 10.4.3 or higher, where the bug was less severe, 1 to never release the data, and 2 to always release the data. The default value was 1, causing the program to use the original workaround.

```
defaults write TeXShop ReleaseDocumentClasses 1
```

Apple fixed the problem on Leopard and none of these kludges are active on that system or higher.

When macOS Mavericks was released, the Preview display of text became slightly flurry on many screens. This problem is long gone, but at the time we attempted to improve the situation with

```
defaults write TeXShop FixPreviewBlur YES
```

When macOS Mountain Lion was released, Apple optimized text and font drawing for the Retina display. This caused a few fonts to appear slightly blurry in small sizes on a non Retina display. The following preference reverted back to the earlier behavior. This problem was fixed long ago, so the preference is now irrelevant:

```
defaults write TeXShop NSFontDefaultScreenFontSubstitutionEnabled -bool YES
```

29.3 Newer Hidden Preferences

<Preferences for the Entire Program>

When macOS is shut down, a dialog appears with an optional check box labeled "Reopen windows when logging back in." If this is checked, all programs running when the machine was shut down will restart the next time the machine is rebooted. One user reported that TeXShop gets into an infinite loop when restarted in this way. I could not reproduce this problem.

There is a hidden preference for users who run into this problem. With this preference set, all other programs will start when the machine is rebooted, but TeXShop will not. Restart it manually.

```
defaults write TeXShop IgnoreStartOnReboot YES
```

<Preferences for the Source Window>

There is a preference item forcing TeXShop to place the source window in front of the preview window when opening files:

```
defaults write TeXShop OpenWithSourceInFront YES
```

This manual describes magic lines which manipulate the spelling dictionary. There is a preference to return to Apple's original method of handling this dictionary:

```
defaults write TeXShop OriginalSpelling YES
```

<Preferences for Syntax Coloring>

Syntax coloring was modified for the commants

```
\footnote, \footcite, \autocite
```

to also color inserted text. For example, the entire source phrase

```
\footnote{This is well known}
```

is syntax colored. There is a hidden preference to turn this feature off

```
defaults write TeXShop SyntaxColorFootnote NO
```

In recent versions of TeXShop, the syntax coloring code was turning off while the source file loaded. It is possible that this code was added to fix bugs if syntax coloring starts too soon, but experiments suggest that the bug no longer exists. So in recent versions of TeXShop, syntax coloring begins immediately. In case of trouble

```
defaults write TeXShop ColorImmediately NO
```

TeXShop gives a Command color to symbols beginning with a backslash and continuing with 'a' - 'z' or 'A' - 'Z'. These are the typical commands used by Latex authors. Latex Macro authors also use '@' in commands. A special hidden preference setting adds those to characters receiving a Command color:

```
defaults write TeXShop MakeatletterEnabled YES
```

Latex3 programmers use '_', ':' and '@' in their commands. A special hidden preference setting adds all of these to characters receiving a Command Color:

```
defaults write TeXShop expl3SyntaxColoring YES
```

<Preferences to Control Spell Checking>

TeXShop Preferences under the Source tab at bottom left has an item for Spell Checking. This item is explained in the main part of this Manual. There are some hidden preferences to fine tune this behavior.

The list of specialized TeX commands can be extended.. The first command below adds one more element to the existing array of user supplied exceptions. The second command erases the array so the user can start over. Note that neither command affects TeXShop's default list of special commands.

```
defaults write TeXShop UserCommandsWithoutParameterSpellCheck -array-add "\\verbatim"
defaults write TeXShop UserCommandsWithoutParameterSpellCheck -array
```

The second series of hidden preferences is related to an earlier experiment with spelling and TeX command parameters. Originally I thought it might be better to turn off spell checking the first two parameters for all TeX commands, and then have a list of special commands whose parameters should be checked. This is the opposite of the current logic. It is possible to test this idea:

```
defaults write TeXShop ExceptionListExcludesParameters YES
```

In this case there is a very small list of exceptions, mainly containing

```
\emph, \verbatim
```

but users can again add exceptions of their own:

```
defaults write TeXShop ExtraCommandsNotToCheckParameter -array-add "\\emph"
defaults write TeXShop ExtraCommandsNotToCheckParameter -array
```

The behavior of the Apple Find Bar was changed in TeXShop. Earlier it picked items one by one, but now the entire window being searched darkens during a search and all instances of the phrase being searched stand out in white, with the active element in yellow. To get the original behavior back:

```
defaults write TeXShop IncrementalSearch NO
```

During one update, the code to create the Tags and Labels menus was made more efficient. But in case of trouble, users could revert some of this code. Trouble never materialized, so the following preferences are obsolete:

```
defaults write TeXShop UseNewTagsAndLabels NO
defaults write TeXShop CreateLabelList NO
defaults write TeXShop CreateTagList NO
```

<Preferences for Coloring Indices>

TeXShop has a hidden preference setting to control the "ColorIndex" tool:

```
defaults write TeXShop IndexColorStart YES
```

<Preference for Tabs>

TeXShop has two magic comment lines for tabs, "useTabs" and "useTabsWithFiles". These magic lines are mainly obsolete, as explained in the section of this document on tabbed windows. The first command automatically added "include" files as tabs, and a user asked that this be extended to "input" files. A hidden preference does that:

```
defaults write TeXShop TabsAlsoForInputFiles YES
```

<Preferences for Save Dialog>

The optional menu to select the encoding was removed from the Save Dialog. To get it back

```
defaults write TeXShop EncodingMenuInSaveDialog YES
```

<Preferences for the Preview Window>

In double page mode, the initial page is shown by itself and then pages are shown side by side. This is called Book Mode. A hidden preference setting turns it off

```
defaults write TeXShop DisplayAsBook NO
```

Some users select very high magnification in the Preview Window, and then use a magnifying glass. This can dangerously deplete memory, so TeXShop does not allow the magnifying glass to work when the Preview magnification is above 250. While not recommended, this limit can be raised

```
defaults write TeXShop GlassMaxMagnification 300
```

When syncTeX syncs from the source window to the preview window, it highlights the selected text in yellow. This can be a little difficult to see. Years ago, TeXShop circled the selected text with a red oval rather than highlighting it. To make this the default

```
defaults write TeXShop syncWithRedOvals YES
```

<Preferences for External Editors>

Two hidden preferences were added, but later replaced by better methods:

```
defaults write TeXShop TextMateSync YES
defaults write TeXShop OtherEditorSync YES
```

<Preferences for Accessibility>

Voice Over is technology for those with limited sight; it reads the screen while the user controls the computer with keyboard commands. A user reported that Voice Over works in the TeXShop Source Window, but not in the Preview Window. This bug was in Monterey, and fixed in Ventura.

Further experiments revealed that Voice Over could be tricked into working in the Preview Window by issuing the menu commands "Next Page" and then "Previous Page". The trick requires that documents contain at least two pages. Add a blank page to one page documents to make the trick work for them. This trick is off by default, but can be turned on to work automatically using

```
defaults write TeXShop FixVoiceOver Yes
```

<Obsolete Preferences Introduced To Find a Bug>

Several years ago, a small number of users reported that while typesetting documents with at least 700 pages, typesetting would suddenly stop in the middle without an error message. This was the most difficult bug I ever faced. The Changes document explains how it was finally fixed. Before the fix, several hidden preferences were introduced so users could help search for a solution. All are now obsolete, but we list them below.

```
defaults write TeXShop DoNotFixTeXCrash YES
defaults write TeXShop DisplayLogInfo   YES
defaults write TeXShop UseTerminationHandler YES
defaults write RepeatTypesetOnError13 NO
```

<Preferences Related to the Flash Bug>

When Apple rewrote PdfKit, they introduced a significant bug. When a document was scrolled so the first page was not showing in the Preview window, and then the document

was typeset, there would be an irritating flash when the new version of the document was loaded. I reported this bug to Apple, who eventually told me they would not be able to fix the bug. However, they suggested that I could fix it in TeXShop using the following elaborate method. Just before reloading the page, cover the preview window with a transparent view. Since the view is transparent, users will not notice it. Now draw a picture of the screen into this transparent view. Users will not notice that because the new image matches the image underneath. Now update the document. There will be no flash because it is hidden under the new view. Finally remove the covering view.

This fix works and has been used in TeXShop for several years. But it can be turned off:

```
defaults write TeXShop FlashFix NO
```

A key parameter in this fix is the length of time the transparent view remains on top of the window. If this time is too short, the flash will be visible when it is removed. If the time is too long, users won't be able to interact with the window until it is removed. Experiments showed that the ideal amount of time was a quarter of a second. Users didn't notice with this speed, while they felt a slight delay with longer times. But this value can be reset with a hidden preference:

```
defaults write TeXShop FlashDelay 0.25
```

<Various Obsolete Preferences>

A bug in TeXShop 3.38 was caused by a single line of code added to help a few users in Japan. These users used Japanese input methods, customized the background and foreground colors of the source window, and picked a dark color for the background. The bug was fixed by removing that line of code. The users requiring the bad line of code can get it back by

```
defaults write TeXShop ResetSourceTextColorEachTime YES
```

We earlier discussed a Hidden Preference named FixPreviewBlur related to a bug causing a blur in the Preview Window when shown on large monitors. This bug was fixed long ago. When dealing with this bug, an additional and now obsolete hidden preference was introduced:

```
defaults write TeXShop InterpolationValue 3
```

Here

```
0 = NSImageInterpolationDefault
1 = NSImageInterpolationNone
2 = NSImageInterpolationLow
```

```
4 = NSImageInterpolationMedium
3 = NSImageInterpolationHigh
```

In Yosemite, there was a problem with the elasticity of scrolling in the source window, and a few hidden preferences were introduced to deal with this problem. The problem no longer exists.

```
defaults write TeXShop SourceScrollElasticity NO
defaults write TeXShop FixLineNumberScroll NO
```

In Yosemite, there was a problem scrolling with left and right arrows in the Preview Window. We provided a work-around, which could later be removed if Apple fixed the problem, as they eventually did:

```
defaults write defaults write TeXShop YosemiteScrollBug NO
```

Another bug caused blank split pages in the Preview window after typesetting. Our fix could be turned off if Apple later fixed the problem, as they did:

```
defaults write TeXShop FixSplitBlankPages NO
```

When scrolling the pdf display with a gesture or mouse in High Sierra, a bug prevented the system from updating the page number box. A fix was provided in TeXShop, but fearing that it would affect performance, a hidden preference was also provided to turn it off if Apple fixed the bug:

```
defaults write TeXShop ContinuousHighSierraFix NO
```

Chapter 30

Contributors

A great many people contributed code to TeXShop. For a partial list, see https://pages.uoregon.edu/koch/texshop/extras.html. I'd like to mention a very small subset to illustrate the different ways one can contribute to an open source project.

Overall Architecture

At three different points in the life of TeXShop, programmers rewrote and reorganized large sections of the code. Their work was crucial; without it, TeXShop would have been a lump of spaghetti that could never be untangled. Max Horn, Dirk Olmes, Yusuke Terada.

Specific Features

Some major features in TeXShop are contributions by a single author. Macros and the Macro Editor are by Mitsuhiro Shishikura. Command Completion is by Greg Landweber. The Latex Panel is by Geoffroy Lenglin and the Matrix Panel is by Jonas Zimmermann. Herbert Schulz provided all the TeXShop engine support for John Collin's script pdflatexmk, and wrote crucial documents explaining Command Completion and TeX Live's automatic conversion of eps to pdf during typesetting.

Independent Projects Used by TeXShop

Some independent projects were incorporated whole into TeXShop. OgreKit and the OgreKit Find Panel are by Isao Sonobe. Line numbers in the source window are provided by the NoodleLineNumberMarker and NoodleLineNumberView Cocoa classes by Paul Kim. The Sparkle update code used by TeXShop and many other programs is by Andy Matuschak.

Localization

Localization must be done by native speakers. Nine languages including English are currently supported. French by Jean-Claude De Soza, German by Steffen Wolfrum, Italian by Nicola Vitacolonna, Spanish by Juan Luis Varona Malumbres, Brazilian Portuguese by Emerson Ribeiro de Mello, Chinese by Linus Yang, Korean by Karnes Kim, Japanese by Yusuke Terada.

Icon

The TeXShop program icon is by Thiemo Gamma. Designing icons is the most difficult task a programmer can face; I've never been able to come close. Thiemo's icon showed up in my mailbox one day and I immediately switched to it. I know that Apple's current style is square with rounded corners, but Thiemo's icon will remain.