TUG

RICHARD KOCH

1. Introduction and History

This is the root document for a series of related documents which explain how to construct MacTeX.

Wendy McKay conceived the idea of a Macintosh install package which would provide everything needed to use TeX on a Macintosh. After advocating the idea at a couple of earlier TUG conferences, she organized a lunch for Macintosh users at the TUG Practical TeX Meeting, 2005, in Chapel Hill, North Carolina. At that lunch, she pointed to Jonathan Kew and assigned the construction of the installer to him (how could Jonathan refuse!). Jonathan had to leave the conference the next morning, and we expected that he would construct the package over several weeks after he returned to England. Instead, Jonathan stayed up all night and had a package the next morning. Over breakfast, he willed maintenance of it to me.

Originally, MacTeX installed a TeX Distribution created by Gerben Wierda, based on teTeX. But in 2006, Thomas Esser, the author of teTeX, announced that the project was ending and recommended that users switch to TeX Live. Gerben Wierda then began revising his distribution to contain a mixture of teTeX and TeX Live, announcing the new distribution, gwTeX, in November, 2006 at the annual TUG meeting in Marrakesh, Morocco. But at that same meeting, Gerben held up a sign containing the words "I quit" and announced that he would no longer support his distribution. I then constructed test versions of MacTeX, one using the old teTeX, one using gwTeX, and another using the full TeX Live. To my surprise and delight, the TUG authorities pushed for the package based on the full TeX Live, and since 2007, that is what MacTeX installs.

2. The Various SubProjects

This document is provided inside a build tree for MacTeX. Do not rearrange folders in this tree. As MacTeX is constructed, these folders will gradually be filled with bits and pieces of the package, until finally the complete package is inside one of the folders.

The MacTeX install package contains a series of subpackages which install the various pieces of MacTeX. A user can choose which packages to install by clicking the "Custom" button during installation. Each subpackage is built in a folder of the build system, and each such folder contains a subfolder named "TUG" with the documentation needed to

Date: May 9, 2019.

build that portion of MacTeX. The subpackages needed for MacTeX are Ghostscript, GUI Applications, and TeX Live.

We build a special version of MacTeX for the DVD. That portion requires Ghostscipt, GUI Applications, and TeXDist.

Finally we build two extra packages: BasicTeX and MinimalTeX.

3. Building Binaries

To create a new TeX Live and MacTeX release, it is first necessary to compile the TeX Live binaries. This step is entirely covered by the material in the *Binary* folder of this Build tree. The resulting binaries aren't used directly; instead they are forwarded to Karl Berry, who inserts them into TeX Live.

One document in the Binary folder is so important that it is duplicated at the top level of the folder as the file *BuildStatus-2019*. This is a plain text document. The document explains everything needed to obtain the source code and compile it. To compile, just copy Terminal commands from the BuildStatus document, paste them into Terminal, and execute. The binaries are then copied from the TeX Live build directory to Binary/RawCode.

Asymptote is a special case. Its source is contained in the TeX Live source, but it is built separately. A folder in *Build* called *AsymptoteBuild* contains all material needed to do that. To build, copy this folder to the build platform. Then follow the instructions in the document *AsymptoteBuild-2019* inside the *AsymptoteBuild* folder. The end result will be a binary named *asy*, which is then moved to Binary/RawCode.

A shell script in the *Binary* directory combines these binaries and creates tar.xz files to be sent to Karl Berry.

The documents BuildStatus-2019 and AsymptoteBuild-2019 are plain text files so they can be easily edited. New flags or compile steps may be required in a subsequent year. They should immediately be added to these documents during compiling so there is an accurate record for the future.

In 2019 and the future, we support the versions of macOS still supported with security updates by Apple. In practice, this means the last three systems of macOS. For example, in 2019 we will support Sierra, High Sierra, and Mojave. Apple will introduce a beta of the next macOS at the June developer conference WWDC, and release it in the fall, and we always make certain that MacTeX supports that. So for most of the year we support four versions of macOS.

4. Apple's PackageMaker

All remaining steps create Apple install packages.

Apple supplies command line programs to create install packages, and Jonathan Kew provided a shell script to create his original Install Package by calling these programs.

After I took over, I discovered that some users had their own copies of Ghostscript and only wanted to install TeX Live. So I split the install package into three pieces bundled together, one for Ghostscript, one for GUI apps, and one for TeX Live. A *custom install* option allowed users to select and install only pieces of the complete package. I found command line scripts cumbersome for this more general package, and switched to using a GUI program from Apple called PackageMaker, version 2.

Install packages created by the original PackageMaker were actually folders; the Finder disguised these folders to look like flat files to the user. The packages had extension ".pkg" if they installed a single package, and ".mpkg" if they contained several pieces which the user could selectively install. The ".mpkg" folders contained the individual ".pkg" folders inside an encompassing folder. Since these packages were really folders, they had to be zipped to upload to various servers. This caused problems because some users had third party unzip utilities which did not work properly.

Apple introduced a new operating system, Mountain Lion, in late summer of 2012. Install packages for this system must be *signed*. Install packages created by PageMaker 2 cannot be signed, so we had to switch to PackageMaker 3. This software had a "legacy" mode which creates the original packages we had been using, but these legacy packages could not be signed. Thus it is necessary to switch to version 3 of PackageMaker, and use it to create modern install packages which install on Leopard and higher systems.

PackageMaker Version 3 was never really finished by Apple. It has crashing bugs, but a little practice allows users to avoid these crashes. It claims to be able to do things it cannot do. But with careful use it did the job through the 2019 release. However, Apple stopped including it with XCode, requiring users to download a zip file containing extra tools. The last such zip file was created in 2014. It is still available from Apple web pages, but hidden in a place that requires extra work to find.

Then in 2019, Apple announced that its next system would require 64 bit binaries. PackageMaker contains only 32 bit binaries, so it is very likely that PackageMaker will become obsolete in 2019. On the other hand, Apple recently warned developers that install packages must be notarizing in the next system, and explaining how to do that. So install packages will not be obsolete; from now on they must be created using command line programs.

After creating MacTeX-2019, I spent time converting the system to use these command line tools. In the end it became clear that this method will be much easier to explain and maintain than the old GUI PackageMaker. These command line tools will be used here.

5. An Overview of Package Creation

The BasicTeX install package is a good place to learn how packages are created, because it is self contained and small enough to encourage experimentation.

The first step in creating this and similar packages is to rename all folders in /usr/local. Thus *bin* becomes *bin-temp* and *texlive* becomes *texlive-temp*. This step insures that none of the contents of these folders becomes part of BasicTeX.

Next BasicTeX is installed in /usr/local/texlive/2019basic using the TeX Live Unix installer. This is explained in detail in the BasicTeX portion of this project. After that, a shell script named *buildPackage.sh* copies this distribution to *root/usr/local/texlive/2019basic* inside the BasicTeX folder. Everything now depends on this *root* folder, so the 2019basic distribution in /usr/local can be removed and the contents renamed back to their original names.

The final step will be to run a script which calls an Apple command line app to create an install package, pointing to *root* as the contents of the package. Before discussing that step, however, we have to face the twin issues of notarizing install packages, and creating apps which adopt a hardened-runtime.

6. Notarizing and Hardened Runtimes

By 2002, Apple had released macOS. I retired from the University of Oregon that year, and the UO dorms got ethernet connections. When freshmen moved into their rooms, they found a CD and a page of instructions taped over the ethernet port. The instructions said that students had to install the anti-virus software on the CD to their computer before connecting to the ethernet. "Failure to follow these instructions," the sheet added, "will result in loss of ethernet connections in this room."

The final line on the page read "Macintosh users can ignore these instructions."

Those days have long gone, and anyone who goes to WWDC, the Apple developer conference, will discover that Apple now employs many engineers working on security threats. Unix has very good protection for the kernel, so intruders are not likely to get root access. But most Macs are owned by a single user, and the fear is that one of that user's applications will be hacked and then used to gather dangerous information about the user. Two months ago, I received an email saying "As you see, I broke into your computer. I recorded your actions on video as you visited porn sites, and I found damaging information in your mail. I have your email contact list. Send me $979 in bitcoin this week, or else I will send all the videos and damaging information to everyone on your contact list."

I ignored the message, but it got me thinking. Several years ago, Apple invented a technology called *sandboxing*, and required that all applications in the Apple App Store be sandboxed. A sandboxed application runs in its own environment and is allowed only limited contact with the outside world. Typically it is not allowed to use the video camera, or access the users mail, or access the contact list. It is not allowed to run other programs outside its sandbox. This last restriction is particularly cumbersome; a sandboxed application could not call TeX to typeset a document. Thus the GUI apps installed by MacTeX are not sandboxed, and not available in the App Store.

This year Apple is introducing technology to help the remaining developers maintain security on the Macintosh. It is called a *hardened runtime*. An application adopting this technology executes with "additional security protections and resource access restrictions." There are 13 such restrictions, including using the video camera, using the address book, using the photos library, linking with third party libraries, and executing JIT-compiled code. However, Apple provides 13 exceptions for these restrictions. If an application needs to access the video camera, it can request an exception to that restriction. It is legal to request all 13 exceptions, and then an application runs exactly as it does now. These exceptions do not have to be approved by Apple; they are granted automatically by checking various boxes when hardened runtimes are adopted.

Details about these restrictions and exceptions are available at

`https://developer.apple.com/documentation/security/hardened_runtime_entitlements#overview`.

In summary, this technology helps developers assist efforts to improve security. If they do not use the camera, then even if their applications are compromised, the attacker cannot use the camera. On the other hand, developers can do anything they do now.

To encourage adoption of this technology, the next version of macOS will require that standard methods for distributing software, like zip files, dmg packages, and install packages, be notarized. This involves sending the package to Apple, where machines search for viruses and send back a certificate if none are found. Apple says that no human hands are involved in this process. But command line apps and other applications in such an install package must adopt a hardened runtime.

For the initial TeX Live 2019 release, only one package was notarized: Ghostscript-9.27. This package has two binaries: gs-X11 and gs-noX11. The first is compiled with X11 support and the second is compiled without that support. When the package is installed, a symbolic link named gs is created pointing to an appropriate binary, depending on whether that particular machine has X11.

But X11 on macOS is provided by a third party. So gs-noX11 has a hardened runtime with no entitlements while gs-X11 has one entitlement, allowing it to link with a third party Library.

7. More on Building BasicTeX

When work began on BasicTeX, we created an install package with no hardened runtimes and attempted to notarize it. Notarization failed and Apple sent back a detailed error report, showing that 33 binaries needed hardened runtimes. Thirty of these were in the bin directory. The other three were lz4, wget, and xz, in the directory tlpkg/installer/, within folders named lz4, wget, and xz.

Experiments with shell scripts showed that the 30 binaries were exactly the elements of the bin directory which are not links and for which "file $f" gives "Mach-O 64-bit executable

x86_64". With help from Apple developer support, we wrote a shell script which signed, timestamped, and adopted a hardened runtime for an app and named the script *signApp*. We wrote a second shell script which signed, timestamped, adopted a hardened runtime, and asked for a "third-party Library" exception, and named the script *signAppNoLibCheck*. Then we wrote a shell script which examined the files in the bin directory and applied either signApp or signAppNoLibCheck to the Mach-O apps there. We also individually signed lz4, wget, and xz.

We then ran a script to create an install package from root/usr/local/texlive/2019basic/... The resulting install package is called BasicTeX-2019-Temp.pkg. We applied another script named signPackage.sh to sign the package, naming the signed version BasicTeX-2019.pkg.

We then ran the script sendnotarizerequest.sh, which sent BasicTeX-2019.pkg to Apple for notarization. When the upload was complete, this script printed a message containing a magic number "RequestUUID = 7200....". We copied this number to requestUUID.tex, because it allows us to download an error message if necessary. After a delay of 10 to 15 minutes, a notification and email arrived back from Apple that notarization was successful. We then ran the script stapleresult.sh, which added the certificate of notarization to the install package. The package was ready to be uploaded.

If the message from Apple had instead warned of notarization errors, we would have run the script detailedinfo.sh. Notice that the magic number from the last paragraph is part of this script. The result back from the script would be a url to a web page, and displaying that url in Safari would give details of the errors which caused notarization to be rejected.

When detailedinfo.sh runs, it asks for a password. That password is the password to xcrun, which was passed into the system by sendnotarizerequest.sh. But there is more to it. The program xcrun is a developer tool which contacts Apple, and Apple has set up 2-factor authentication for such tools. But 2-factor authentication does not work well inside shell scripts. So Apple allows developers to assign passwords to a few developer tools in their developer account, and if this password is found, 2-factor authentication is not used.

8. Summary

Each package in MacTeX has similar documentation in a folder named TUG. Almost all packages work like BasicTeX. There is a root folder containing the material to be installed. Part of the documentation explains how to construct this root folder. There is a "script" folder containing the post-install script. There are scripts to build an install package, sign it, sign the apps inside, notarize the package, and add the notarization certificate.

9. How To Make Yearly Updates in Packages

There are some universal steps required to update scripts for a new year. I'll describe the steps here, using the Ghostscript package as an example.

Some packages retain the same name from year to year, while other package names change. When the name changes, it typically contains a a year or version number; for example, Ghostscript-9.27.pkg. Although it will be obvious that the name changes, the individual package TUG document will confirm this fact.

It is important that some packages get new names for the following reason. When a user installs a Package, their Mac stores a receipt for the package. If a new version of the package is later installed, the installer updates files which changed in the new package, installs additional files from the new package, *and removes files that used to be in the package, but now aren't*. Ghostscript 9.27 installs support files in /usr/local/share/ghostscript/9.27, which depend on a version number. If the name of the package didn't change, the old support files would be removed, but we want to preserve them in case the user retreats to the older version. Renaming is particularly important for TeX Live itself, since we certainly don't want the installer to remove the old copy of TeX Live.

To switch to the new name:

- Edit the "buildPackage.sh" script, which may contain references which differ from year to year.
- Edit the files to be shown to the user during installation: License.rtf, ReadMe.rtf, and Welcome.rtf. Edit these files to reflect the new package name and release date. Make other changes as appropriate.

10. Signing Packages

Starting with Mac OS 10.8, Mountain Lion, install packages must be signed. Signing requires a "Signing Certificate" from Apple, which is kept in the developer's keychain and maintained by the program /Applications/Utilities/Keychain Access. The signature is not kept in this MacTeX Developer package — when install packages are signed, the software looks up the developer's signature in their keychain.

Packages must be signed after they are constructed using the command line program "productsign". "Productsign" accepts an input package and outputs a corresponding signed package, so after building, individual portions of this developer package will contain two install packages. The command to run producsign is inside a short script in each folder named "signPackage."

Obtaining a certificate requires a developer account at `https://developer.apple.com/`. Full access costs $100 a year. As soon as you log in, you'll see an entry "Developer Certificate Utility" with lots of information about certificates.

You will ultimately be given two certificates, one for signing install packages and another for signing applications. For instance, my certificates are called "Developer ID Installer: Richard Koch" and "Developer ID Application: Richard Koch". The system will reject the "Application" certificate when signing packages, so use the "Installer" certificate.

Obtaining certificates can be tricky. Good luck.

Apple delivers the certificates in just a few minutes, but the exact name of the certificate is important. I spent several days debugging because I neglected the space between "Installer:" and "Richard Koch". If you are like me, you'll have *lots* of certificates managed by Keychain Access, some obsolete, and it can be difficult to keep straight the active ones. Follow the instructions from Apple carefully, since the process isn't quite as straightforward as it first seems.

The really tricky part comes if you switch computers. Apple has special software to bundle the certificates and keychain information on the old computer, and then install this information on the new machine. Follow these instructions carefully.