TUG

RICHARD KOCH

1. Introduction

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

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

The GUI portion requires a small subpackage named Setup-2015.

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

Finally we build one extra package: BasicTeX.

2. 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 Build tree as the file *BuildStatus-2015*. This is a plain text document. The document explains everything needed to obtain the source code and compile it. The end result of that operation will be a folder containing the 32 bit Intel binaries, a folder containing the 32 bit PowerPC binaries, and a folder containing 64 bit Intel binaries. To compile, just

Date: May 7, 2015.

copy Terminal commands from the BuildStatus document, paste them into Terminal, and execute.

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 each building platform: Leopard Intel and Snow Leopard Intel. Then follow the instructions in the document *AsymptoteBuildStatus-2015* inside the *AsymptoteBuild* folder.

After these binaries are obtained, they are copied to *Binary/RawCodeFromCompile*. Two shell scripts in the *Binary* directory combine these binaries and create tar.xz files to be sent to Karl Berry.

The documents BuildStatus-2015 and AsymptoteBuildStatus-2015 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.

3. About Apple's PackageMaker

Before 2012, MacTeX was built using Apple's original PackageMaker application, version 2. This was replaced in Leopard with a new PackageMaker, version 3, but we continued to use the original version until 2012.

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.

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 has a "legacy" mode which creates the original packages we had been using, but these legacy packages cannot 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.

MacTeX can be constructed on any system which runs the modern PackageMaker and the separate signing software. Currently it is constructed on Yosemite. The modern packagemaker runs only on modern systems, so switching operating systems may require obtaining a new program. But the packages install on Leopard and higher.

As of February, 2015, PackageMaker is available to Apple Developers in the downloads section of the developer website, under the link "Additional Downloads", as "Auxiliary Tools for Xcode Late July 2012" with a date of August 7, 2012. This version, 3.0.6, was released on June 15, 2012.

This version works on Mountain Lion, Mavericks, and Yosemite. According to the link https://discussions.apple.com/thread/4083583, this program is deprecated in XCode 4.6. On the other hand, command line programs to create such packages are still in OS X, and the software to run these packages and actually install is also present.

The link

http://stackoverflow.com/questions/11487596/making-os-x-installer-packages-like-a-pro-xcode4-developer-id-mountain-lion-re

leads to suggested alternatives, including an open source packagemaker by Stephane Sudre,

http://s.sudre.free.fr/Software/Packages/about.html

Just to be safe, the modern version of PackageMaker is in the MacTeX build folder in a subfolder named PackageMaker-ML. If Apple were to truly deprecate PackageMaker, they would also remove the software to install these packages, and then even the open source program would be useless and we'd have to start over in some other way.

The new PackageMaker creates flat files rather than folders. This has a great advantage: it is no longer necessary to zip these packages before placing them on the internet. Before 2012, users often downloaded with a different browser than Safari, and unzipped with a third party application rather than Apple's default utility. Unhappily, some third party unzipping applications don't preserve Unix line feeds, so the resulting packages contain postinstall scripts which won't run.

Switching to the modern PackageMaker has one additional consequence. With the original PackageMaker, we created a ".pkg" install package for each component: TeXLive, Ghostscript, Convert, GUI-Applications, etc. Then these install packages were combined to create the final ".mpkg" MacTeX package. The new PackageMaker requires that we create a "root" folder for each individual package containing the data to be installed, but does not require that we finish the task and create the individual install packages. Consequently, the various individual folders for TeXLive, Ghostscript, etc. support an optional final step to create individual install packages, but this step can be skipped when creating MacTeX. Creating the install package for the TeX Live piece with the modern PackageMaker is very time consuming, so the step should certainly be avoided for that package. On the other hand, it is useful to create a separate flat package for Ghostscript so it can be tested independently of MacTeX.

4. Details for Building the Full MacTeX Package

To show how this works in practice, we'll outline the steps required to build the full MacTeX. The first step is to build the "root" folders in Ghostscript-9.16, TeXLive-2015, and GUI-Applications.

Take Ghostscript-9.16 for example. We rename `/usr/local` and then install Ghostscript on Yosemite, in the process creating a fresh `/usr/local`. Earlier we compiled Ghostscript on PowerPC, Intel-32, and Intel-64, once with support for X11 and once without this support. These binaries are in a subfolder of the Ghostscript folder named RawBinaries-2015. After that, a script combines these binaries into binaries in the "Binaries" subfolder of the Ghostscript folder. We now run the script buildPackage.sh in the Ghostscript folder. It copies the installation in `/usr/local` to the root folder in the Ghostscript folder, and adds additional binaries from the Binaries folder as appropriate. Some of these binaries will be kept and some thrown away in the final postinstall script for the Ghostscript package, depending on properties of the machine where it installs Ghostscript.

Making the TeX-Live root folder is easier. We sent the Mac binaries to Karl Berry at TUG after compiling them, and he inserted them in the TeX Live pretest package. We again rename `/usr/local`. Then we install the TeX Live pretest using the TeX Live install script. Finally we run buildPackage.sh, which copies this installation to the root folder, making a few minor changes.

The GUI-Applications root folder is constructed in a slightly different way. We put the pieces of this installation in a subfolder named "apps" in the GUI-Applications folder. This folder contains several copies of each GUI program for various operating systems. Again we run buildPackage.sh, which copies these pieces into a root folder. During installation, the postintall script will choose the correct version of each of these GUI programs, renaming it appropriately.

In the end, we have root folders in each of Ghostscript, GUI-Applications, and TeX Live.

Now turn to the MacTeX-2015 folder. It contains two copies of the graphic template used to make MacTeX, called "MacTeXMaker" and "MacTeXMaker.copy". The reason for the copy will become clear in a moment.

Double click MacTeXMaker to open it in PackageMaker. This template contains five major section. The first is activated by clicking "Edit Interface" at the top right. PackageMaker then shows the full installation as a user would see it, and allows you to edit the various things seen by this user. Making changes here is only necessary once a year. Indeed, the actual interface just shows the files background.jpg, License.rtf, ReadMe.rtf, and Welcome.rtf, and editing the interface is just a matter of editing these four files.

Four sections remain, the MacTeX-2015 Distribution piece at top left, and the TeXLive-2015, GUI-Applications, and Ghostscript-9.16 pieces at bottom left. These last three pieces correspond to the three choices available for a custom install.

Clicking on any of the four sections reveals settable items at the right. These items need editing only once a year as new packages are created, and the changes required are immediately obvious.

So the sections seen so far only change once a year, and need not be checked during the creation of a new beta of MacTeX. But notice that each of the three sections on bottom left have a "disclosure arrow." Turning these arrows reveals three additional sections, /usr/local for TeXLive, /Applications for GUI-Applications, and /usr/local for Ghostscript-9.16. These three pieces correspond to the three root files which will be used to make MacTeX.

During the beta test period, the contents of these root folders will change. When we converted to the new Packagemaker, some of my tests convinced me that Packagemaker would become confused by new contents in the root folders, and not actually install changed versions. So the method I have adopted is to always recreate the three pieces of MacTeX which depend on the three root files.

To do this, open both "MacTeXMaker" and "MacTeXMaker copy". Place them side by side, keeping track of which is not a copy. Open the disclosure arrow on the copy for Ghostscript-9.16. For a very brief moment, a "busy rotor" will appear as Packagemaker reads in the Ghostscript data. It is very important to wait until this rotor is gone, since two rotors running at the same time can crash Packagemaker.

Now open the disclosure arrow on the active MacTeXMaker for Ghostscript-9.16 and wait until its rotor is gone. On this active copy, drag the "local" icon on the left to the trash. The item will vanish, together with its configuration data to the right. Reach into the Ghostscript-9.16 folder, find the "root" subfolder, open it and find "user" inside and "local" inside that. Drag this "local" to the active MacTeXMaker, dropping it on the Ghostscript-9.16 item. This will create a new "local" piece. A rotor will appear as PackageMaker reads the new data. Wait until it disappears.

Next we recreate the configuration data to the right. Use the copy of MacTeXMaker to find the correct data (that's why it is there). Under the first "Configuration" tab, only the "Install:", "Destination:", and "Package Identifier:" items need to be changed. The "Install" item only needs to be changed to show a relative path. Under the "Contents" tab find the button named "Apply Recommendations" and push it. This gives each installed file the permissions recommended by Apple. Under the "Components" tab there will be nothing showing. Finally under the "Scripts" tab, change the "Scripts directory:" and "PostInstall:" entries.

Repeat these tasks for the TeXLive and GUI-Applications pieces. The TeXLive case will work exactly like the Ghostscript case *except* that it takes Packagemaker a long time to read the root data, so the rotor will turn and turn. Have patience.

In the GUI case, there will be entries under the "Components" tab near the very end of the process. These contents will show all of the GUI applications to be installed, with a checked box to indicate that the installer permits relocation. These boxes are very dangerous because they allow MacTeX to search the entire disk and update any copy of the GUI apps it happens to find So it is is very, very important to uncheck the checked entries in this panel. Complete the final "Scripts" tab items as usual.

Finally click the "Build" button at the top of the template and a MacTeX install package will be created. A save dialog allows you to change where it is saved. Be sure to save inside the "MacTeX-2015" folder containing these pieces.

The only remaining problem is that the package isn't yet signed. Issue the command "sh signPackage.sh" while inside the MacTeX-2015 folder to sign the package. As explained later in the document, this requires that the developer be a registered Apple developer. The short "signPackage" script must be edited slightly to provide credentials for the new developer.

5. Important Warnings

This section contains lessons learned in 2013. The material here is important for all packages made with the new PackageMaker.

Folders in which packages are built should live on the hard disk rather than an external disk. Building constructs subfolders named "root" which contain copies of material to be installed. This root folder and its contents are usually owned by root. But material on an external drive is owned by the user instead, so owners will be incorrect after installing.

The final MacTeX template refers to root folders of the various sub projects. In the template, each sub project is listed in the left column; clicking on an arrow there reveals a panel in which the root folder is determined and configured.

As explained earlier, we accept Apple's recommendation for permissions on installed files. For binary files like texlive/2015/bin/universal-darwin, the recommendations are unexpected. Actual binaries have owner root and group wheel, but symbolic links have owner an individual user and group wheel. Permissions are -rwxr-xr-x. Experiments and email conversations show that these choices work well.

For the GUI-package, the third tab, "Components", is very important. It lists actual apps and offers to relocate them. *Turn this off.* Otherwise the installer won't upgrade programs in /Applications/TeX, but instead will search the disk for a copy to update. Always check that these items are off.

6. MacTeX Products

There are three main products: MacTeX, BasicTeX, and MacTeX Install Step 2. The first is the main distribution for the web. The second is a much smaller TeX distribution for users with limited download bandwidth; it only contains a TeX distribution, without GUI applications, Ghostscript, etc. The third is a variant of the first for the DVD; TeX Live is installed from a copy elsewhere on the DVD, but configured by a piece of this package.

These three packages need to be built last, because they are contain Ghostscript and TeXLive. The packages in this last list can be made in any order. It is useful to create Ghostscript several months before everything else since it does not depend on TeX Live.

7. Platforms to Build MacTeX

We create 32 bit Intel code on an Intel Leopard machine, 32 bit PowerPC code on a PowerPC Leopard machine, and 64 bit code on an Intel Snow Leopard machine.

The machine used to create packages is not crucial. We create the latest packages on Yosemite.

8. 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-9.16.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 9.16 installs support files in /usr/local/share/ghostscript/9.16, 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.

9. Making Install Packages using the PackageMaker GUI

Details of using this GUI for individual packages are given in the TUG documentation for these packages. One GUI item to be set for packages is titled "Requirements". This item can be used to ensure that the operating system is sufficient to support the package. Currently, our packages require at least Leopard, OS X 10.5. However, it is not necessary to set this requirement, because our install packages automatically require Leopard.

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.

The PackageMaker GUI interface contains an entry which tells the software the name of the appropriate signature. So you'd think that setting the signature would be a simple matter of filling in this entry. If the entry is filled in, a signature is certainly added, because a dialog appears at the end of building asking for permission to access that signature. But experiments show that packages created in this way *are not properly signed* and wlll be rejected in Mountain Lion. Ignore this GUI setting.

Instead, 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/`. I believe accounts are free, but they might require a fee; full access costs $100 a year for Mac development and an additional $100 a year for iOS development. Only Mac access is needed. 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. A description of the process, using XCode, can be found in the "developer_id_tutorial.pdf" file included in the "Signing" section of this package. Read the section titled *Obtaining Developer ID Certificates*. This seems to describe

the latest process, but you will find other Apple documentation which describes contradictory and more complicated methods. I already had certificates, so my process didn't exactly follow the scheme in this document. 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.

Testing signed packages is itself tricky. First, the "Security and Privacy" preference pane must be set to allow applications downloaded from "Mac App Store and identified developers" only. After that, if an unsigned package is moved to a Mountain Lion system via wireless or a USB Flash Drive, it will install fine. To test, an install package must be placed on a server and then downloaded with Safari or another browser. Then an incorrectly signed package will refuse to install.