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 are Convert-IM, Ghostscript, GUI-Applications, Latin-Modern-Fonts, TeX-Gyre-Fonts, and TeXLive.

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.

One document in this folder is so important that it is duplicated at the top level of the Build tree as the file *BuildStatus-2013*. 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 copy Terminal commands from this 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, Leopard PPC, and Snow Leopard Intel. Then follow the instructions in the document *AsymptoteBuildStatus-2013* inside the *AsymptoteBuild* folder.

Date: April 13, 2013.

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-2013 and AsymptoteBuildStatus-2013 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 Lion. The modern packagemaker runs only on the system it was designed for, so switching operating systems requires obtaining a new program. But the packages install on Leopard and higher.

As of April, 2013, PackageMaker is available to Apple Developers in the downloads section of the developer website as "Auxiliary Tools for Xcode Developer Preview." The Lion version of PackageMaker is in this documentation. It must be part of the MacTeX build system rather than elsewhere in the file system.

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 separate flat packages for Ghostscript and Convert-IM so these can be tested independently of MacTeX.

Warning: It is important that the folders in which packages are built live on the hard disk rather than an external disk. Building constructs folders named "root" which are copies of material to be installed. This root folder and its contents are often owned by root. But material on an external drive is owned by the user instead.

4. MacTeX Products

There are four main products: MacTeX, BasicTeX, MacTeX-DVD, and MacTeX-Additions. 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; it installs TeX Live from a copy elsewhere on the DVD rather than from a copy inside the package. Finally, MacTeX-Additions installs everything except TeX Live, for users who obtained a TeX distribution some other way.

These four packages need to be built last, because they are contain Ghostscript, Convert from Image Magick, Latin Modern Fonts, TeX Gyre Fonts, and TeXLive. The packages in this last list can be made in any order. It is useful to create Ghostscript and ImageMagick several months before everything else since they do not depend on TeX Live.

5. 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 have been creating packages on Lion machines.

Modern versions of PackageMaker depend on libraries in the operating system and thus cannot be moved from one system to another. Obtain the version for the particular platform used to create MacTeX. The actual install packages are compatible between systems.

6. PackageMaker Bugs

Unfortunately, the modern PackageMaker has significant bugs. One of these bugs caused major problems with the DVD version of MacTeX in 2012. Consequently, in 2013 we create the DVD version of MacTeX using the old legacy packages. We still use the modern PackageMaker to create these legacy packages. Note that DVD install packages do not need to be signed.

We'll describe that particular bug. The original PackageMaker can run two scripts: a preinstall script and a postinstall script. So can the new PackageMaker. But there is one significant difference. In the original PackageMaker, the preinstall script can report an error if conditions on the install machine are bad, and this error is then reported in the GUI before installation is aborted. In the new PackageMaker, there is no way to report preinstall errors to the user.

This difference is crucial for the DVD installer because some users move the install package from the DVD to their install machine, eject the DVD, and then install. This fails because TeX Live is on the DVD but not in the install package. The preinstall script can discover this error, but cannot report it to the user.

The new PackageMaker has an alternate mechanism to report errors to the user. It can call various built-in applescript routines to test common conditions, and report errors from these scripts to the GUI and thus to the user. In 2012, we used this mechanism to test whether the DVD was mounted. The test worked correctly on Leopard and Snow Leopard machines. But unhappily, the built-in applescript always reported an unmounted DVD on Lion and Mountain Lion, destroying the DVD installer on these machines.

7. Creating Legacy Packages

The folder for each subpackage in MacTeX contains a folder named OldStyle. Inside is a script called buildOldPackage.sh. To build an old style subpackage,

```
sudo sh buildOldPackage.sh
```

Building these legacy packages takes much less time than building new style packages. Note that we only need a legacy package for TeXLive-2013-DVD, not for TeXLive-2013, reducing the time considerably.

8. Putting Legacy Packages Together to Form MacTeX-DVD

There is a final problem. It is necessary to combine these legacy subpackages into a final mpkg legacy result. The documentation for PackageMaker says that it supports a command line operation to do that, but that is a lie. The modern PackageMaker cannot make the final package, and the old PackageMaker does not run on Lion.

Luckily, there is an out. Stephene Sudre has written an open source version of the old PackageMaker called *Iceberg*. See

`http://s.sudre.free.fr/Software/Iceberg.html`

This software can combine the individual legacy packages and produce an mpkg output which looks to the user exactly like our previous install Package.

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-9.02.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 9.02 installs support files in /usr/local/share/ghostscript/9.02, 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.
- Edit the GUI project files used by PackageMaker to make the new packages, as explained in individual package documentation.

10. Making Install Packages using the PackageMaker GUI

Details of using this GUI for individual packages are given in the TUG documentation for these packages. But there are issues which affect all packages, so they are discussed here.

One GUI item to be set for packages is titled "Requirements". This item can be used to ensure that the disk has adequate space, that the operating system is sufficient to support the package, etc. 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. If the user attempts to install, say, TeX-Gyre-Fonts.pkg, on Tiger, a dialog will appear with the text "Couldn't open 'TeX-Gyre-Fonts.pkg'. This package type requires Mac OS X 10.5." The dialog only contains one button: OK.

Our packages need to test that there is adequate free space on the disk for their contents. This appears to be an easy matter. Selecting the main install package at top left of the GUI leads to a right side panel containing tabs "Configuration", "Requirements", and "Actions". The Requirements tab leads to a blank list, with plus and minus buttons at the bottom to add and subtract items. If the plus is pressed, a drop down menu presents several items that can be selected, including

- Megabytes Available on Target
- Maximum CPU Frequency
- Memory Available (Bytes)
- Result of Script
- Result of Script for Target
- Result of Javascript
- Result of Javascript for Target

and many more items. However, most of these present a multitude of problems. The items "Result of Script" and "Result of Javascript" lead to the problem of embedding those scripts in the Install package, which isn't done automatically by Packagemaker. Then it turns out that the scripts are called incorrectly by install packages made by Packagemaker. Notes on the internet explain how to get around this problem, but these notes don't work in Lion and Mountain Lion.

The correct approach is to use "Metabytes Available on Target" or "Memory Available (Bytes)". But the first of these works only if the user is allowed to select a target disk for the installation. If we insist on installing in the System Disk (as is appropriate for MacTeX), this command is ignored. So the appropriate command is "Memory Available (Bytes)".

However, trial and error reveals that this command can only test for sizes smaller than 4 gigs (i.e., 4294967295 bytes or less). Larger numbers lead to javascript errors. Luckily, this number is (just barely) large enough to handle MacTeX.

In the end, I decided not to set this check on packages except for one, MacTeX-2013-DVD. The other packages list the amount of space required just before allowing the user to click "Install." Then a package has a custom install option, the amount of required memory

changes with the options selected. So I believe these packages will complain if there is not enough free space. MacTeX-2013-DVD is a special case. Consult the documentation for that package to see how the memory requirement is set.

11. 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". Details are in the individual build packages. "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.

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 certifices, 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." 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.