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. This document lists features common to all pieces.

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 will introduce 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 must switch to PackageMaker 3. This software has a "legacy" mode which creates packages that can be installed on Tiger, but 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.

As of April, 2012, the latest PackageMaker is version 3.0.5. This was a component of the XCode development system until a few months ago, when Apple began distributing a completely self contained XCode to be installed in /Applications, containing all API's, headers, etc. This self contained application does not contain PackageMaker, which available in a separate "Auxiliary Tools" free download from Apple.

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.

There are three main products: MacTeX, MacTeX-DVD, and MacTeX-Additions. The first is the main distribution for the web. The second 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 three 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.

Building MacTeX requires compiling various binaries, and then assembling the results into subpackages and the final package. The assembly step can be done on any reasonable machine; for convenience we use a Lion machine. This is the only machine which needs to have a copy of the texmf tree, copies of the various GUI apps, and so forth.

Compiling binaries is a matter of compiling Ghostscript, compiling convert from Image Magick, and compiling TeX Live. Compiling is done on both PowerPC and Intel machines and then the resulting binaries are combined into universal binaries using lipo. So compiling requires both PowerPC and Intel machines. Currently the PowerPC machine must have operating system 10.5 (Leopard) installed, and the Intel machine must have operating systems 10.5 (Leopard) and 10.6 (Snow Leopard) installed. Intel Snow Leopard is only required for 64 bit binaries.

2. 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.

3. 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-2012-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. I haven't yet checked that.

MacTeX-2012-DVD is a special case. Consult the documentation for that package to see how the memory requirement is set.

4. 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.