DOC-GHOSTSCRIPT

RICHARD KOCH

1. Ghostscript

This is the portion of the TUG documentation which deals with making the Ghostscript package.

2. Overview

From 2017 on, MacTeX will support the platforms for which Apple provides security updates, and thus the three most recent versions of macOS. MacTeX is built on these oldest of these versions. In 2021, MacTeX is built on Mojave and supports Mojave, Catalina, and Big Sur, on both Arm and Intel.

Building Ghostscript is done in three steps. The first builds the binaries, which are placed in the folder "Binaries." Two Intel binaries are built on Mojave, one with X11 support and one without this support. In addition, we build the Ghostscript dynamic library libgs.dylib.9.53.3, but only once because this library requires X11 support. We also build Arm binaries the same way, this time on the oldest system supporting Arm, Big Sur. Then both sets of binaries are merged using lipo, and signed.

In the second stage, Ghostscript is built and installed in /usr/local. Then a buildPackage script copies the resulting installation to the "root" folder in the Build Directory. This script replaces the binaries in this root folder with the carefully constructed binaries in Binaries.

The script makes modifications for TeX Fonts by Arnold-Voisin. The Arnold-Voisin additions were constructed in 2012.

In 2015, very extensive word was done to improve support for CJK fonts by Bruno Voisin, Norbert Preining, and others. Ultimately that led to a script by Norbert Preining called cjk-gs-integrate. See the ReadMe document in MyResources for details.

In the third stage, the root folder is used to construct an install package using the Ghostscript-Build template.

We also build a small install package the same way to make the Ghostscript library. Our final package contains both Ghostscript and this Ghostscript Lib.

Date: March 4, 2022.

To construct the package, download the latest Ghostscript tar file from `http://www.ghostscript.com/` and put the file in the "ghostscriptsource" folder. Then unzip this file; the resulting folder is the source code for Ghostscript.

3. Building Binaries

Build binaries on Mojave for Intel and on Big Sur for Arm. Build a version with X11 support and a version without this support.

Before building, move /usr/local/bin to /usr/local/bin-temp and /usr/local/share to /usr/local/share-temp. Leave /opt/X11 alone for X11 build, but change it to /opt/X11-temp for non-X11 build.

To build:

```
make clean
./configure --disable-compile-inits
make
```

Move gs to the "Binaries" folder on the machine used to build the actual install package.

If building for X11 support, add a final line

```
make so
```

Then go to the file sobin in the ghostscript-9.55 source folder. Move the library "libgs.dylib.9.55' to the "Binaries" folder inside the Ghostscript-9.55 folder. Move the two symbolic links to the same location if desired, although they will be ignored.

4. Preparing the Root Folder

Final preparation is done on the most recent system, currently Big Sur Arm.

Edit the file "buildPackage.sh" to reflect the current version of Ghostscript. Currently this file contains "9.55", referring to folders installed by Ghostscript. Change this number appropriately.

(The next step was modified for Ghostscript 9.55 from a more complicated step used earlier. Bruno Voisin provided a modified version of the Ghostscript Resource folder containing a folder named Font of symbolic links to TeX Live fonts, and a folder named Init with three files named Fontmap, Fontmap.cmr, and Fontmap.MacTeX. These last three files are stable from year to year, but the Font folder is updated each year. When we create a "root" folder which will eventually be installed, we copy this informations into the folder using the "buildPackage" script. To make all of this more transparent, we will retain Bruno's Init folder, but the Font folder will be named TeXfonts, created in the top level of the build directory, and copied into "root" by "buildPackage." We made this change by making a slight revision of "buildPackage" and a slight revision of "buildTeXfonts.sh".)

Next make sure the latest TeX Live is installed on this machine and type

```
sh buildTeXfonts.sh
```

This step will create a folder named "TeXfonts" in the main build directory, containing symbolic links to the pfb font files in this latest TeX Live distribution.

Next change directory to /usr/local and rename bin to bin-temp and and share to share-temp. Then switch to the ghostscript source folder and install Ghostscript using

```
make clean
./configure --disable-compile-inits
make
sudo make install
```

In this step we are not interested in the binaries, but instead in the support files for Ghostscript.

Then change to the main Ghostscript directory of the MacTeX build system and type

```
sudo sh buildPackage.sh
```

This step will copy Ghostscript support files from /usr/local to the root directory of MacTeX's Ghostscript build folder. It will add the binaries constructed earlier. Then it will make modifications from Arnold-Voisin so Ghostscript will understand TeX files.

5. Adjusting Files in the Ghostscript Install Package

This package contains some files which are shown to the user during installation. One of these files is a brief Welcome document, a second is a longer ReadMe file, and a third shows the License for the code. After the Ghostscript install package is constructed, you may need to revise these documents. For example, the Welcome document contains the release date. Usually TeX Live authors are overly ambitious and predict a release several months earlier than it actually occurs, and you'll need to change the release date from time to time.

None of these files appear in MacTeX, but they appear if you make an optional Ghostscript Install Package.

6. Signing, Timestamping, and Adopting Hardened Runtimes for Binaries

Sign and adopt a hardened runtime for everything using

```
sudo sh liposcriptsign=
```

This script signs gs-noX11 and gs-X11. Note that gs-X11 requires a third party library exception because it links with X11.

7. Building the Final Install Package

The install package is constructed using scripts in the folder "CreateInstallPackage." This folder also contains Resources and Scripts which will become part of the package. We first build Ghostscript-9.55-Start.pkg by

```
sudo sh pkgbuildscript.sh
```

This script reaches up a level to find the root folder containing the package material.

Note that the final Ghostscript package only requires Ghostscript-9.55-Start. So it makes sense to stop here.

However, if desired, create Ghostscript-9.55-Temp.pkg by running a second script

```
sudo sh productbuildscript.sh
```

Finally, created a signed package Ghostscript-9.55.pkg by running the script

```
sh signPackage.sh
```

This script contains my Apple signing identity; future builders will first have to join Apple's Developer Support and get their own signing identity.

The CreateInstallPackage folder contains two other files and one other script. The files are requirement.xml and distribution.xml, and the script is create_distribution.sh. In future years, it may be enough to edit distribution.xml by hand. This file contains the Ghostscript-9.55-Start.pkg name, and references to the background image, Welcome dialog, ReadMe dialog, and License dialog. It contains the identifier of the package, currently "org.tug.mactex.ghostscript9.55". And it contains the restriction that the package can only be installed on machines running system 10.14 or higher.

But if necessary, the file can be recreated as follows. The system requirement is contained in requirement.xml. Edit this first. Then run the script create_distribution.sh. This script provides some information in distribution.xml, and outputs a sample distribution.xml file. But other items must be added by directly editing the file.

8. Notarizing the Final Install Package

Scripts to notarize Ghostscript-9.55.pkg are in the folder "NotarizePackage" inside "CreateInstallPackage." The actual Ghostscript-9.55.pkg created above can be left where it is; it need not be moved to the new folder.

To notarize,

```
sh sendnotarizerequest.sh
```

There will be a delay when nothing is printed in Terminal as the package is uploaded to Apple. When the upload is complete, a message of the form

```
RequestUUID = 96f0932f-b50f-4d73-ad65-1220fd9c8efe
```

will be printed. This magic number can be used to discover what went wrong if notarization fails, so I recommend copying it to the file requestUUID.tex. There will be a pause of about ten minutes and then a notification from Apple and an email from Apple will arrive. If all is well, this message says that notarization succeeded. To add the certificate to the install package

```
sh stapleresult.sh
```

The package Ghostscript-9.55.pkg is then ready for uploading and release.

In case of failure, open the script detailedinfo.sh and notice that the magic UUID number is part of this script. Copy and paste the new number from requestUUID.tex to this script. Then run it.

```
sh detailedinfo.sh
```

This command will return a web url. Use Safari to go to that url, where a very detailed error report will clearly state what went wrong.

The detailedinfo script runs a command named xcrun which communicates with developer support. Apple has adopted two-factor authentication security for such communication, but this security is not very convenient when running shell scripts. So developers can log into their Apple developer account and assign a password to xcrun and a few similar commands. Then when the script runs, typing this password is enough to run the command. Note that the detailedinfo script contains my account name and password. When someone else makes these packages, they will replace this information with their own personal information.