Opera Build and Distribution Process
May 2014

This document describes the process that the build master goes through to integrate software from Opera teammates through to the ultimate distribution on SourceForge and the local CFHT server.

As Opera team members develop Opera Modules and libraries, these are from time to time integrated in to the master source tree and distributed. The *Build Master* is responsible for this process. Generally the modified files are sent via email or shared disk in a folder. A small tool is available to quickly perform the diffs:

./scripts/operadiff ~/Desktop/updates

This tool performs diffs and informs the Build Master of the changes. The Build Master then reviews all the changes and integrates them in to the master source, asking questions of the developer should there be suspicious looking changes.

After integrating all the changes, the Build Master then does a complete build to ensure that all modules and libraries build without compile errors, or warnings, or linker problems.

If a new library or tool or core module is added, then the Build Master must edit the Makefile.am in the appropriate directory to add the new software.

In the case of a new module, the module name should be added to bin_PROGRAMS:

```
# this lists the binaries to produce
bin_PROGRAMS = operaSNR operaWavelengthCalibration \
        operaMasterFlat operaPolar operaBias operaExtraction \
        operaGain operaReductionSet operaGeometryCalibration \
        operaMasterComparison operaMasterBias \
        operaMasterFabPerot operaInstrumentProfileCalibration operaPixelSensitivityMap \
        operaNormalize operaCreateProduct operaExtractSpectralLines
operaExtractionApertureCalibration \
        operaStarPlusSky operaStarOnly operaPolarIntensity operaFluxCalibration \
        operaCreateFluxCalibration operaTelluricWavelengthCorrection
operaBarycentricWavelengthCorrection \
        operaOrderSpacingCalibration operaEchelleDispersionCalibration operaPolarimetryCorrection
\
        operaCalculateSpectralResolution operaMasterFluxCalibration \
        operaObjectInTheSkySetup operaObservingConditionsSetup operaSpectrographSetup \
        operaTelescopeSetup operaNormalizeAcrossOrders operaCreateFlatFieldFluxCalibration \
        operaTelluricCorrection operaStitchOrders operaGenerateLEFormats
```

The module would be added to the end of the list. Then the build rule would be added:

```
operaGenerateLEFormats_SOURCES = operaGenerateLEFormats.cpp operaGenerateLEFormats.h
```

as an example.

In the case of a library, similar steps are followed, except there are two entries, one for static libraries and one for dynamic:

```
lib_LIBRARIES = liboperaFFT.a

lib_LTLIBRARIES = liboperaException.la libPolynomial.la liboperaSpectralOrderVector.la
liboperaSpectralElements.la liboperaSpectralOrder.la liboperaInstrumentProfile.la \
        liboperaGeometry.la liboperaWavelength.la liboperaCCD.la liboperaFITSSubImage.la
liboperaEspadonsSubImage.la liboperaImage.la liboperaStats.la \
        liboperaMath.la liboperaLMFit.la liboperaFit.la liboperaConfigurationAccess.la
liboperaParameterAccess.la liboperaFITSImage.la liboperaPolarimetry.la \
        liboperaEspadonsImage.la liboperaFITSProduct.la liboperaMEFFITSProduct.la liboperaJD.la
liboperaHelio.la liboperaMatrix.la liboperaSpectralLines.la liboperaLib.la \
        libGaussian.la liboperaSpectralFeature.la liboperaGeometricShapes.la
liboperaExtractionAperture.la liboperaFluxVector.la libGainBiasNoise.la \
        liboperaMultiExtensionFITSImage.la liboperaMultiExtensionFITSCube.la liboperaFITSCube.la
liboperaWIRCamImage.la liboperaMuellerMatrix.la \
        liboperaImageVector.la liboperaStokesVector.la libPixelSet.la
liboperaSpectrumSimulation.la liboperaSpectralEnergyDistribution.la \
        libgzstream.la libLaurentPolynomial.la liboperaSpectralTools.la \
        liboperaObservingConditions.la liboperaObjectInTheSky.la liboperaSpectrograph.la
liboperaTelescope.la liboperaInstrumentEnvironmentSetup.la
```

as an example, here lib_LIBRARIES are static and lib_LTLIBRARIES are dynamic.

Then after the library name is added, the build rule is added, for example:

```
liboperaInstrument_la_SOURCES = operaInstrument.cpp operaInstrument.h
liboperaInstrument_la_LDFLAGS = -version-info 1:0:0
```

Next, the library name has to be added to the list of libraries in each of the test, tools, and core-espadons Makefile.am files. The list is rather long, and looks like this:

```
AM_LDFLAGS = -loperaInstrumentEnvironmentSetup -loperaObservingConditions -loperaObjectInTheSky -
loperaSpectrograph -loperaTelescope -loperaSpectralOrderVector -loperaSpectralOrder -
loperaSpectralElements -loperaSpectralLines -loperaSpectralFeature -loperaExtractionAperture -
loperaSpectralTools -loperaFluxVector -lGaussian -loperaInstrumentProfile -loperaGeometry -
loperaWavelength -lPolynomial -lLaurentPolynomial -loperaMath -loperaJD -loperaHelio -loperaCCD -
loperaFit -loperaImage -loperaLMFit -loperaConfigurationAccess -loperaParameterAccess -
loperaFITSSubImage -loperaEspadonsSubImage -loperaFITSProduct -loperaMEFFITSProduct -
loperaFITSImage -loperaEspadonsImage -loperaWIRCamImage -loperaMultiExtensionFITSCube -
loperaMultiExtensionFITSImage -loperaFITSCube -loperaPolarimetry -loperaException -lGainBiasNoise
-loperaFFT -loperaMuellerMatrix -loperaStokesVector -loperaImageVector -loperaGeometricShapes -
loperaMatrix -loperaSpectralEnergyDistribution -loperaSpectrumSimulation -lPixelSet -loperaStats
-loperaLib -lfftw3 -lgzstream -lcfitsio -lz -lsofa_c -lpthread -lm
# This is for Linux...
LIBS = -loperaInstrumentEnvironmentSetup -loperaObservingConditions -loperaObjectInTheSky -
loperaSpectrograph -loperaTelescope -loperaSpectralOrderVector -loperaSpectralOrder -
loperaSpectralElements -loperaSpectralLines -loperaSpectralFeature -loperaExtractionAperture -
loperaSpectralTools -loperaFluxVector -lGaussian -loperaInstrumentProfile -loperaGeometry -
loperaWavelength -loperaEspadonsSubImage -loperaFITSProduct -loperaMEFFITSProduct -
loperaFITSImage -loperaEspadonsImage -loperaWIRCamImage -loperaMultiExtensionFITSCube -
loperaMultiExtensionFITSImage -loperaFITSCube -loperaPolarimetry -loperaException -lGainBiasNoise
-loperaFFT -loperaMuellerMatrix -loperaStokesVector -loperaFluxVector -loperaImageVector -
loperaFITSSubImage -loperaImageVector -loperaGeometricShapes -loperaMatrix -lPolynomial -
lLaurentPolynomial -loperaMath -loperaJD -loperaHelio -loperaCCD -loperaFit -loperaImage -
loperaLMFit -loperaConfigurationAccess -loperaParameterAccess -lPixelSet -
loperaSpectralEnergyDistribution -loperaSpectrumSimulation -loperaStats -loperaLib -lfftw3 -
lgzstream -lcfitsio -lsofa_c -lz -lpthread -lm
```

There is one entry for Linux and one for Mac. The Linux entry must be in order of dependency.

If a new module is added, then most likely a new rule will need to be created to execute the module. The Makefiles are in harness/espadons. There are 7 Makefiles:

```
Makefile.analysis                    - for rules outside the scope of espadons
Makefile.calibration                 - calibration step rules
Makefile.configuration               - configuration such as path names
Makefile.core                        - reduction modules
Makefile.parameters                  - constant parameters for the modules
Makefile.upena                       - upena-compatibility rules
Makefile.util                        - general handy utility rules
```

In general, team members would only add calibration modules or reduction modules. A rule looks like this:

```
################################################################################################
# Telluric Wavelength Correction
################################################################################################
%i.tell$(gzip): %.e$(gzip)
        @start=$$SECONDS; \
        if [ ! -e $(calibrationdir)$@ ] ; then \
                $(bindir)operatrace $(TRACE) $(errfile) $(MACHINE) "$
(bindir)operaTelluricWavelengthCorrection \
--inputObjectSpectrum=$(spectradir)$*.e$(gzip) \
--inputWaveFile=$(calibrationdir)$(QUALIFIERS).wcal$(gzip) \
--telluric_lines=$(configdir)$(telluric_atlas_lines) \
--telluric_spectrum=$(configdir)$(telluric_reference_spectrum) \
--spectralResolution=$(telluric_spectralResolution) \
--initialWavelengthRange=$(telluric_initialWavelengthRange) \
--initialWavelengthStep=$(telluric_initialWavelengthStep) \
--XCorrelationThreshold=$(telluric_XCorrelationThreshold) \
--subtractCentralWavelength=$(telluric_subtractCentralWavelength) \
--normalizationBinsize=$(telluric_normalizationbinsize) \
--sigmaThreshold=$(telluric_sigmathreshold) \
$${pargs} \
--outputWaveFile=$(calibrationdir)$@ $(optargs)" 2>&1 | tee -a $(logdir)$*.log ; \
                echo "$(pref) Telluric Wavelength Correction $@ created in $(deltat)" ; \
        fi
```

Piece by piece,

```
%i.tell$(gzip):               - the target (what to make)

%.e$(gzip)                    - the dependency (what this rule needs in order to work)

$(bindir)operatrace $(TRACE) $(errfile) $(MACHINE) “$(bindir)operaTelluricWavelengthCorrectin…
                              - execute the rule, note the trace is used to print out the
                              rule as it is executed.
```

The rest of the entries are arguments or parameters the module needs in order to work. Once this is done and the build succeeds, then the Build Master should do a trial run to make sure that the calibration and reduction performs without error.

Next is distribution. There is a script that handles this:

```
cd ~
./opera-1.0/scripts/operainstallweb [—nodoc —noemail]
```

This script cleans the distribution of binaries, creates a zip file and a tar file and copies them along with the READMEs to SourceForge and /data/world/doug/.

—nodoc means do not recreate the doxygen documentation and —noemail means don't send an email announcing the distribution to team members.

The pathname to transfer files via scp to SourceForge is in the script. The password is CFHTelescope.