An introduction course to the ELT Development Environment

Sylvie, Carlos & Federico

05/07/2018

ELT webpage, VM installation and repositories

CGU, July 2018

The ELT WEB page

- https://www.eso.org/~eeltmgr/
  - ELT DevEnv RELEASE NOTES
  - ELT Linux Installation Guide
  - ELT SVN Usage Guidelines
  - Guide to Developing Software for the ELT

The Linux Installation

- https://www.eso.org/~eeltmgr/
  - Minimum HW required:
    - 2x CPUs
    - 4GB RAM
    - 100GB disk
    - 1x NIC
  - Standard CentOS-7-x86-x86_64-DVD-1708.iso
    - https://www.centos.org/download/
    - ftp://ftp.eso.org/pub/eelt/
  - VirtualBox
    - https://www.virtualbox.org

The SVN repositories

- Repository for prototyping:
  - http://svnhq9.hq.eso.org/p9/
- Official repository for integration
  - http://svnhq8.hq.eso.org/p8/
- Eventually to GIT

ELT Jira projects and the User Dev Environment

SFE, July 2018

JIRA Projects

EELTMGR

- Ticketing system for current ELT DevEnv release
- For tasks of max. 1 day (Helpdesk style)
- Accept email for ticket creation (eeltmgr@eso.org)

ELTDEV

- Stories: new requirements to the ELT DevEnv
- Bugs: report problem in previous delivery
- Dashboard

ELT DevEnv on Linux

- Hostname convention: elt<**PROJ**><**NN**>
  - PROJ = dev, jen, int, cii….
  - eltint20, eltdev26
- Accounts: eeltdev, eeltmgr, NIS (dev)
- Common RPM : eelt-common*
  - Installed under /eelt/
  - Default pc files
  - Common libraries
  - Basic tools: getTemplate, lmod, too, elt-devenv..

LMOD: ENVIRONMENTAL MODULES SYSTEM

- LMOD: lua based Environment Modules system
  - provide a convenient way to dynamically change the users' environment through modulefiles.
- Basic commands:

```
$ module list
$ module avail
$ module load package1 package2 …
$ module unload package1 package2 …
$ module help [package]
$ module save [collection]
$ module restore [collection/system]
$ module show package
```

EELT getTemplate

- Based on Cookiecutter :
  - A command-line utility that creates projects from **cookiecutters** (project templates)
- PRJ_TEMPLATES=/eelt/2.1.1/templates
- Python script getTemplate.py

```
$ ./getTemplate.py -d introot $INTROOT

$ ./getTemplate.py -f c++-main myFile
```

A very gentle introduction to the ELT Build System and Jenkins CI services

FPE, July 2018

The waf framework

- waf framework: https://waf.io/
  - Why not make / cmake / ant / maven / gradle / ….
    - Need to support many languages for ELT project and desire to have only one build system
  - Python based
    - Tools / Extras can be added using Python language
  - Focuses on expandability
    - Need for multiple language support and multiple tools support
  - Focuses on performance
    - Rebuilds are not based just on timestamp but on hashes of intermediate generated artifacts; parallel by design
  - Dependency management
  - Logging and debugging support

waf

- wscript: build scripts defining configuration, options and build steps
  - Python code
  - Interaction with the waf framework
- Command line execution of phases
  - configure
  - build
  - test
  - install / dist
  - Custom commands

waf: an example

```
def options(opt):
        opt.load('compiler_cxx python pyqt5 ')
def configure(conf):
        conf.load('compiler_cxx python pyqt5 ’)
        conf.check(header_name='stdio.h', features='cxx’)
        conf.check_python_version((3,5,0))
def build(bld):
        bld.shlib(source='a.cpp inc/a.h', target='alib’, export_includes=‘inc’)
        bld.program(source='m.cpp', target='app', use='alib’)
        bld.stlib(source='b.cpp', target='foo’)
        bld(features="py pyqt5",  source="src/test.py src/gui.ui",
                install_path="${PREFIX}/play/", install_from="src/")
```

wtools

- wscripts are readable and easy but still…
- wtools as a layer for:
  - Simplification for common tasks for users
  - Centralized maintenance and roll-out of new features
  - Easier to enforce certain practices
- Can reduce script to a single line:

```
from wtools.module import declare_cprogram

declare_cprogram(target="foo", use="bar")
```

- Tasks for primary artifacts and additional ones are created: tests, installation, linting …

wtools

- Based on set on conventions:
  - Directory structure, file positioning, file naming
- Currently supporting:
  - C/C++ program, shared and static library,
  - Python program and package,
  - Qt5 C++ or Python program
  - Java JAR packages.
  - Protoc/RTI DDS C++ compilation
- Custom modules that leverage full waf can be created for specific needs not included in wtools

Directory structure

<package>/<module>/src # Any source files (incl, headers, mocs etc.)

/src/include/<a> # public include files in case of C/C++

/resource # Resources

config/ - contains default configuration files

audio/ - contains sounds, music and other audible files

image/ - contains images and other visual artefacts

model/ - contains models

dictionary/ - contains dictionaries

data/ - contains runtime configuration

/interface # Interface specifications (IDL, mockups?)

/doc # non-generated documentation

/test # unit tests

wscript # build script

Sample session

- Online generated documentation:
  http://eeltdev8.hq.eso.org:8080/job/WTools-doc/lastSuccessfulBuild/artifact/html/index.html
- Sample project:
  http://svnhq9.hq.eso.org/p9/trunk/EELT/ICS/Prototypes/wtools/test/project
- Top level wscript:
  - declare_project
  - recurse
  - requires
    - cxx, python, java, qt5, pyqt5, boost, rtidds, protoc,
- Package level wscript:
  - declare_package
  - recurse

Sample session

- Module level wscript:

declare_cprogram	C/C++ program
declare_cshlib	C/C++ shared library
declare_cstlib	C/C++ static library
declare_qt5cprogram	Qt5 C/C++ program
declare_qt5cshlib	Qt5 C++ library
declare_cprotobuf	C++ protobuf shared (default) or static library
declare_crtidds	C++ RTI-DDS shared (default) or static library

Sample session

- Module level wscript:

declare_jar	Java Archive (jar) library
declare_config	Configuration only module
declare_pyprogram	Python program
declare_pypackage	Python package
declare_pyqt5program	Python program with Qt5 usage
declare_pyqt5package	Python package with Qt5 usage
declare_custom	Custom module (build function must be defined by hand)

Sample session

- Artefact name
  - target
- Dependencies
  - The *use* keyword
  - External dependencies and pkg-config
- Attributes customization
  - includes
  - export_includes
  - cflags / cxxflags

Sample session

- waf
  - --help
  - -v (-vv, -vvv)
  - --zone=xxx
- waf configure
  - --mode=[release|debug|coverage]
  - --pyqt5-pyside2
  - --boost-includes , --boost-libs
- waf list
- waf build
  - -j X

Sample session

- waf test
  - --alltests
  - Test output and output files generated
- waf install
  - PREFIX environment variable (during configure!)
  - Remember: PATH, LD_LIBRARY_PATH, PYTHONPATH, CLASSPATH
- waf eclipse
- waf docs
- waf clean / distclean

Remember

- waf
  - --help
  - -v (-vv, -vvv)

Jenkins for ELT

- http://eeltdev8.hq.eso.org:8080/
- Most of jobs based on “new” concept of pipelines
  - Some plugins (ie. cppcheck) do not support pipelines
  - Some pipelines are split for readability’s sake
  - Definition using Groovy and Pipelines language
    - In the near future may migrate to Jenkinsfile
- Default execution “on commit”
- Current machines setup
  - 1 high speed blade with no RT
  - 1 good hardware with RT
  - To be soon expanded?

Jenkins for ELT

- Standard operations
  - Configure
  - Build
  - Unit test (with or without coverage)
  - Integration test (with or without coverage)
  - Code linting
  - Code checking
  - Documentation
  - Artifact saving
  - Email notification
    - To committer that broke the build
    - To fixed recipient