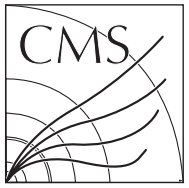
Available on CMS information server

CMS CR -2009/112



The Compact Muon Solenoid Experiment

Conference Report

Mailing address: CMS CERN, CH-1211 GENEVA 23, Switzerland

16 May 2009

Usage of the Python Programming Language on the CMS Experiment

R. Wilkinson, B. Hegner

Abstract

Being a highly dynamic language and allowing reliable programming with quick turnarounds, Python is a widely used programming language in CMS. Most of the tools used in workflow management and the GRID interface tools are written in this language. Also most of the tools used in the context of release management: integration builds, release building and deploying, as well as performance measurements are in Python. With an interface to the CMS data formats, rapid prototyping of analyses and debugging is an additional use case. Finally in 2008 the CMS experiment switched to using Python as its configuration language. This talk will give an overview of the general usage of Python in the CMS experiment and discuss which features of the language make it well-suited for the existing use cases.

Presented at *Computing in High Energy and Nuclear Physics,21-27 Mar, 2009,Prague,Czech Republic,15/05/2009*

Usage of the Python Programming Language in the CMS Experiment

R. Wilkinson[1], B. Hegner[2]

1. California Institute of Technology, Pasadena, CA, USA; rickw@caltech.edu
2. CERN, Geneva, Switzerland; Benedikt.Hegner@cern.ch

Abstract. Being a highly dynamic language and allowing reliable programming with quick turnarounds, Python is a widely used programming language in CMS. Most of the tools used in workflow management and the GRID interface tools are written in this language. Also most of the tools used in the context of release management: integration builds, release building and deploying, as well as performance measurements are in Python. With an interface to the CMS data formats, rapid prototyping of analyses and debugging is an additional use case. Finally in 2008 the CMS experiment switched to using Python as its configuration language. This talk will give an overview of the general usage of Python in the CMS experiment and discuss which features of the language make it well-suited for the existing use cases.

1. Introduction

Many software projects on Compact Muon Solenoid experiment (CMS) have independently chosen to use Python[1], not as a result of a top-down decision from management. When asked why they use Python, some common reasons emerge. Python is seen to be easy to learn, without the steep learning curve of C++. Python syntax is seen as simpler and more comprehensible than either C++ or Perl, which makes it easier to understand code written by others. This simplicity leads to the option of writing prototype code in Python, and then, once objects and behaviours are determined, translating to C++ for performance. Finally, many standard tools are native to the language, and many useful external packages exist, such as cherrypy[2] for web programming, PyROOT[3] for physics analysis, and PyQt for graphics.

2. Job Configuration

CMS jobs are defined by configuration files. We use a single executable, “cmsRun”, which loads modules and runs them as defined by the configuration file. A software release contains over 6000 configuration files. Three quarters of these are fragments, meant to be shared, defining the parameters for a single module or sequence of modules. The rest are full, executable configurations. Our standard full-chain release validation job defines over 700 modules, over 150 sequences of modules, and over 13,000 configurable parameters.

These configurations had been implemented using a custom language syntax, parsed by Flex and Bison[5]. This system was designed to be a simple declarative language, but it soon proved too inflexible. Users commonly wanted to copy and modify sets of parameters and modules, which meant that we had to provide that syntax for each of the many elements of the language. The production

system especially needed easy access to modify input and output file names and random number seeds. We needed a full programming language, and since the production system was already written in Python, it was the natural choice.

To design the language syntax, we tried to mimic the original language as much as possible. The new Python configuration returns a single Python data structure. This structure was translated to the framework's internal C++ data structure using a boost::python[6] interface. Note that unlike the Python frameworks used by some other experiments, our interface is not interactive. The configuration is completely defined by the configuration file. This decision was made to facilitate provenance tracking[7].

After the transition, which is explained in more detail elsewhere in these proceedings[8], the maintenance burden was indeed reduced. In addition, the new system had powerful new features. The configurations became easier to debug, because users could inspect the configurations interactively, and could test their configuration syntax simply by compiling it. It became easier to build configurations, and to keep the configurations internally consistent by, for example, changing input and output file names simultaneously. We no longer need separate perl or shell scripts to edit the configurations. Users can now use command-line arguments, and higher-level python functions such as loops. Finally, a browsing GUI was created, which uses PyQt[9].

We also have configuration builder utilities, written in Python, to assemble common uses, such as simulation and reconstruction chains, and to add common sets options to them, such as those needed for fast simulation or cosmic ray reconstruction. This configuration builder is used for release validation, and to create standard workflows for production.

3. Analysis

CMS stores its data in ROOT files. There are two common methods to access this data[10]. The most common is to use the standard framework, creating a C++ "EDAnalyzer" module to read the data, and perhaps create a new ROOT file, extracting selected data. Another way is to use "FWLite", a utility which provides easy access to the objects in the event, and their C++ member functions, in either C++ or Python. FWLite auto-loads the C++ objects it needs, with class dictionaries provided by ROOT's REFLEX utility.

FWLite can be combined with PyROOT to create a powerful and intuitive analysis tool. The following code can be used to create analysis plots.

```
from PhysicsTools.PythonAnalysis import *
from ROOT import *
# prepare the FWLite autoloading mechanism
gSystem.Load("libFWCoreFWLite.so")
AutoLibraryLoader.enable()
events = EventTree("reco.root")
# book a histogram
histo = TH1F("photon_pt", "Pt of photons", 100, 0, 300)
# event loop
for event in events:
    photons = event.photons  # uses aliases
    print "# of photons in event %i: %i" % (event,len(photons))
    for photon in photons:
        if photon.eta() < 2:
            histo.Fill(photon.pt())
```

The above code produces histogram windows directly from a Python prompt, with a simplicity approaching pseudocode. To further simplify the use of Python interactively, the user could save the first four lines of setup code into a new script, and use that script to start an interactive Python session:

```
> python –i openRecoFile.py
```

4. Production

The CMS Data Management and Workflow Management team uses python extensively in data processing. Clusters of python daemons perform tasks such as request management, allocation, job submission, tracking, bookkeeping, and error handling[11]. These daemons use a common Python framework for event-driven message passing and MySQL persistency.

In this architecture, the processing work is done by "ProdAgents", independent daemons which act as front ends to diverse resources, ranging from farms to grid systems. These ProdAgents receive tasks from a "ProdMgr", which manages requests, and does the final accounting.

In addition, many web applications use cherrypy servers, for such tasks as data moving[12], site itoring[13], data monitoring[14], and database browsing[15]. These tools are being consolidated into a single framework[16].

5. Conclusion

CMS uses python for a wide variety of applications, including scripting, job configuration, analysis, GUIs, web interfaces, message passing, and database interfaces. We anticipate that Python's popularity on the experiment will continue to grow, as more shared utilities and frameworks in Python become available.

References

[1] http://www.python.org
[2] http://www.cherrypy.org
[3] http://root.cern.ch/root/HowtoPyROOT.html
[4] http://www.riverbankcomputing.co.uk/software/pyqt/intro
[5] http://flex.sourceforge.net, http://www.gnu.org/software/bison
[6] http://www.boost.org/doc/libs/release/libs/python/doc/
[7] C. D. Jones, "File Level Provenance Tracking in CMS", these proceedings
[8] R. Wilkinson, "Using Python for Job Configuration in CMS", these proceedings
[9] A. Hinzmann, "Visualization of the CMS Python Configuration System", these proceedings
[10] C.. D. Jones, L. Lista, and B. Hegner, "Analysis Environments for CMS", Journal of Physics Conference Series **119** (2008) 032027
[11] S. Wakefield *et al.*, "CMS Production and Processing System – Design and Experiences", these proceedings
[12] V. Kuznetsov, "CMS FileMover: One Click Data", these proceedings
[13] S. Metson, "SiteDB: Marshalling the People and Resources Available to CMS", these proceedings
[14] L. Tuura, "Authentication and Authorization in CMS' Monitoring and Computing Web Services", these proceedings
[15] A. Pierro, "CMS Conditions Database Web Application Service", these proceedings
[16] S. Wakefield *et al.*, "Job Life Cycle Management Libraries for CMS Workflow Management Projects", these proceedings