CREATING TOOLS TO ASSIST IN DEVELOPMENT OF CMS SOFTWARE

G. Adamov [1,2,a]

[1] *Laboratory of Information Technologies, Joint Institute for Nuclear Research, 6 Joliot-Curie, Dubna, Moscow region, 141980, Russia*

[2] *Institute of Quantum Physics and Engineering Technologies, Georgian Technical University, 77 Kostava Str., Tbilisi, 0175, Georgia*

E-mail: [a]adamov@jinr.ru

Software packages, created for the modern physics experiments, present a sets intertwined code structures, written by different people and bundled together to perform range of different reconstruction and analysis tasks on the experimental data. Sometimes due to complicated nature of such frameworks a new set of tools is required to simplify their further development. In this work we investigate an example of such tool, created for the CMS experiment to analyse the structure of its software components.

Keywords: python, refactoring, CMSSW, source control

1. CMS Software Overview

CMS Software (CMSSW) is an overall collection of software for the reconstruction and analysis of data taken from the CMS detector. Following is a general description of the framework interpreted by author, the CMSSW and all of its parts are best described in [1].

A tool described here is based on the main idea of the CMSSW offline analysis process – a system of workflows. Without going deep into details it is basically predefined configurations and instructions given to the CMSSSW to handle different types of analysis tasks performed with data collected by CMS experiment. There are a lot of different parts and modules for various tasks inside the framework, but for the workflow system to function properly the most important of them are the following three.

cmsRun is a main executable of CMSSW event processing model, it manages the execution of the framework modules which contain all the code for calibration, reconstruction, simulation etc. It is configured at runtime by user's job specific configuration file.

cmsDriver is a tool to create production-solid configuration files from minimal command line options.

runTheMatrix.py contains standard workflows (series of cmsDriver commands) for each release. It is used for running a sequence of specific modules for the specific set of tasks based on the workflow description.

Figure 1. A rough schema of CMSSW workflow dependencies

```
\_ runTheMatrix.py -l 1306.0,5.1
   \_ python /afs/cern.ch/user/.../CMSSW_10_1_0_pre1/bin/slc6_amd64_gcc630/runTheMatrix.py -l 1306.0,5.1
   |   \_ /bin/sh -c cd 5.1_TTbar+TTbarFS+HARVESTFS; cmsDriver.py TTbar_8TeV_TuneCUETP8M1_cfi  --conditions auto:run1_mc --fast ...
   |   |   \_ cmsRun TTbar_8TeV_TuneCUETP8M1_cfi_GEN_SIM_RECOBEFMIX_DIGI_L1_DIGI2RAW_L1Reco_RECO_EI_VALIDATION_DQM.py
   |   \_ /bin/sh -c cd 1306.0_SingleMuPt1_UP15+SingleMuPt1_UP15+DIGIUP15+RECOUP15+HARVESTUP15; cmsDriver.py step3 ...
   |       \_ cmsRun step3_RAW2DIGI_L1Reco_RECO_EI_PAT_VALIDATION_DQM.py
   \_ python libmon.py -c runTheMatrix.py -l 1306.0,5.1
```

Figure 2. Example of the process tree running two workflows

The simplified diagram of the workflow subset structure, shown in Figure , describes the relation between workflow and the parts used in it.

Workflow in this case is a predefined configuration, with specific unique id and name, describing the way and execution order of the set of configuration files needed for specific task.

Step file is a configuration file run during the workflow execution and is responsible for configuring the specific parameter and handling of the files, usually root files, needed for the workflow. Step file also defines libraries that had to be used during analysis and parameters given to them.

Library is a module compiled into a single loadable file by a CSMSW build tools and is included during runtime to perform a specific task.

Module is a set of configuration/analysis code that handles the specific analysis task. It defines the data that had to be used, describes the tasks that had to be performed on this data and the output to be produced as the result of processing this information (plots, datasets, etc.). It consists of the source files written both in Python and C, which basically tell CMSSW what data to process and how to process it.

More details on CMS software are expounded in [2].

2. Description of the task

To make the maintenance of the workflow structure easier and simplify the modules development, the idea of a help tool was introduced by the CMS software group management.

This tool had to list all libraries used during each step of the workflow process, categorize and save information about them. This should potentially allow the developer to get throughout information about workflow, determine what modules and what source code is used in it, quickly understand which steps it consists of. Also it had to allow the reverse search from source, for example any part of the module source code is changed with the workflow information and the source map (which is generated by separate tool) developers can quickly understand what workflows will be affected by it.

It also has to be portable enough to run in the shared cluster environment such as lxplus at CERN, without installation of any additional libraries.

As one of the main priorities it was also requested to not interfere with the main software packages in any way, basically to separate this new script from the CMSSW core components so it can be easily adapted for different releases and not be dependent on them.

Also since multiple workflows can run in parallel at the same time, it is required that script could simultaneously capture all the information of the running processes.

To satisfy the following request a tool was developed, written in Python language, since it is widely used by the development team. This tool runs in parallel with the main workflow process, monitors it and extracts the list of step configuration files and libraries loaded for the analysis purposes. It then compares the libraries with pre-made list of source to module name mapping and writes all the results to the database as well as generating a JSON file with needed information. From this point developers can take a quick look at the JSON file or access a database through the CLI (Command Line Interface) tool to view detailed information about the given workflow.

3. Detailed overview of the program

The main challenge for the program is to catch all the libraries used in all the workflow configurations ran at the same time. To do that this program wraps around runTheMatrix.py and runs it as a subprocess, this way it can see and continuously monitor all the child cmsRun processes, responsible for running the step configurations (Figure , shows the dependency tree created during run). It then takes the PID of each cmsRun process and reads the map file located at /proc/<PID>/map from the file system. These files describe a region of contiguous virtual memory in a process and contain information about libraries loaded during the process execution. This method was chosen since it gives the most reliable and easy to parse information about libraries loaded from CMSSW core. Script then sorts this file contents and selects the desired libraries, which are distinguished from system libraries by a path pointing to framework location (Figure 3).

```
7f254dfe0    /cvmfs/cms.cern.ch/slc6_amd64_gcc530/cms/cmssw/CMSSW_8_1_0_pre15/lib/slc6_amd64_gcc530/libCondFormatsGeometryObjects.so
7f254e072    /cvmfs/cms.cern.ch/slc6_amd64_gcc530/cms/cmssw/CMSSW_8_1_0_pre15/lib/slc6_amd64_gcc530/libCondFormatsAlignment.so
7f254e0fe    /cvmfs/cms.cern.ch/slc6_amd64_gcc530/cms/cmssw/CMSSW_8_1_0_pre15/lib/slc6_amd64_gcc530/pluginCSCGeometryESModule.so
7f254e140    /cvmfs/cms.cern.ch/slc6_amd64_gcc530/cms/cmssw/CMSSW_8_1_0_pre15/lib/slc6_amd64_gcc530/libDataFormatsGeometrySurface.so
7f254e14c    /cvmfs/cms.cern.ch/slc6_amd64_gcc530/cms/cmssw/CMSSW_8_1_0_pre15/lib/slc6_amd64_gcc530/libGeometryCSCGeometry.so
7f254e164    /cvmfs/cms.cern.ch/slc6_amd64_gcc530/external/boost/1.57.0-giojec2/lib/libboost_serialization.so.1.57.0
7f254e1fa    /cvmfs/cms.cern.ch/slc6_amd64_gcc530/cms/coral/CORAL_2_3_21-giojec10/slc6_amd64_gcc530/lib/liblcg_CoralBase.so
```

Figure 3. Libraries extracted from man file

After the workflow execution is finished all the collected results are written into the JSON file and SQLite database. (Example can be seen in Figure 4).

The JSON output files contain information about the workflow id, step files and the libraries that were used in them. SQLite database in addition has separate table to connect library names with the module source files it was compiled from. This table is parsed from the list of source to module name relations generated separately by different tool for each major framework release.

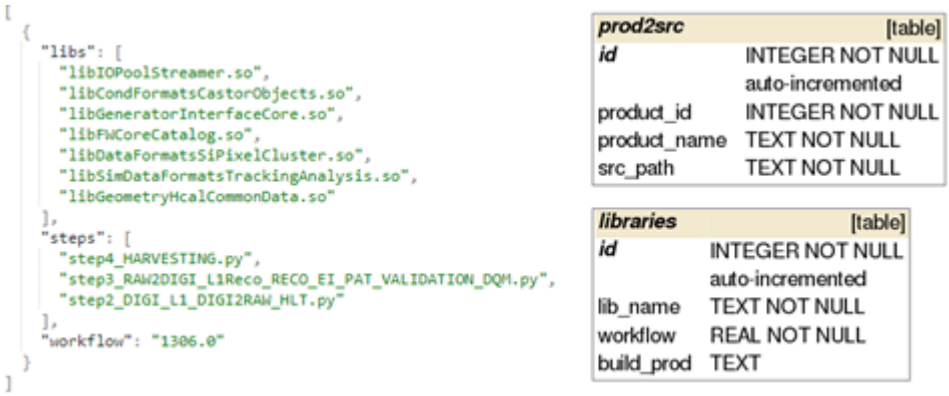

Figure 4. Examples of the produced JSON file and the SQLite database structure

The results then can be searched with the CLI tool integrated into the main executable of the script. With this tool developer can see the dependency between each step configuration in workflow and source files, used to compile libraries loaded in those steps.

CLI tool has two ways of outputting the results: as a formatted text into a terminal or a JSON file with requested information and search parameters.

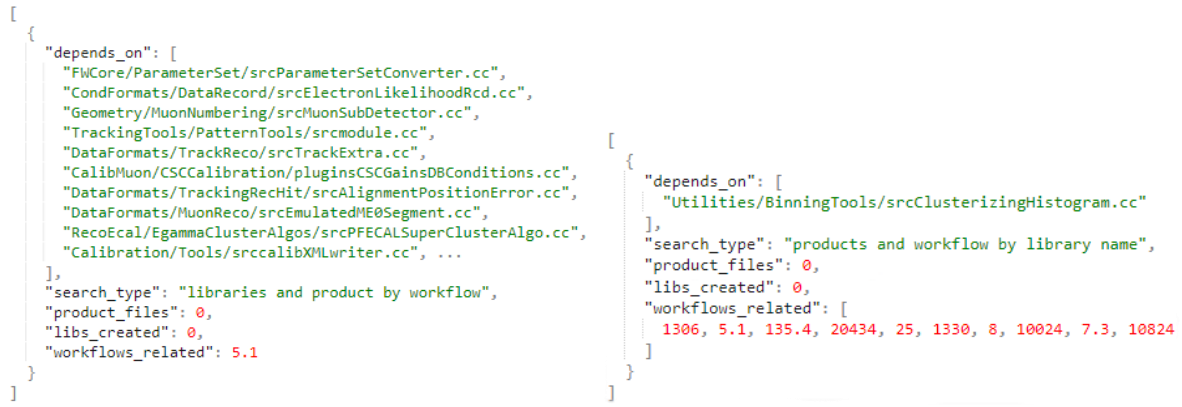

Figure 5. Examples of the search result in a form of JSON

There are several search types available: search for source files by library name, search for libraries build from given module name, list of libraries used in specified workflow, list of source files used in specified workflow. For each search a custom JSON structure is created. (Figure , shows examples of such search). Results of the search in the chosen format are then presented to the developer for further overview.

4. Future possibilities

Currently the described tool is used internally by a handful of developers as an additional option for codebase discovery and considered to satisfy all initial requirements.

Still it is possible to improve the existing tool. For example by creating a web interface for it, which will allow to run and search the workflows, as well as execute code search, through the more easy and understandable web page.

Another thing that can be improved is a database, which could be made more persistent and centralized to store all the information about current release as well as give an ability to see changes made to the workflows between the release versions. It will also open an opportunity to add the statistics of the workflows and files usage inside the framework.

This change however will require a dedicated server space and as a consequence will cause less portability of the script, so it is debatable if such an outcome worth all the effort required. That's why at the moment it is decided to leave all as it is, collect more user response and request from developers and then decide what further steps to take.

5. Conclusion

This work, while not offering something groundbreaking or innovative, shows how small tools like the one described can help in the development of much larger sophisticated software tools such as CMS Software framework.

Such tools can ease the life of developers and present an additional option for code refactoring and code discovery in projects with large codebase. This in return gives an ability to spend less time searching for the right parts of code and instead quickly change it, knowing exactly what parts of the framework will be affected. It also allows new developers to easily navigate big project, know consequence of each action and thus greatly increase projects development speed which is a crucial for effective development of any large system.

References

[1] The CMSSW Documentation Suite, The CMS Offline Workbook. Official CERN Twiki, https://twiki.cern.ch/twiki/bin/view/CMSPublic/WorkBook

[2] D. J. Lange. “The CMS Reconstruction Software”, Journal of Physics: Conference Series, 2011. Vol. 331, no. 3. P. 032020.