Journal of Physics: Conference Series **331** (2011) 072046 doi:10.1088/1742-6596/331/7/072046

CMS Configuration Editor: GUI based application for user analysis job

A de Cosa[12]
Istituto Nazionale di Fisica Nucleare Sezione di Napoli
Complesso Monte Sant'Angelo, Ed. 6 – Via Cintia
I-80126 Napoli, Italy

E-mail: annapaola.de.cosa@cern.ch

Abstract. We present the user interface and the software architecture of the Configuration Editor for the CMS experiment. The analysis workflow is organized in a modular way integrated within the CMS framework that organizes in a flexible way user analysis code. The Python scripting language is adopted to define the job configuration that drives the analysis workflow. It could be a challenging task for users, especially for newcomers, to develop analysis jobs managing the configuration of many required modules. For this reason a graphical tool has been conceived in order to edit and inspect configuration files. A set of common analysis tools defined in the CMS Physics Analysis Toolkit (PAT) can be steered and configured using the Config Editor. A user-defined analysis workflow can be produced starting from a standard configuration file, applying and configuring PAT tools according to the specific user requirements. CMS users can adopt this tool, the Config Editor, to create their analysis visualizing in real time which are the effects of their actions. They can visualize the structure of their configuration, look at the modules included in the workflow, inspect the dependences existing among the modules and check the data flow. They can visualize at which values parameters are set and change them according to what is required by their analysis task. The integration of common tools in the GUI needed to adopt an object-oriented structure in the Python definition of the PAT tools and the definition of a layer of abstraction from which all PAT tools inherit.

1. Introduction

The analysis workflow for the CMS experiment is based on the concept of modular analysis, and it is completely managed within CMSSW, the framework of CMS [1]. CMSSW provides a flexible way to develop user analysis code. The Python scripting language is adopted to define the job configuration that drives the analysis workflow by managing all modules involved and setting parameters for each one. The number of modules involved is usually rather high, making challenging operations such as inspecting and editing of the entire configuration. The ConfigEditor is a graphical tool conceived to facilitate physics analysis development in the CMSSW environment. It makes use of a set of common tools defined in the CMS Physics Analysis Toolkit (PAT) [2][3][4]. They can be steered and configured allowing customization of data flow and analysis to fit specific user physics issues. The

[1] Università degli Studi di Napoli "Federico II", Naples, Italy
[2] INFN Sezione di Napoli, Italy

Published under licence by IOP Publishing Ltd

integration of PAT tools in the Graphical User Interface (GUI) is achieved by applying an object-oriented organization in their definition.
The design of the ConfigEditor is aimed to meet needs of all kind of users, independently from their level of experience. One of the main goals of this tool is the absence of any programming knowledge requirements for developing a basic analysis. A typical user-defined analysis can be produced starting from a standard configuration file, applying and configuring PAT tools according to the requirements of the specific task. The ConfigEditor allows to create such analysis without writing any lines of code, providing at the end of browsing-editing chain as final product a Python configuration file ready to be run.
The two main features of the ConfigEditor are Browsing and Editing configuration files. They will be explained in detail in the next two sections. The section 4 will be dedicated to more technical details on object-oriented structure conceived for tools integration with the GUI.

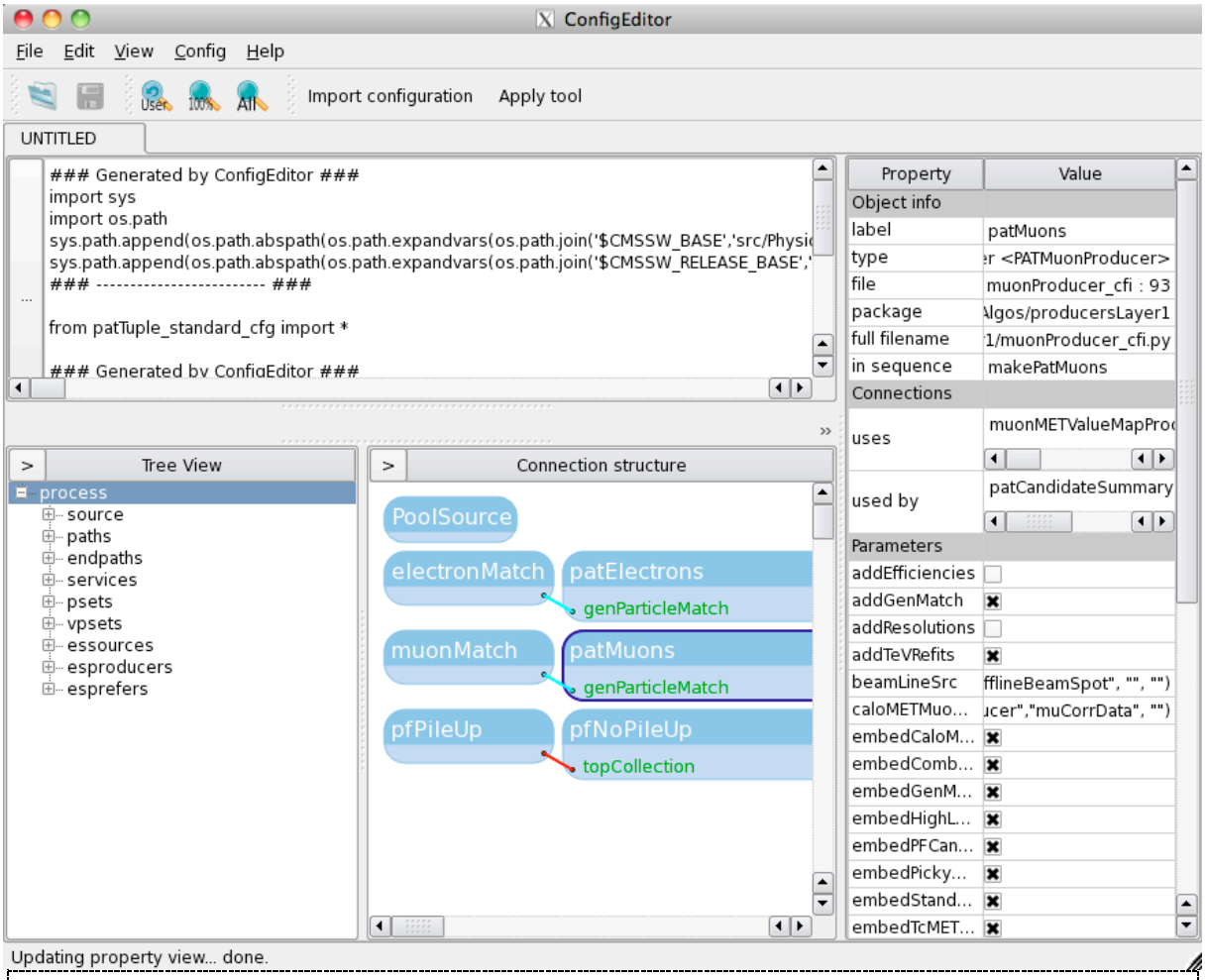

Figure 1. Configuration browsing with the ConfigEditor. The configuration sequence is visualized by a *Tree View* in the left column, the same sequence is shown in the central part in the *Connection Structure* by a graphical representation. The *Property View* Box on the right gives information about the selected module: in which file it is implemented and in which package user can find it, its connections with other modules and finally the list of its configuration parameters.

2. Browsing with the ConfigEditor

The configuration of a CMS analysis workflow may be spread over hundreds of files. It makes it difficult inspecting the entire chain of modules and managing each aspect of the analysis. The graphical interface of the ConfigEditor allows to inspect Python configuration files and to browse them visualizing all involved modules and all other included configuration files. Figure 1 shows how the GUI appears to the user. It is divided into three columns. The *Tree View* in the left column and the graphical representation, in the *Connection Structure* box in the centre, show graphically the job structure giving the possibility to analyze module by module the entire organization. In the *Property View* box (on the right) the user can get information about the selected module, checking which dependencies with other modules are existing and which is parameter setting, specifying in which file certain module can be found. The Graphical Representation can be saved and used, for instance, for documentation purposes.

3. Editing with the ConfigEditor

The ConfigEditor aids the user in setting up an analysis workflow. Its manual implementation can be rather complex and can require a good level of knowledge of the CMS framework and of the Python scripting language. By the ConfigEditor user can access graphical to specific tools developed for workflow configuration managing. This feature allows to approach analysis configuration even if user has not a good grasp of CMSSW functionalities and of the Python language. The Physics Analysis Toolkit provides several tools conceived to customize and configure PAT workflow and output. They are provided as plugins to import in Python configuration and to call like simple functions. An example from these tools is the *removeMCMatching*. It can be included to remove the matching of the reconstructed particles to generator particles in order to run the analysis on data.
A typical analysis consists of three main steps. The following subsections will explain how to edit an analysis configuration with the ConfigEditor graphical interface.

3.1. Import Configuration

To create the analysis workflow we need to start from a standard configuration. Clicking on *Import configuration* in the menu bar of the GUI the users can choose the Python configuration file to start from among those available and import it (Figure 2).

3.2. Apply Tools

To customize the imported configuration file it is useful to import a specific tool for the analysis task Clicking on *Apply tool* on the menu bar and selecting it in the tool box among all tools available. For each one a description is provided together with the list of parameters to set (Figure 3). *AddJetCollection* tool, for instance, can be included in configuration file to add a further jet collection to user PAT-tuple.

3.3. Replace Parameters

In addition to apply one or more tools, it is possible to configure parameter setting by replacing values in the *Property View* box on the right for a selected module in order to complete the customization (Figure 4).

3.4. Resulting User Configuration File

The resulting user configuration file appears in real time in the top left corner. User can check the configuration action by action. Once saved the configuration file is ready to be run (Figure 5).

4. Technical details: Python class structure for tools

PAT tools are developed according to a Python object-oriented structure with the definition of a layer of abstraction all tools inherit from: the *ConfigToolBase*, the base class which works as interface with the GUI. The integration of such tools within the ConfigEditor is allowed by Python introspection abilities. By means of introspection mechanisms we can examine Python objects, get information about them and manipulate them. To develop the *ConfigToolBase* class and integrate it with the GUI a huge use of introspection has been made. It results in a flexible structure providing features needed for application in the GUI:

- build step by step configuration files;
- get documentation of each tool and parameter;
- check for right type and value for inserted parameters;
- add comment lines to config files;
- manage parameters changing their values.

The interface of *ConfigToolBase* class consists of a series of methods to manage tools properties (parameters and tool actions) and connect them with the GUI providing the way to access them, to get information about them, to dump code needed to include tools in the configuration file the user is creating with the ConfigEditor.
The object-oriented structure allows to store tools added to the process and applied modifications creating a history of the process itself. Tools and modifications can be added to the history and also removed. The process inspection code allows to manage modifications and it can also be used for debugging configuration. The method *apply* implemented in the *ConfigToolBase* class, add the tool itself to process history and call *toolCode* method. The tool action has to be implemented in this method for all daughter classes. *dumpPython* method returns a string with the code to include in a configuration file to apply the tool. Documentation about tools is accessible via *__doc__* attribute, a string containing object description. A parameter can be added to the tool by *addParameter* method, specifying its label, value, description, type and, optionally, a list of allowed values. Information about each one can get to provide documentation in the GUI.
In *ConfigToolBase* class there are two methods implemented to make a check about correctness of parameter type and value inserted by the user: *typeError* and *isAllowed*. In case type is not of kind required an error message appears telling the user which is the right type. *isAllowed* method acts in case a list of supported values is specified for the specific tool. If the user put a wrong value a message advises him it is not supported, and suggests him the list of allowed values. Comment can be added to configuration file by *setComment* method. The comment added by the user is dumped in the configuration file together with the other lines of code. Python inspection acts in parameter managing as well. Modifications are added to process history and dumped by *dumpPython* function.
Daughter classes have to be implemented just inheriting from the *ConfigToolBase* and redefining *__init__* (the object constructor) calling *addParameter* method for each tool parameter, specifying tool label and implementing *toolCode* method according to the specific tool action.
This structure can be adopted for all kind of tools, including user-defined ones, not only for PAT.

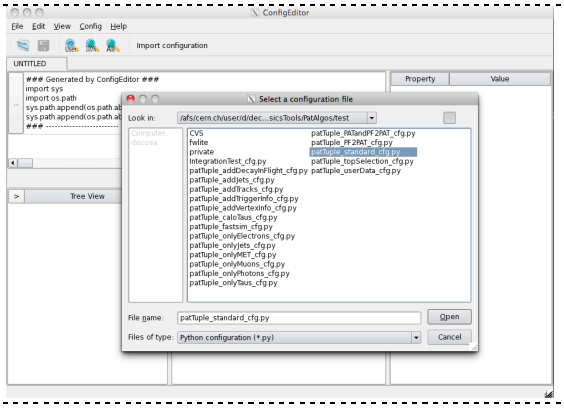

Figure 2. Configuration editing with the ConfigEditor. First step: Import Configuration. ConfigEditor provides a list of all available standard analyses to start from.

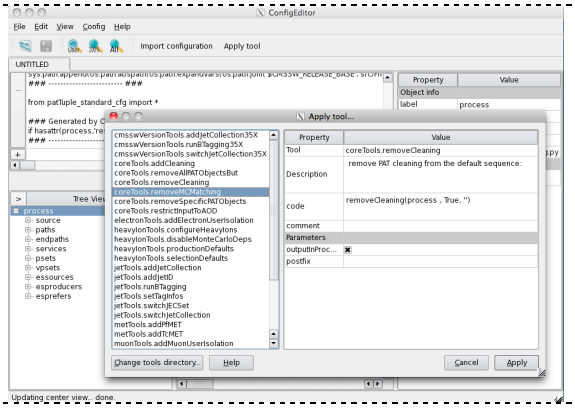

Figure 3. Configuration editing with the ConfigEditor. Second step: Apply tools. All tools have a description of their action and the list of all parameters to set.

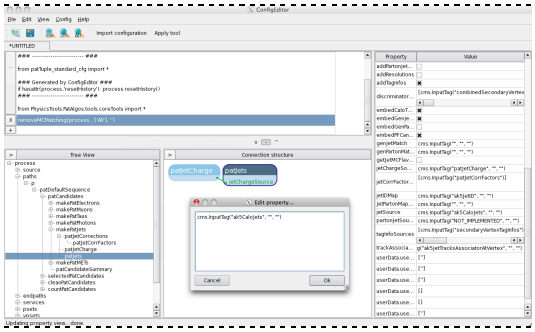

Figure 4. Configuration editing with the ConfigEditor. Third step: Replace parameters. Each module is configurable according to user needs by replacing parameter values.

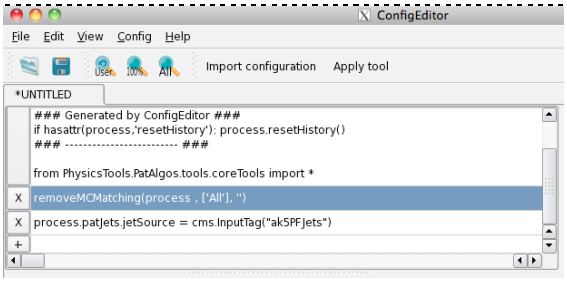

Figure 5. Configuration editing with the ConfigEditor. On the top left corner the resulting user code appears in real time. Users can check it step by step. Once saved, the code is ready to be run.

5. Conclusions

Using ConfigEditor users can access all features of configuration tools and make use of them by a graphical interface instead of writing complicated Python configurations themselves. Through the GUI users can visualize the list of all available standard config files provided and the list of all available tools and their parameters. Moreover the entire sequence is visualized module by module with all information. All changes applied by users are safe, since original file remains unchanged.

By means of this peculiar aspect, ConfigEditor has been adopted for educational purposes in the tutorials organized in the CMS community to show users how to use PAT in their analyses [5]. It is achieving great success, since it helps users to understand deeply Python configuration and how to manage it. The use of the ConfigEditor is spreading in the entire CMS community offering a quick and easy access to analysis development.

6. References

[1] Jones C. D. et al. 2006 The New CMS Event Data Model and Framework, Proc. CHEP 2006 (Mumbai, India, 13-17 February 2006).
[2] Fabozzi, F.; Jones, C.D.; Hegner, B.; Lista, L. Physics Analysis Tools for the CMS Experiment at LHC IEEE Transactions on Nuclear Science, Vol. 55, Issue 6 (2008) 3539 – 3543.
[3] W. Adam et al., The CMS physics analysis toolkit, J.Phys.Conf.Ser.219:032017,2010.
[4] Hinzmann A., Tools for Physics Analysis at CMS, Proceedings of the CHEP 2010 conference in Taipei, Taiwan.
[5] Lassila-Perini K., Planning and Organization of an E-learning Training Program on the Analysis Software in CMS, Proceedings of the CHEP 2010 conference in Taipei, Taiwan.

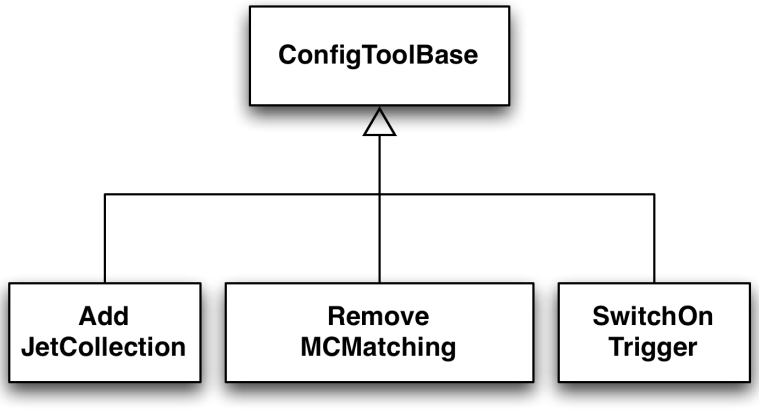

Figure 6. Object-oriented structure adopted for tools architecture. It allows tools integration with the GUI and can be applied to any kind of tool.