Python Module for Parsing COOL data in JSON format. User Manual

version 0.10.2

Ivan Yeletskikh, Andrea Formica

Contents

1 Introduction

This manual describes functionality of the Python module that retrieves data from the ATLAS COOL database via RESTful service in the form of JSON string, implements parsing and creates ROOT file with the data saved in it. The module is intended to be the part of the user web interface of COOL database. Here and afterwards we refer to this module as Python Parser Module. The current version of the manual describes Parser Module version 0.10.2. Section 2 gives a brief review of the Module structure and design of some key parts of the code. Section 3 describes the user interface, outputs of Module and presents a testing guide.

2 Code structure

The Python Parser Module fulfil the following tasks:

- Retrieve the JSON string via RESTful according to user input URL;
- Parse the JSON string;
- Create ROOT tree or ROOT histograms that repeat the whole structure of the JSON string using pyROOT;
- Depending on user options, fill the TTree and return the output to user or visualize the histograms on the web;

Each of the tasks is implemented via dedicated Python functions, gathered within 'Parser' class. In what follows, there are brief descriptions of some of the algorithms used in Parser Module.

The Parser Module receives information from COOL in the form of JSON string, which it needs to parse and to save all or part of the information. The structure of the input string is arbitrary: it could contain simple key-value pairs as well as sub-dictionaries containing keys-values, lists of such dictionaries, etc. The information about the structure of JSON string is gathered in the special part of it, called 'parserHeader'. The 'parserHeader' consists of key-value pairs that repeat the structure of the main part of the string. Instead of actual values linked with keys, the header gives types of values, e.g. "Long", "Float", "String".

The example of the input string is given below:

$$\{\text{"parserHeader"} : \{\text{"key1"} : \text{"string"}, \text{"key2"} : [\{\text{"key20"} : \text{"long"}, \text{"key21"} : \text{"float"}\}]\}, \text{"key1"} : \text{"parser\_module"}, \text{"key2"} : [\{\text{"key20"} : 0, \text{"key21"} : \text{"1.0"}\}, \{\text{"key20"} : 1, \text{"key21"} : \text{"2.0"}\}, \{\text{"key20"} : 2, \text{"key21"} : \text{"3.0"}\}]\}. \tag{2.1}$$

The "key1", "key2", "key20" and "key21" are arbitrary names of keys, distinguishing and pointing to some objects. While "key1" contains some simple string-type value, "key2" is a list of dictionaries, each containing the subsets of similar key-value pairs. Looking at header, the function "Parser::GetKeyTypeValuePatterns" pull out the information about the structure of the input string,including names of the keys and types of values stored under these keys. All this information is passed further in the form of lists – the "types_list" and the "keys_list" – to the "Parser::GetRootTree" or "Parser::GetHistograms" functions. Since the structure of the input string is essentially non-linear (as we stressed – each key could contain either simple value or dictionary of keys-valuer or list of such dictionaries...) – the "Parser::GetKeyTypeValuePatterns" is recursive. When it finds out that current key points to dictionary or list of dictionaries it launches itself to untwist the included substructures. The example of "keys_list" is given in (2.2). It contains all needed information to address all values in the input string. The "[n]" element indicates

that we deal with a list in particular location, so, we can access the value in the input string, just calling ['key2'][n]['key20'], where n in this case runs from 0 to 2. The example of "typess_list" is given in (2.3).

$$["key1", ["key2", "[n]", "key20"], ["key2", "[n]", "key21"]]. \tag{2.2}$$

$$["string", "long", "float"]. \tag{2.3}$$

The "Parser::GetRootTree" function, relying upon this information, creates python variables named after input string keys (in our example it would be "key1", "key2_key20", "key2_key21") and after that creates ROOT::TTree with correspondent branches. Simple keys-values correspond to branches of simple types, while lists in the input string correspond to, e.g., vector type branches.

The described approach converts non-linear input structure into linear ROOT tree, with all standard methods available to analyse it afterwards. Figure 1 gives an illustration of how the output ROOT file look – the information from JSON string is saved in branches and histograms.

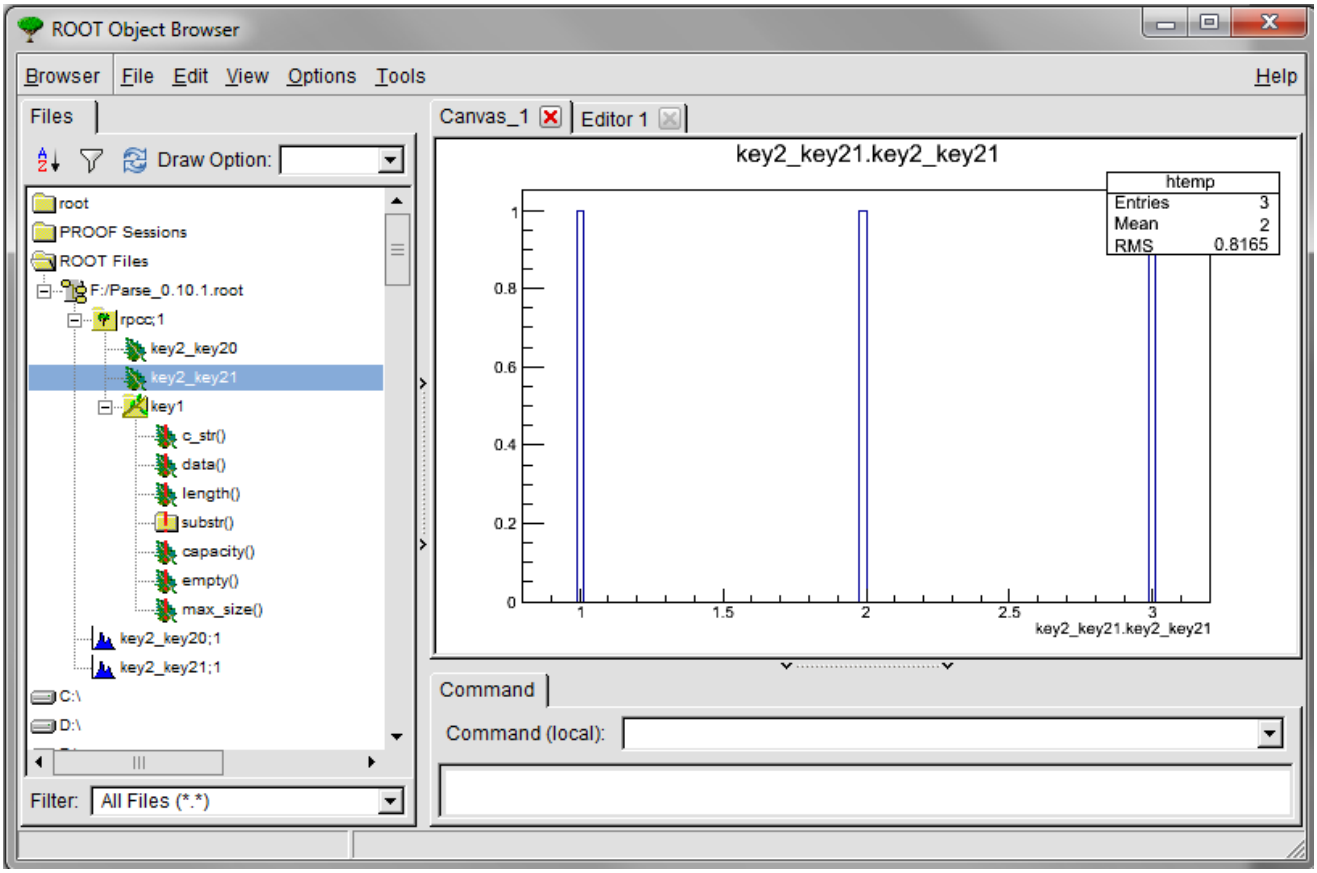

Figure 1: Parser module output example corresponding to input string (2.1).

3 Interface and outputs

The Parser Module possesses the command line interface and could be implemented in the higher-level software. In the version 0.10.3 – the command line interface could retrieve several options – the RESTful URL to obtain the input string from; the flags defining, what user wants to obtain – root tree, histograms or both; and finally the strings for ROOT tree name and output file name. The future versions may also include options, defining particular parts of information to parse.

3.1 Step by step getting started guide

One can test the Parser Module on lxplus. First of all, it is necessary to insure the ROOT and Python are set up correctly.

On 'lxplus5', the steps for doing that are the following:

- Log to lxplus.cern.ch via ssh: *ssh [username]@lxplus5.cern.ch*. In case of logging from outside CERN, one should open port needed to access COOL server: *ssh -D 3129 [username]@lxplus5.cern.ch*;

- Set up athena:
  export AtlasSetup=/afs/cern.ch/atlas/software/dist/AtlasSetup
  alias asetup='source $AtlasSetup/scripts/asetup.sh'
  asetup 17.2.0, 32, here

- Set up Python:
  export PATH=$ROOTSYS/bin:$PYTHONDIR/bin:$PATH
  export LD_LIBRARY_PATH=$ROOTSYS/lib:$PYTHONDIR/lib:$LD_LIBRARY_PATH
  export PYTHONPATH=$ROOTSYS/lib:$PYTHONPATH

- Download folder with Parser files: */afs/cern.ch/user/i/iyeletsk/public/PYTHON/PARSE_0.10.3*

- To run test one should execute file 'start.sh' in the Parser folder. This file contains the following command:
  python runParser.py –tree –hist –url 'http://voatlas135.cern.ch:8080/JBRestCool/rest/plsqlcooljso
  ATLAS_COOLOFL_RPC/COMP200/RPC/DQMF/ELEMENT_STATUS/fld/
  RPCDQMFElementStatus_2012_Jaunuary_26/tag/1650186752/chanid/0/Inf/time/data/list'
  –tree_name 'rpc0' –file_name 'parse_0.10.3.root'

  The first two options ('–tree' and '–hist') indicate that we want to store information in forms of both tree and histograms, they are both False by default, the argument '–url' contains the example of RESTful URL (RPC COOL data, element status information, one of the channels), the '–tree_name' is output ROOT::TTree name and '–file_name' is output ROOT file name, their default values are respectively 'out_tree' and 'out_file.root'.

The output tree should look as shown on Figure 2.

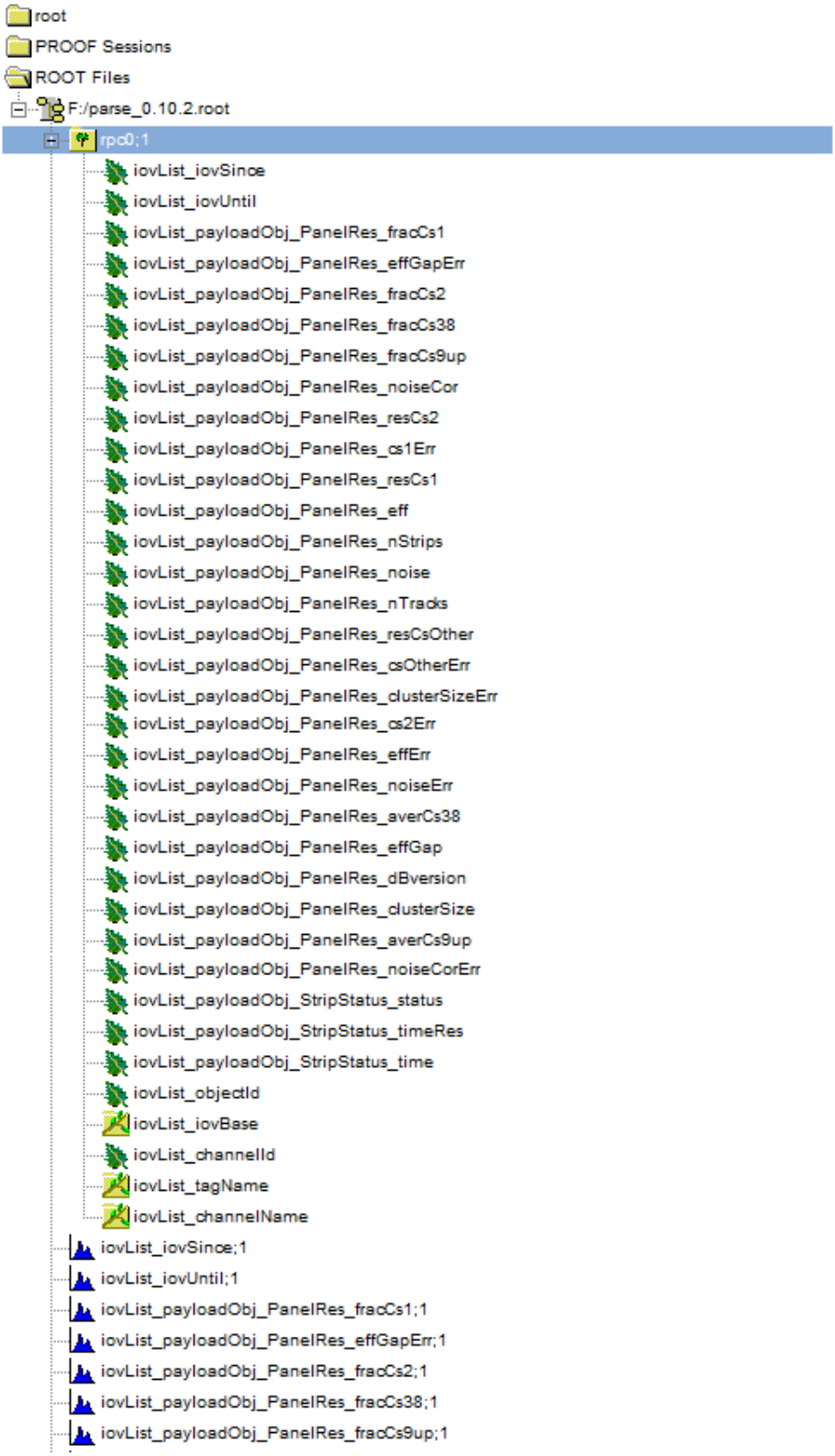

Figure 2: Output tree content for 'getting started' example.