Journal of Physics: Conference Series **219** (2010) 032050 doi:10.1088/1742-6596/219/3/032050

Expression and cut parser for CMS event data

Luca Lista[1,5]**, Christopher D Jones**[2] **and Giovanni Petrucciani**[3,4]
[1] INFN Sezione di Napoli, Complesso Universitario di Monte Sant'Angelo, via Cintia, I-80126, Naples, Italy
[2] Fermi National Accelerator Laboratory, P.O. Box 500, Batavia, IL 60510-5011, USA
[3] Scuola Normale Superiore di Pisa, Piazza dei Cavalieri 7, I-56126 Pisa, Italy
[4] INFN Sezione di Pisa, Edificio C - Polo Fibonacci, Largo B. Pontecorvo, 3, I-56127, Pisa, Italy

E-mail: luca.lista@na.infn.it

Abstract. We present a parser to evaluate expressions and Boolean selections that is applied on CMS event data for event filtering and analysis purposes. The parser is based on Boost Spirit grammar definition, and uses Reflex dictionaries for class introspection. The parser allows for a natural definition of expressions and cuts in users' configurations, and provides good run-time performance compared to other existing parsers.

1. Introduction

CMS provides collaborators with a flexible framework[1] that allows one to define applications for simulation, reconstruction, on-line event selection and off-line analysis in a modular way. Users can plug together modules provided with the standard CMS software releases along whith their user-defined modules. A module can perform analysis tasks, production of new object collections to be stored in the event, and event filtering. All modules may have configurable parameters, and jobs for event processing are configured with python scripts. In order to reach the desired degree of flexibility in the configuration, mainly for analysis applications, we realized a parser that allows the framework to interpret expressions and Boolean conditions (usually intended as "cuts") written by the user as strings, and evaluate those user-defined expressions and cuts on objects that are retrieved from the CMS event store. In this way, many analysis modules for object selection and event filtering achieve the sufficient generality for a wide range of applications without the need of writing a large number of specialized modules.

2. Parser description

The parser can interpret an expression that returns a floating point or Boolean value and then evaluate that expression on a user-supplied object. We don't support at the moment expressions that evaluate on multiple objects. This could become a future extension of the current parser.

[5] To whom any correspondence should be addressed.

2.1. Object method calls

Object methods returning a floating point, integer or Boolean value can be part of the evaluation, and are specified by the user as string variables. For instance, the expression **"pt"** evaluates in C++ as **object.pt()**, **object** being the C++ object on which the evaluation applies. Cascading method calls are implemented with a dot ('**.**'), and are automatically applied to methods that return an object either by value, by reference or by pointer. We also support persistent references, a type of reference specific to the CMS Event Data Model[1], with the same syntax. So, the expression **"track.pt"** evaluates as **object.track().pt()**, if the method **track()** returns an object or a reference to an object, or **object.track()->pt()** if the method **track()** returns a pointer or a persistent reference. We support one or more arguments to method calls that can be either any type of integer or floating point, string and enumerators (via either single or double quotes: **"abc"** or **'abc'**). So, for instance, the expression **"daughter(1).pt"** will evaluate as **object.daughter(1).pt()** and so on. For methods that admit default argument values, the default arguments can be omitted in the expression. In case of overloaded methods that may lead to ambiguous argument types, we give precedence to integer arguments. So, for instance, **"f(1)"** will match **f(int)** if both **f(int)** and **f(float)** are defined, while **"f(1.0)"** will only match **f(float)**. In case of an overloaded method that can take either a string or an enumerator, the string takes precedence.

2.2. Mathematical and logical expressions

The parser supports the usual mathematical operators **+**, **-**, ***** and **/**, as well as **^** for power raising, and the logical operators **&&**, **||**, and **!**. We support the comparison operators **<**, **>**, **<=**, **>=**, **==**, as well as the "trinary" operator cases: **a < x < b**, **a > x > b**, etc., that are frequent in analysis applications to specify a variable range. Operators can be used with parentheses of any level of nesting, and all the math functions provided by **cmath** are also supported, with either one (**sin(x)**, **log(x)**, …) or two arguments (**atan2(y, x)**, **min(x, y)**, …). We also support functions specific to some of our physics application, like the chi-squared probability, as **chi2prob(c2, ndf)**.

3. Parser implementation

The parser is written in C++ using the tool Spirit[2], which is part of the Boost C++ libraries[3]. Spirit allows one to specify the desired grammar in C++ with an approximate Extended Backus Normal Form (EBNF) syntax[4]. The implementation requires a number of helper class structures to implement the required actions to be performed on the call stack, but the core part of the grammar code is rather compact, around 70 lines of C++ code. We map string literals to method calls using Reflex dictionaries[5], now supported as part of ROOT[6]. Reflex dictionaries must be generated for the object class that is being used in the user application in order for the parser to work in a user application. Since this is a requirement for all objects that are stored in CMS Event Data Model, the ability to work with the parser is supported for all objects in the event. The result of the parsing phase is an object tree where each node represent an 'atomic' evaluation step, as in Figure 1. The complete expression evaluation is performed by recursively performing the evaluation of all the nodes in the tree.

4. Parser applications in CMS software

The cut and expression parser has become a core part of the CMS software release, and is used in a variety of applications, either as part of user modules' configuration, or as part of the configurability of pre-defined modules. In particular, it is the standard selection technique recommended for CMS Analysis Tools[7], in particular in the recently established PAT, the CMS Physics Analysis Toolkit[8].

4.1. Use in C++ code

We provide a C++ user interface to the parser via two class templates: **`StringCutObjectSelector`** and **`StringObjectFunction`**, whose template argument is the object type the parser is able to evaluate. Examples of user code fragment are given below, and are hopefully self-explanatory:

```
StringObjectFunction<reco::Track> f("px^2 + py^2");
StringCutObjectSelector<reco::Track> select("pt>15.0 && abs(eta)<2");

reco::Track trk = …; // get a track from somewhere

bool pass = select(trk);
double ptSquare = f(trk);
```

For simplicity, the expression string is displayed above as hard-coded in the C++ but in a real application this is of course taken as input from the job configuration, in order to allow the user to specify it within configuration scripts.

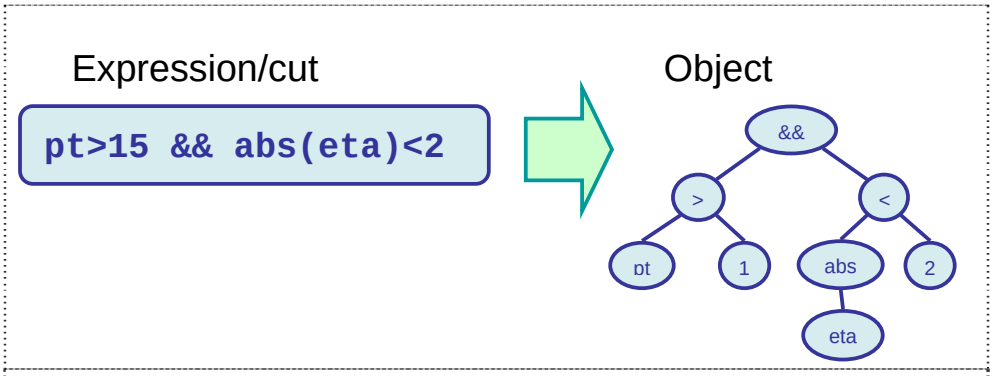

Figure 1. Example of translation from a string cut to an object tree model. The "leaves" of the tree that refer to an object method are mapped to Reflex method calls.

4.2. Use as part of a modules configuration

Several modules provide a user selection as part of their configuration implemented with the cut parser. An example of a selection of candidates for the decay Z→μμ with cuts on the muon daughters' transverse momentum (method: **`pt`**) and pseudo-rapidity (method: **`eta`**) is given below as a fragment of the configuration script. Below the daughter indices 0 and 1 represent the first and second muon of the Z decay:

```
zCandidates = cms.EDFilter("CandSelector",
   src = cms.InputTag("dimuons"),
   cut = cms.string("min(daughter(0).pt,daughter(1).pt)>20 &&" +
   "fabs(daughter(0).eta)<2 && fabs(daughter(1).eta)<2 &&" +
   "daughter(0).isGlobalMuon=1 && daughter(1).isGlobalMuon=1"
)
```

Another application where the expression parser is adopted is a generic histogrammer module that allows users to store histograms with the spectra of the desired quantities. The quantities to be plotted are specified using the expression parser. A configuration fragment for this module can be found below. In the example, the spectrum of the mass of $Z \rightarrow \mu\mu$ candidates and the maximum transverse momentum of the muon daughters is plotted:

```
zPlots = cms.EDAnalyzer("CandHistoAnalyzer",
  src = cms.InputTag("zCandidates")
  histograms = cms.VPSet(
    cms.PSet(
      min = cms.untracked.double(0.0),
      max = cms.untracked.double(200.0),
      nbins = cms.untracked.int32(200),
      name = cms.untracked.string("zMass"),
      description = cms.untracked.string("Z mass [GeV/c^{2}]"),
      plotquantity = cms.untracked.string("mass")
    ),
    cms.PSet(
      min = cms.untracked.double(0.0),
      max = cms.untracked.double(200.0),
      nbins = cms.untracked.int32(200),
      name = cms.untracked.string("mu1Pt"),
      description = cms.untracked.string("Highest muon p_{t} [GeV/c]"),
      plotquantity =
        cms.untracked.string("max(daughter(0).pt,daughter(1).pt)")
    ),
  )
)
```

A similar module that saves simple ROOT trees with customizable variable content is also supported. The parser is also used as part of the event display Fireworks[9] in order to interactively select a sub-set of the objects to be displayed: a user-specified selection can be typed in a dedicated text box area.

5. Performance

Once the expression parsing is performed at run time in the initialization phase, the evaluation of the expression or cut can be done on every object of the specified type, without the need to re-do the parsing again. The extra cost of the evaluation with respect to a C++ compiled expression is a virtual function call for each evaluation node, the call of the object methods via Reflex, and the lack of possible optimization. We observe a modest performance loss compared to C++, that makes the parser suitable for off-line analysis applications. We anyway discourage the application of this tool for performance-critical applications, like on-line event selection, where parser cuts were initially used in some cases. We compared the performance of the evaluation of the object tree model generated by our parser with equivalent CINT[10] expressions, and we measures typically factors of 50 improvements in our application.

6. Conclusions

We developed a parser that allows users to specify expressions and cuts as part of their application configuration. The parser is now widely used in CMS software applications, in particular for off-line analysis, where expressions and cuts are part of the job configuration, and in the new Fireworks event display. The tool provides good run-time performance that make it suitable for off-line analysis interactive and batch processing. The possibility to flexibly configure their application has been appreciated by users who expressed very positive feedback. A more detailed user reference of the parser can be found in Ref. [11].

References

[1] Jones C D, Kowalkoski J, Paterno M, Sexton-Kennedy E and Tannenbaum W 2006 *Proc. of CHEP 2006*, vol 1, ed S Banerjee (India: Macmillan) pp 248-251
[2] de Guzman J 2009 *Spirit*: http://spirit.sourceforge.net/
[3] Dawes B, Abrahams D, Rivera R 2009 *Boost C++ libraries*: http://www.boost.org/
[4] Donald K E 1964 Backus Normal Form vs. Backus Naur Form *Communications of the ACM* vol 7 (12) pp 735–736
[5] Reflex: http://root.cern.ch/drupal/content/reflex
[6] Brun R and Rademakers F 1996 *Nucl. Inst. & Meth. in Phys. Res.* vol A **389** pp 81-86
See also http://root.cern.ch/
[7] Lista L, Fabozzi F, Jones C D and Hegner B 2008 Physics Analysis Tools for the CMS Experiment at LHC IEEE Trans. on Nucl. Sci. vol 55 (6) pp 3539-3543
[8] Petrucciani G *et al.* 2009 PAT: the CMS Physics Analysis Toolkit *Proc. of CHEP 09*, Prague, 21-27 Mar. 2009
[9] Kovalskyi D 2008 Fireworks: A Physics Event Display for CMS *Proc. of CHEP 09*, Prague, 21-27 Mar. 2009
[10] Goto M *C++ Interpreter - CINT*, CQ publishing, ISBN4-789-3085-3 (Japanese)
See also http://root.cern.ch/drupal/content/cint
[11] Physics Cut and Expression Parser - CMS Offline Software Guide:
https://twiki.cern.ch/twiki/bin/view/CMS/SWGuidePhysicsCutParser