ROOT4J - Pure Java ROOT IO Library
Spark ROOT - Bridging ROOT to Spark

Viktor Khristenko (Iowa), Jim Pivarski (Princeton University — DIANA)

Motivation

- Using Spark for Physics Analyses requires data access!
- C/C++ Run-time environment -> JVM - hard!
- Should utilize ROOT's TStreamerInfo record for JIT compilation of Classes required - no need to replicate the whole ROOT's Class Infrastructure,
    - ROOT reading facility can actually be quite lightweight!

Previously rejected solution

dianahep

FreeHEP ROOTIO

Developed by Tony Johnson in 2001

FREE HEP

Last Published: 2013-03-01 | Version: 2.2.1 FreeHEP | JAS | WIRED

General
- **Introduction**
- License
- Team

User Info
- Summary
- API Doc
- Jar File(s)
- Dependencies
- Forum
- Bug Reports

Developer Info
- Source Code

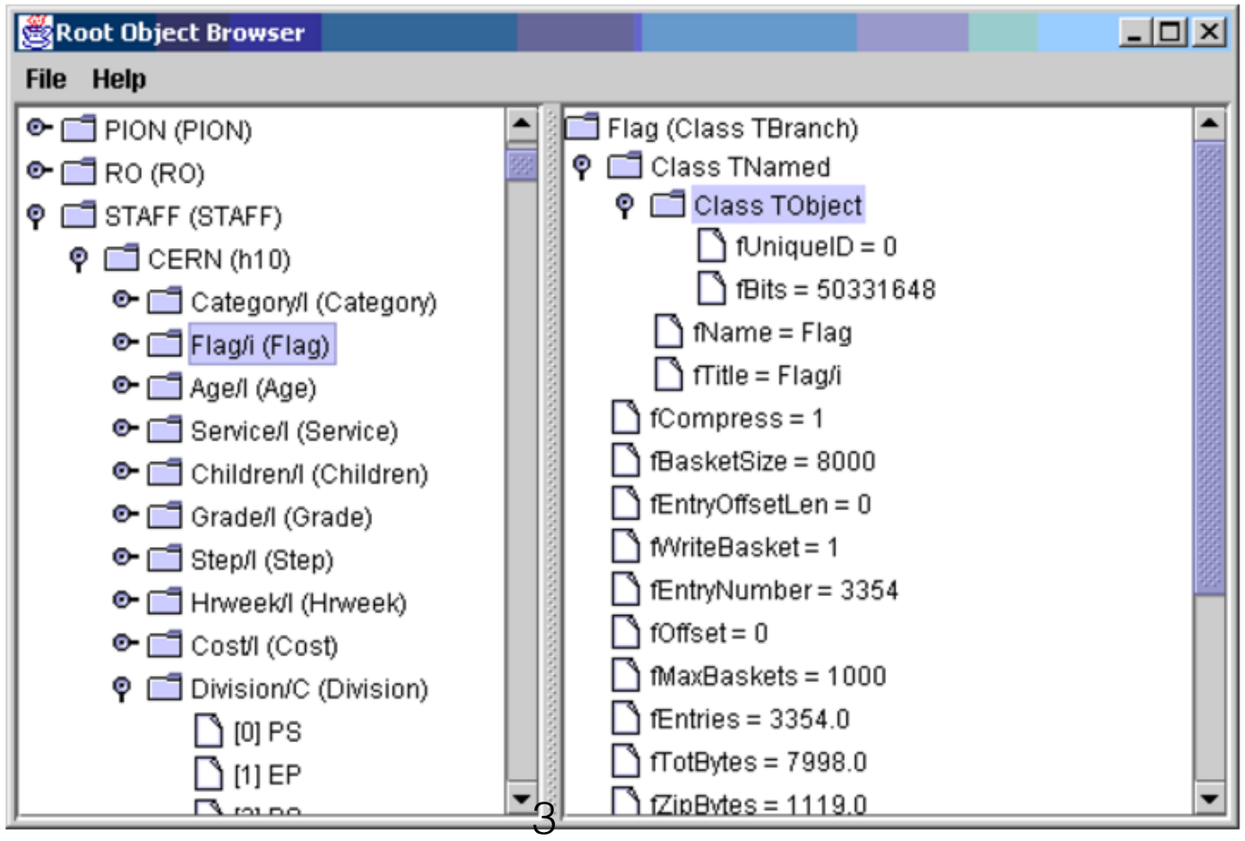
Built by: maven

Root Object Browser

As an illustration of the use of the Java interface, we have built a sample application which is a simple Root Object Browser. It can be used to open any Root file and look at all the objects inside the file. If you already have Java 2 installed (JDK 1.3), you can download the root.jar file containing the application, and run it using the command:

```
java -jar root.jar
```

(on Windows you can just double-click on the root.jar file). A screen shot of the application is show below. The pane on the left shows the directory structure of the file. The object browser knows how to navigate directories (TDirectories), trees (TTrees and TBranches) and these will all be shown in the left pane. Clicking on any object in the left pane will cause the details of the object to be shown in the right pane. The right pane knows how to follow embedded pointers to other objects.



ROOT4J **LGPL 2.1 (like original)**

A fork of http://java.freehep.org/freehep-rootio/

Under rapid development 👍

root4j is a pure Java IO Library for ROOT **with no binding** to the original implementation of ROOT in C++. Building on success of ROOT, we mimic the behavior of the classes from the original implementation. root4j utilizes the TStreamerInfo Record to infer the schema/description of Classes present and relies on JIT compilation of classes for objects requested to be read in.

Supported Features

- Large set of main ROOT Classes is available.
  - TDirectory
  - TKey
  - TTree - TBranch/TBranchElement, etc.....
  - TStreamerInfo
  - ...
- Reading ROOT files. **Able to read CMS (MINI)AOD files**
- Using TStreamerInfo to infer the description of classes present in a ROOT file.
- JIT Compilation of classes for objects that are requested to be read in.
  - **There is an initial list of classes that do already have the streaming functionality(the rule for how to read the bytes and organize them into BasicTypes) implemented.**
  - TList is such an class - description.
- TTree support.
  - **Currently only old-style (thru leaves and baskets) exists for iterating thru the TTree entries.**

Spark ROOT

Under rapid development 👍

Connect ROOT to ApacheSpark to be able to read ROOT TTrees, infer the schema and manipulate the data via Spark's DataFrames. Reading is provided by root4j.

Requirements

- Apache Spark 2.0.
- Scala 2.11
- root4j - **installed locally - available in the local Maven cache**

Quick Test Example - No Schema inferring.

```
./spark-shell --packages org.diana-hep:spark-root_2.11:0.1-pre1,com.databricks:spark-avro_2.11:3.0.1
import org.dianahep.sparkroot._
val df = spark.sqlContext.read.root("/Users/vk/software/diana-hep/test_data/test_1.root")
------
NOTE: read interface is analogous to AVRO or Parquet, or other formats: json....
------

scala> df show
warning: there was one feature warning; re-run with -feature for details
+--------------------+
|               muons|
+--------------------+
|[[46.21682,0.0,0....|
|[[29.387526,0.0,-...|
|[[66.35497,0.0,-2...|
|[[39.299873,0.0,-...|
|[[44.04615,0.0,0....|
|[[91.52747,0.0,-2...|
|[[37.770947,0.0,1...|
|[[55.39022,0.0,-2...|
|[[63.379906,0.0,-...|
|[[32.845856,0.0,-...|
|[[32.77975,0.0,-1...|
|[[118.41255,0.0,0...|
|[[47.456944,0.0,1...|
|[[39.395023,0.0,-...|
|[[231.34692,0.0,-...|
|[[108.01287,0.0,0...|
|[[34.297974,0.0,2...|
|[[50.622807,0.0,2...|
|[[43.058167,0.0,-...|
|[[47.028915,0.0,1...|
+--------------------+
only showing top 20 rows
```

Currently tested ROOT files by hardcoding the SchemType

At this point we are only reading, analyzing - no writing back to ROOT - will be added

```
scala> for (x <- df) println(x)
    [WrappedArray([46.21682,0.0,0.23936497,-0.06334016], [40.08369,0.0,1.0085266,-2.7310233])]
    [WrappedArray([29.387526,0.0,-0.8683134,-1.2712506], [12.471459,0.0,-0.98095286,-1.5620013])]
    [WrappedArray([66.35497,0.0,-2.2571073,1.7760249], [14.915713,0.0,-0.019391235,1.1315428])]
    [WrappedArray([39.299873,0.0,-1.3723173,-2.41111], [18.740532,0.0,-1.4823006,-2.3708372])]
    [WrappedArray([44.04615,0.0,0.5730408,3.01369], [42.99739,0.0,1.411812,-0.06999289])]
    [WrappedArray([91.52747,0.0,-2.2693365,-2.974106], [67.13079,0.0,-2.2886407,-2.958496], [46.67421,0.0,-
------
each WrappedArray corresponds to 1 TTree Entry with 2 or more muon objects per 1 Array.
Can use pattern matching with case classes.
------
scala> case class Muon(pt: Float, q: Int, eta: Float, phi: Float);
scala> case class Event(muons: Seq[Muon]);
scala> val rdd = df.as[Event]
scala> for (x <- rdd) println(x)
    Event(WrappedArray(Muon(46.21682,0,0.23936497,-0.06334016), Muon(40.08369,0,1.0085266,-2.7310233)))
    Event(WrappedArray(Muon(29.387526,0,-0.8683134,-1.2712506), Muon(12.471459,0,-0.98095286,-1.5620013)))
    Event(WrappedArray(Muon(66.35497,0,-2.2571073,1.7760249), Muon(14.915713,0,-0.019391235,1.1315428)))
    Event(WrappedArray(Muon(39.299873,0,-1.3723173,-2.41111), Muon(18.740532,0,-1.4823006,-2.3708372)))

------
this way it is much easier to identify what is what... and makes it much more elegant for analysis
------
```

Next Steps

- RoadMap - both repos contain “detailed” description of tasks
  - https://github.com/diana-hep/root4j
  - https://github.com/diana-hep/spark-root
- The work has begun
  - Priority right now - Provide an automatic Schema Infernal via TStreamerInfo.
  - HDFS Access
  - *** XRootD - will simplify Creating a real test bed - no need for locally stored files. ***