ROOT analysis in Java

Khushi Taori[1]

[1] *Physics, Georgia Institute of Technology, Atlanta, GA 30332*

ABSTRACT.

Data from Large Hadron Collider experiments at CERN, such as ATLAS, are stored in ROOT files, which can only be read through ROOT itself. ROOT is a data analysis framework written in C++ that is the main platform for analysis of High Energy Physics (HEP) data. In order to make analysis easier, a Java program (Java Analysis Studio For Particle Physics, or Jas4pp) was created to read ROOT files. However, this program is unable to read files created with ROOT V6, an important update to ROOT file structure with features such as an in-time compiler and a way to change the input/output capability of data. The purpose of this project was to inventory what ROOT V6 data structures are readable with the current Java program, debug any limitations and potentially update it to allow reading of all containers. We created a file containing many commonly used ROOT objects, such as histograms and trees. This file was opened using Jas4pp libraries, and all errors were noted down. Many objects such as TTree, in which most HEP data is stored, could not be opened. The Java library that supports this program was examined, and its structure was determined. The issue was found to most likely be that a class too large or too small was generated as the objects are loaded, and thus the objects could not be read. This problem was not corrected but future work will focus on fixing this issue. The updating of this Java library will allow for more efficient analysis of HEP data and collaboration with other fields.

I. INTRODUCTION AND BACKGROUND

One of the major experiments at the Large Hadron Collider (LHC) particle accelerator at CERN is ATLAS [1]. ATLAS is a general purpose detector, used for projects such as finding new particles. All data from LHC experiments are stored in ROOT [2], a data analysis framework written mostly in C++. The various types of data storage in ROOT – histograms, graphs, lists, and trees, among others – along with the large amount of data that can be stored and analyzed make ROOT the best tool for high-energy physics research. While convenient for storing high-energy physics data, ROOT comes with some limitations. For example, it is the only program that can currently open data from LHC experiments, which can be problematic since it is not a common tool outside of high-energy physics. This makes collaboration with other fields difficult. Additionally, it can only be run on Linux and is very complicated to use, and analysis is limited to C++ and pyROOT, which is slow. This makes it very hard for new users to learn to navigate the system and analyze the data.

In order to make ATLAS data more accessible and lessen the learning curve, we are developing a Java package to open these files. This Java package is known as Jas4pp, or the Java Analysis Studio For Particle Physics [3]. Jas4pp also supports analysis with other scripting languages, such as Python, Jython, and Groovy. Using programs like Jas4pp will hopefully make the process of analyzing data from ATLAS experiments faster, easier, and more efficient overall. Additionally, the Java package would be able to run on other platforms, not just Linux-based platforms. By opening up ROOT analysis, it will be possible to work with other fields as well, as ROOT has been limited to high-energy physics until now.

However, at the moment this Java package is not able to read all of the objects that this data can be stored in. With the introduction of ROOT version 6 came many important changes, including the ability of the end-user to change the input/output

behavior of TTree objects through a class called TIOFeatures and the change from a C interpreter to a just in-time compiler [4]. This update also resulted in Jas4pp not being able to properly read files written in ROOT version 6. In this project, I first determined what Jas4pp can open after the ROOT update, and then I focused on finding where the problems are in the supporting libraries.

II. DESIGN AND METHODS

The first part of this project involved determining what objects can and cannot be read with the existing Java package. First, ROOT and Jas4pp were installed on a Windows computer with the Ubuntu Linux distribution. A list of the most common ROOT objects, including various types of histograms, graphs, arrays, and vectors, was then created, and a C++ script was written to create instances of these objects. Using this C++ code, a ROOT file, Example.root, was created and then opened with Jas4pp, both in the graphical implementation and using a Jython script. Many objects could be read using the Jython script, but not all of these objects were visible graphically. This suggests a problem with the graphical portion of Jas4pp, but not the library supporting this program itself.

However, many objects could not be opened at all, whether graphically or through the script. This poses a problem, as these objects are commonly used. The TTree object especially is one of the most important used in ROOT, as almost all data produced by ATLAS is stored in these constructs. TTrees allow data to be stored in a hierarchical manner in a list of TBranches, which in turn consist of TLeaf objects. This is likely an issue with the way that the supporting library reads the objects, and the second part of this project focuses on understanding how the library works and why objects such as TTrees cannot be opened. The following image demonstrates the structure of TTrees and why they are such an essential object in ROOT.

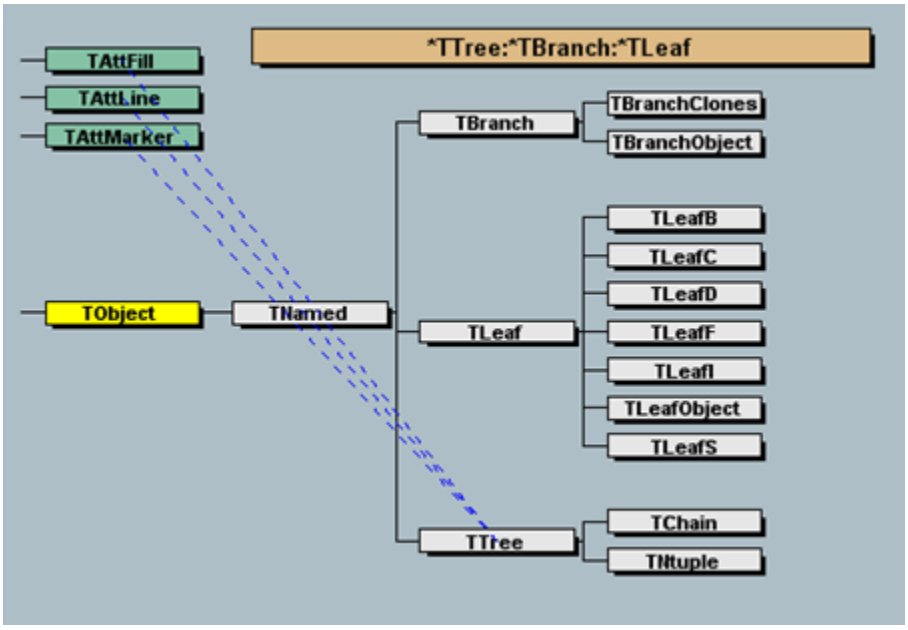

Figure 1: Structure of TTree object [5]

RootIOreader [6] is the Java library used to support Jas4pp and contains code from the FreeHEP ROOTIO project [7] as well. Since ROOT version 3, ROOT files have come with TStreamerInfo lists, which is what this package uses to read files. TStreamerInfo describes a version of an object, and it also contains the information necessary to stream and display these objects. This library was downloaded onto Ubuntu in a similar way to Jas4pp, and it included a shell script, read.sh, and a ROOT file similar to the one the first part of the project, Example.root. The package was compiled, creating a jar file, and the shell script was executed, running the jar file. The output produced after running this package was analyzed to better understand how the program interprets ROOT objects and gain insight into the issue with TTree.

III. RESULTS AND DISCUSSION

Table I, located in the appendix, shows which objects could and could not be opened using the Java packages. The first column shows results with the graphical implementation, and the second shows the results of using the Jython script. All of

the various types of histograms were visible in both ways, and in the below image you can see an example of a TH1D (1-dimensional histogram with double values) object being opened graphically.

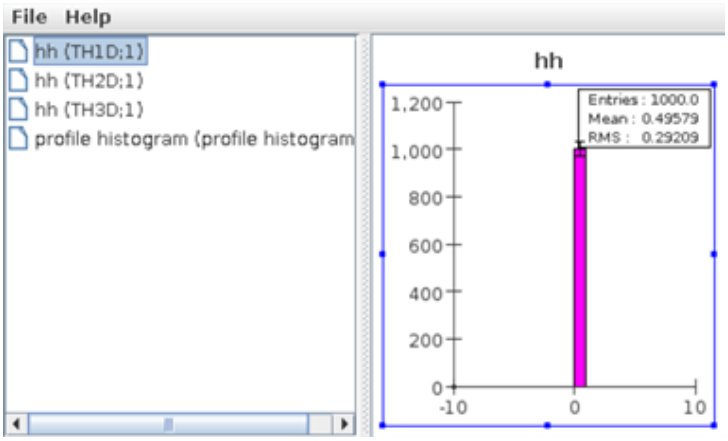

Figure 2: Histogram opened with graphical implementation of Java library [5]

Next, TF1 and TF2 were tested. These are functions with one and two parameters, respectively. As seen in Table I, these objects could not be opened using either implementation of the library and instead gave me an IOException due to a specific class being missing. Below is an image of a TF2 object and what it should look like.

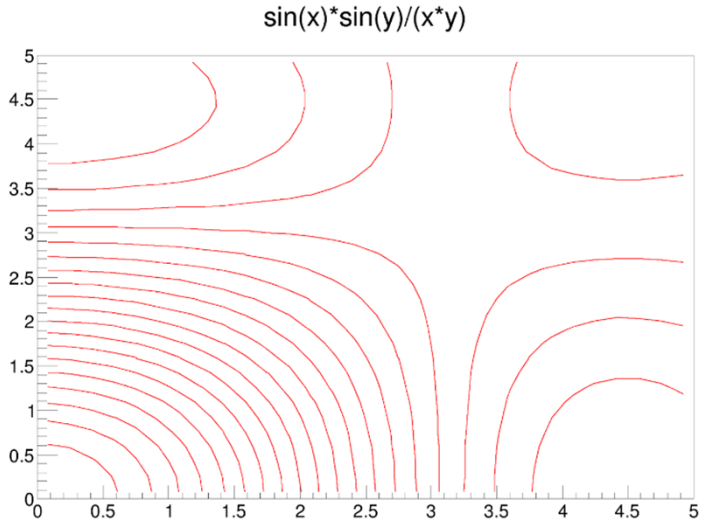

Figure 3: Example of a 2-parameter function [5]

Then, graph objects TGraph, TGraphErrors, and TGraphAsymmErrors were tested. The Jython script opened these objects successfully, but they were not visible graphically. This is a problem as these objects are used in data analysis and in visualizing results, so if the objects cannot be seen, Jas4pp cannot be used as intended for data analysis. The following image shows a TGraph object opened in ROOT, which is similar to how it should look in Java.

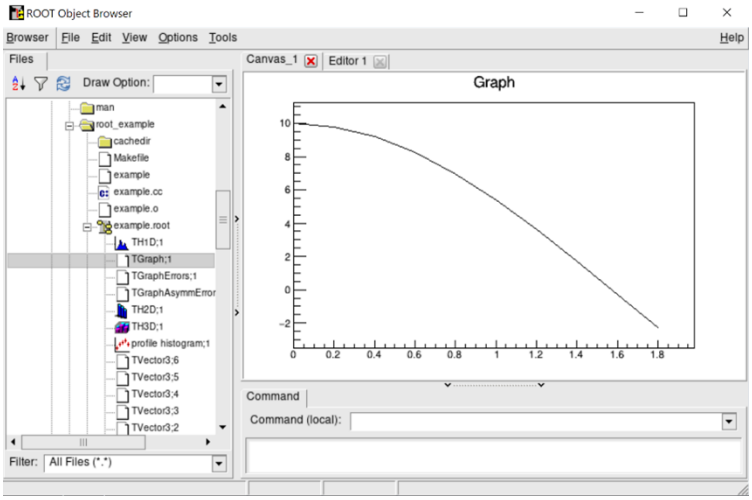

Figure 4: TGraph object displayed in ROOT Object Browser

Some non-graphical objects were also tested. These include TVector3, TLorentzVector, TMap, TList, TArrayD (an array of doubles), and TClonesArray (an array of identical objects). I did not expect these to be visible in the graphical implementation, but other than the two vector objects, none of them were visible using the script as well.

Finally, TTree could not be opened using either implementation. As discussed previously, this poses a major problem. While objects like histograms, graphs, and functions are commonly used in HEP data analysis, almost all data produced is first stored in TTrees. If the object cannot be opened, the data cannot be accessed and analyzed at all.

In the second half of the project, running the Java package resulted in quite a few different errors. Firstly, we found that Java cannot handle fIOBits, an unsigned char variable in the TIOFeatures class. While this issue was fixed by specifically defining the type of the variable, there was a larger issue that was believed to be the main source of the package's crash. This issue arises from the fact that TIOFeatures could not be streamed in the same way as other objects.

In order to understand why TIOFeatures cannot be streamed, we must first understand how this package works during runtime. First, a template for an in-memory class is created using GenericRootClass.java. Each object to be streamed comes with an interface and, as previously mentioned, TStreamerInfo. GenericRootClass.java uses this information to fill in the template class. Then, RootClassLoader.java is called to load the rest of the information and dynamically create this class, which is stored in-memory and cannot be accessed later. These classes themselves cannot be viewed, but special debugging files are created during runtime to access the information inside such in-memory classes. These files are created by the BCEL library and have the extension *.j [8].

Once the in-memory classes have been created, the Java package now has all the necessary information about the classes readily available. The shell script now calls on the class Main.java to read the passed-in file – in this case, Example.root. Main.java creates a RootFileReader object, which then takes in the file and reads it. This call sequence, specifically for TTree and TIOFeatures, is detailed in the following diagram.

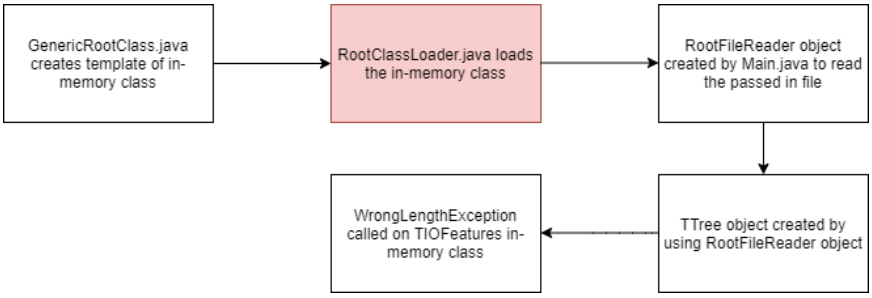

Figure 5: Flowchart displaying how RootIOreader library reads objects

In the case of TIOFeatures, from analysis of how the package runs, we have found that the issue arises when RootClassLoader.java is called. GenericRootClass.java correctly creates the template of the in-memory TIOFeatures class, but after Main.java is called, a WrongLengthException is thrown. This indicates that RootClassLoader.java did not properly load the TStreamerInfo and interface describing TIOFeatures.

This can be the result of a few different errors. One possibility is that the interface for TIOFeatures is set up incorrectly or that the TStreamerInfo is wrong. The interface for TIOFeatures is different than some of the other ones, so it is possible that it does not contain enough information or the necessary information to construct the in-memory class. Another could be that RootClassLoader.java itself is the problem and creates a class much larger or smaller than needed, causing a WrongLengthException to be thrown. I also noted that the in-memory class for TIOFeatures is in a different directory than the other in-memory classes. A potential error here is that RootClassLoader.java is unable to access the class created by GenericRootClass.java and therefore cannot fill in the appropriate information.

IV. CONCLUSION

In this project, it was found that many commonly used objects cannot be opened with pre-existing Java packages. Part of the reason that objects like TTree cannot be

opened lies in the RootClassLoader.java file and the information used to create the objects. Based on these results, future work should be done on understanding and fixing how Java loads ROOT. Future steps could include updating the TIOFeatures interface, updating the TStreamerInfo, or changing the location of the TIOFeatures in-memory class. The findings of this research project can be used by developers working both on the Java packages and ROOT itself to help approach a solution from both ends. By updating Jas4pp to be compatible with the latest ROOT releases, HEP data analysis will be much easier and more efficient, changing the course of the field drastically.

V. ACKNOWLEDGEMENTS

I would like to thank my supervisors, Sergei Chekanov and Peter van Gemmeren in the High-Energy Physics department at Argonne National Laboratory. They were very helpful and supportive throughout this entire project and have taught me a lot. My experience would not have been the same without them!

I would also like to thank the University Student Program Team who have been incredibly helpful throughout the summer and have organized this entire program for us summer students. I would like to specially thank Carol Lin, Robert Schuch, Lindsey Cullen, and Robert Boomsma for organizing such a great experience for all the interns and for all their hard work and support.

This work was supported in part by the U.S. Department of Energy, Office of Science, Office of Workforce Development for Teachers and Scientists (WDTS) under the Science Undergraduate Laboratory Internships (SULI) program.

VI. REFERENCES

[1] The ATLAS Collaboration et. al. 2008 JINST 3 S08003

[2] R. Brun and F. Rademakers, ROOT - An Object Oriented Data Analysis Framework, Proceedings AIHENP'96 Workshop, Lausanne, Sep. 1996, Nucl. Inst. Meth. in Phys. Res. A 389 (1997) 81-86. See also "ROOT" [software], Release 6.24/00, 04/15/2021, https://root.cern/releases/release-62400/

[3] S.V. Chekanov, G. Gavalian and N. A. Graf, "Jas4pp - a Data-Analysis Framework for Physics and Detector Studies" arXiv:2011.05329, Comp. Physics. Comm. 262 (2021) 107857, ANL-HEP-164101, SLAC-PUB-17569

[4] ROOT. "ROOT Version 6.12 Release Notes" https://root.cern/doc/v612/release-notes.htmlrelease-6.1204 (Accessed 25 July 2021)

[5] ROOT. "ROOT: analyzing petabytes of data, scientifically." https://root.cern.ch/ (Accessed 25 July 2021)

[6] S. Chekanov and T. Johnson, computer code RootIOreader library https://github.com/chekanov/RootIOreader/

[7] T. Johnson, computer code FreeHEP ROOTIO (FreeHEP libraries) http://java.freehep.org/freehep-rootio/

[8] Apache Software Foundation, computer code BCEL https://commons.apache.org/proper/commons-bcel/

VII. APPENDIX

Table I.

	Visible in Java GUI?	Jython
TH1D	yes	yes
TH2D	yes	yes
TH3D	yes	yes
TProfile	yes	yes
TH1F	yes	yes
TH2F	yes	yes
TH3F	yes	yes
TF1	java.io.IOException: java.io.IOException: Class not found during object read: TStreamerObjectAnyPointer	
TF2	java.io.IOException: java.io.IOException: Class not found during object read: TStreamerObjectAnyPointer	
TGraph		yes
TGraphErrors		yes
TGraphAsymmErrors		yes
TArrayD		Not found in example.root
TClonesArray		Not found in example.root
TLorentzVector		yes
TMap		Not found in example.root
TVector3		yes
TList		Not found in example.root
TTree		Not found in example.root