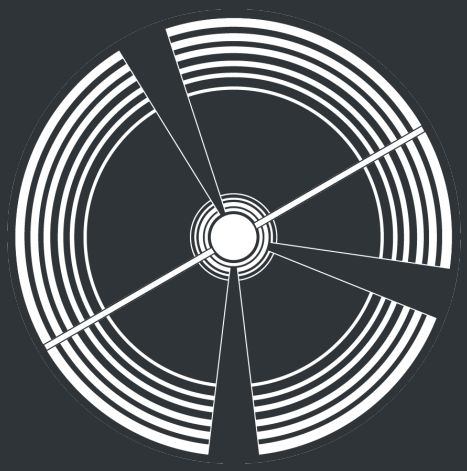

HistoMaster - User Guide

Daniel C. Foulds-Holt *

University of Cambridge

Peter C. DeRosa †

University of Massachusetts Lowell

January 30, 2021

*Email: dfh@hep.phy.cam.ac.uk
†Email: perosa88@gmail.com

Abstract

This is a software package that was conceptualised at the early hours of the morning within a nuclear physics office at the University of Massachusetts Lowell. In the pursuit of nuclear physics research each student will come into contact with ROOT - a data analysis framework written by the scientists at CERN. ROOT is immensely powerful and allows scientists and students alike to accurately manipulate and visualise data. However, it is also unwieldy, difficult to learn, and importantly, difficult to install. As students who would use ROOT at the office every day, we would often remark about how handy it would be to have it installed on our laptops or any random computer we would sit down to work at. However, ROOT is a large program and installing it is a day long process at least - it is impractical to do this for every computer you use.

This is where HistoMaster comes in. A small, self contained data analysis package. While not as powerful as ROOT, it allows the user to open and analyse '`.root`' files with a quick to install code that is easy to use. Through HistoMaster we hope to democratise the analysis of particle and nuclear physics data and lower the barrier to meaningful information.

Collaborators

Nicky Evans University of Oxford

Opening a ROOT File

ROOT is an open source code, meaning that anyone interested can open up the files and trawl through the endless lines of code in order to find what they need. When the goal is opening ROOT files, the information we need is the structure within the '`.root`' proprietary file format and the compression algorithm used for the data. The header formats are found easily enough:

ROOT File Header

Byte Number	Name	Use
1->4	"root"	= Root file identifier
5->8	fVersion	= File format version
9->12	fBegin	= Pointer to first data record
13->16	fEnd	= Pointer to first free word at the EOF
17->20	fSeekFree	= Pointer to FREE data record
21->24	fNbytesFree	= Number of bytes in FREE data record
25->28	nfree	= Number of free data records
29->32	fNbytesName	= Number of bytes in TNamed at creation time
33->33	fUnits	= Number of bytes for file pointers
34->37	fCompress	= Compression level and algorithm
38->41	fSeekInfo	= Pointer to TStreamerInfo record
42->45	fNbytesInfo	= Number of bytes in TStreamerInfo record
46->63	fUUID	= Universal Unique ID

These first 63 bytes (in a file $<$ 2GB) can be used to return the compression algorithm through `fCompress` this will be discussed later. The second piece of useful information is the byte number given by `fBegin` corresponds to the start of the first TKey. The TKey is a ROOT structure containing all the information about a TObject. The first TKey in the file will be a TFile, this can be easily discarded as it's only useful information is the filename. The second TKey will be the first TObject.

TKey Header Format

Byte Number	Name	Use
1->4	NBytes	= Length of compressed object (in bytes)
5->6	Version	= TKey version identifier
7->10	ObjLen	= Length of uncompressed object
11->14	Datime	= Date and time when object was written to file
15->16	KeyLen	= Length of the key structure (in bytes)
17->18	Cycle	= Cycle of key
19->22	SeekKey	= Pointer to record itself (consistency check)
23->26	SeekPdir	= Pointer to directory header
27->27	lname	= Number of bytes in the class name
28->..	ClassName	= Object Class Name
..->..	lname	= Number of bytes in the object name
..->..	Name	= lName bytes with the name of the object
..->..	lTitle	= Number of bytes in the object title
..->..	Title	= Title of the object
----->	DATA	= Data bytes associated to the object

In order to read the data, we must first work through the other bytes. This is done by creating three classes: ROOTHeader, FileHeader, and KeyHeader. These correspond to the ROOT file header at the beginning of the file, the TFile header which is the first TKey in the file, and the general TKey header which precedes each object in the file. The difference between FileHeader and KeyHeader is that the lTitle value in the FileHeader is 2 bytes whereas in the KeyHeader it is 1 byte. Each class is able to read the data within the binary file and by reading the ROOTHeader, the FileHeader, and finally the KeyHeader, we arrive at the first bytes associated with the data.

In ROOT files this data is compressed. The algorithm used to compress the data depends on `fCompress`, a user set value. However, in many situations, this is left unchanged. Partly because the difference between algorithms is small, and partly because many ROOT users are somewhat unaware of their options when it comes to compression. The default compression algorithm is '`zlib`' - this is a popular compression algorithm that can quickly reduce file size and easily be implemented.

Code structure

In an effort to produce modular, fast, and easy to read code, the responsibilities for reading data from ROOT files have been distributed between three primary classes defined in header files. The first of these classes is ROOTHeader.

ROOTHeader

This need only be called once for each file as it contains general information like the compression algorithm and universal unique ID. Once a file is opened, a instance of the ROOTHeader is created. This can then read the first 63 bytes from the file pointer, noting down, amongst other values, the pointer to the first data record `fBegin`. This is invariably further through the file than the first 63 bytes. In order to start reading at the beginning of the data, the code must read and discard bytes until the `fBegin`-th byte has been reached. At this point the ROOTHeader stores the pointer to the end of file `fEnd` and passes the file pointer to the FileHeader class.

FileHeader

The FileHeader is the header containing the TFile TKey information - almost exactly the same as the KeyHeader class, it only differs by an additional byte in the `lTitle` value. This again need only be read once in the opening of a ROOT file. It can then pass the data on to the first histogram reader.

Histo

Given the early stage of the project - only the Histogram infrastructure has been built yet. However, operationally, all objects will behave similarly - inheriting reading functionality from KeyHeader. Histo firsts initializes a KeyHeader, from which it reads and then extracts the header information. The `NBytes` value obtained from the header gives the length of the uncompressed object in bytes. Importantly this also includes the header. Because of this, the next (`NBytes` − `KeyLen`) bytes contain the compressed histogram data. This will need to be decompressed. This decompression will be handled later.

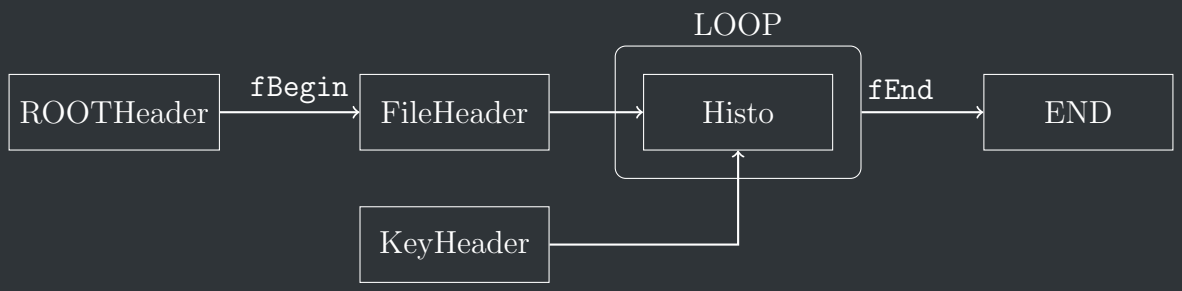

Figure 1: A diagram showing the data flow as HistoMaster opens a ROOT file.