Journal of Physics: Conference Series **396** (2012) 052011 doi:10.1088/1742-6596/396/5/052011

ROOT I/O in JavaScript

Bertrand Bellenot

CERN, PH-SFT, Geneva, Switzerland

bertrand.bellenot@cern.ch

Abstract. ROOT is used by almost all experiments throughout High Energy and Nuclear Physics to write, read and analyse data. As use of mobile devices (tablets, smart phones) is becoming more and more popular, offering a portable way of monitoring or inspecting ROOT files from any web browser, without having to install any application or library on the server side or on the client side is important. To achieve this, a JavaScript I/O library is being developed. The graphic part is done by using a third-party JavaScript visualization library.

1. Introduction

ROOT [1] is used by almost all experiments throughout High Energy and Nuclear Physics, and with the growing use of mobile devices (smart phones and tablets), the question has arisen as to whether it is worth the effort to port ROOT on those devices, or if another solution should be provided, allowing also to easily share and browse any ROOT file on the web, independently of the device used.

The main requirement of the discussed solution is to be usable on most platforms, just using the already available web browser. In addition it has to be lightweight, without requiring installation of any library or any application on the client or on the server. It has to be easy to use and easy to extend and maintain, and finally, it should be fast, with a memory footprint being as small as possible.

These requirements have been addressed by using HTML and JavaScript, with load on demand ability, and HTTP 'byte range request' [2]. Using it is as simple as copying the ROOT files on any plain web server. The data are transferred over the web via a simple HTTP GET request and the visualization happens on the client side. The major advantage of this approach is that it is open to any new platform that will appear in the future, as long as it has a web browser and JavaScript remains to be industry standard. Most of the code has been implemented in a JavaScript library named JSROOTIO, available in subversion [3].

2. Overview of the ROOT I/O subsystem

ROOT provides a machine-independent compressed binary format, including both the data and its description [4]. ROOT files can be structured into 'directories', exactly in the same way as an operating system organizes the files into folders. ROOT directories may contain other directories, so that the structure of a ROOT file is more similar to that of a file system rather than to an ordinary file.

A ROOT file contains a list of class descriptions (*TStreamerInfo* [5]) describing the schema of the object types for all class versions contained therein, as well as the elements describing each persistent data member of the class (*TStreamerElement*). The class description is recursive, because to fully describe a class, its ancestors and object data members have to be described as well. This makes it possible to automatically generate a JavaScript object out of its description.

The objects in the file are described in a logical record header (*TKey* [6]). This logical record header contains all the information needed to identify uniquely a data block on the file. The *TKey* is followed by the object data. The structure of a ROOT file, including its file header and record headers is illustrated in Figure 1.

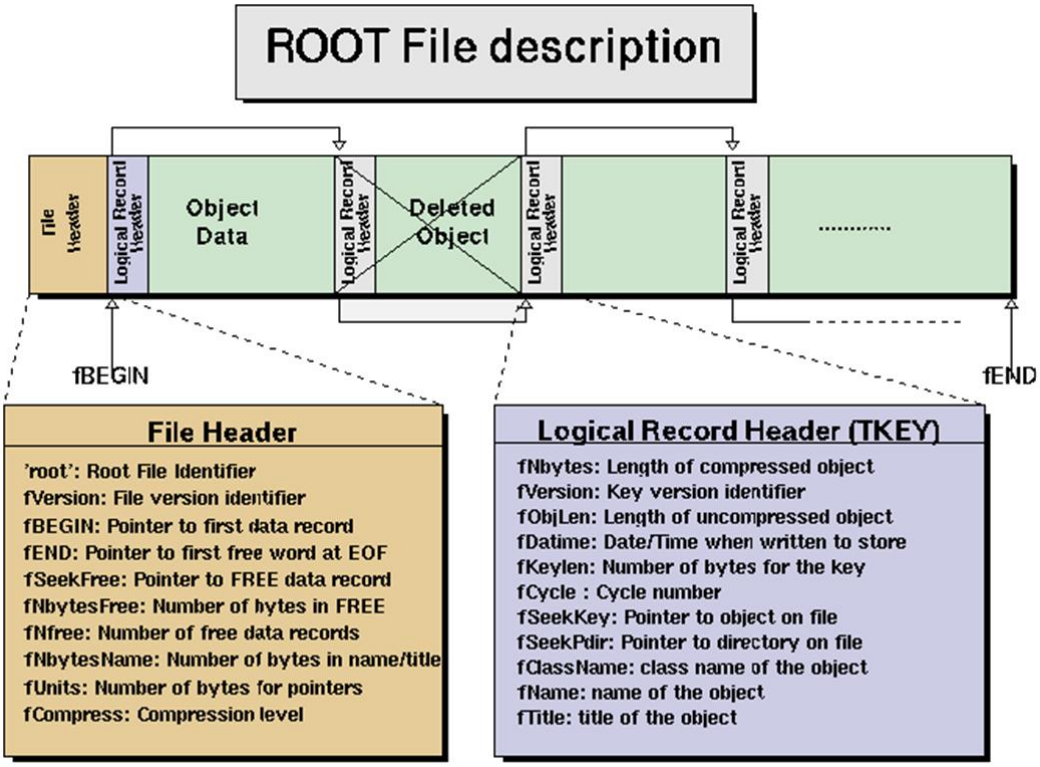

Figure 1. ROOT File description, showing file and record headers.

3. Reading a ROOT file with JavaScript

In order to minimize data transfer and memory use (i.e. to avoid downloading the whole ROOT file), we use the HTTP byte range request (available in HTTP/1.1) to download only a single compressed object when the user wants to read it. Unfortunately, some browsers, such as Opera, don't support this feature yet.

On opening a file, the system reads the list of *TStreamerInfos* and the list of *TKeys*, and displays them as a list tree in the web browser. Only when the user selects an item in the list tree, the script reads the compressed buffer from the file, inflates (decompresses) the buffer, and streams the object from the inflated buffer using the matching *TStreamerInfo*.

To read the data from the server, we use the XMLHttpRequest AJAX API to perform the HTTP HEAD and GET requests. This API is browser dependent. On Internet Explorer, the binary data are stored in its responseBody member (in a VBScript format), and have to be converted into a JavaScript string format. On other browsers, the data can be in any of their response, mozResponse, mozResponseArrayBuffer, or responseText object member.

Those compressed (zipped) objects are in binary format, and JavaScript has little support for raw binary data. The goal was not to rely on future features of JavaScript like ArrayBuffers. Thus, binary data are simply stored in a JavaScript string. Then, accessing a single byte is easy.

Inflating (i.e. unzipping) the buffers required a JavaScript implementation of zlib's inflate function [8]. Then, implementing the *TStreamerInfo* functionality in JavaScript involved parallel step by step debugging of C++ and JavaScript. The *TStreamerInfo* can be displayed for educational or informational purposes (Figure 2). We see the *TH1* base classes (*TNamed*, *TAttLine*, *TAttFill*, *TAttMarker*), and several of the *TH1* data members (fNCells, fXAxis, and fYAxis), with their type.

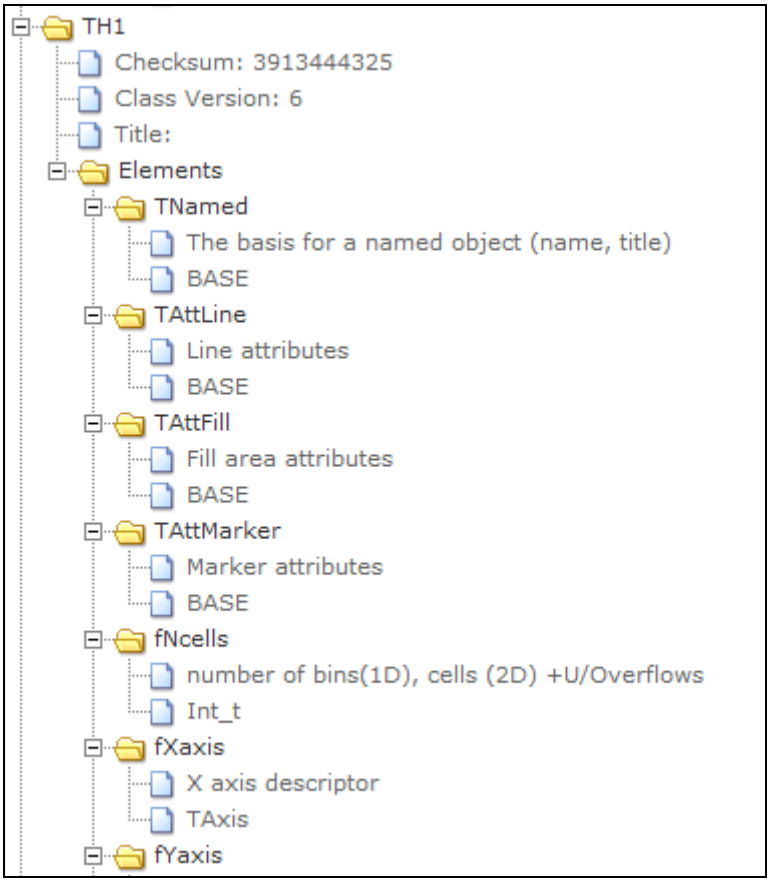

Figure 2. Example of visualization of a *TStreamerInfo*, showing the *TH1* base classes and some of its class members.

The *TKeys*, for their part, are in an uncompressed format, and they contain basic information on the object they describe, such as its name and its type. Formatting and displaying them is done with a JavaScript tree menu. Figure 3 shows the file header and the list of keys contained in the well-known hsimple.root example file. In this figure, the 'hpx' key has been opened in order to show the information describing the *TH1F* object in the file.

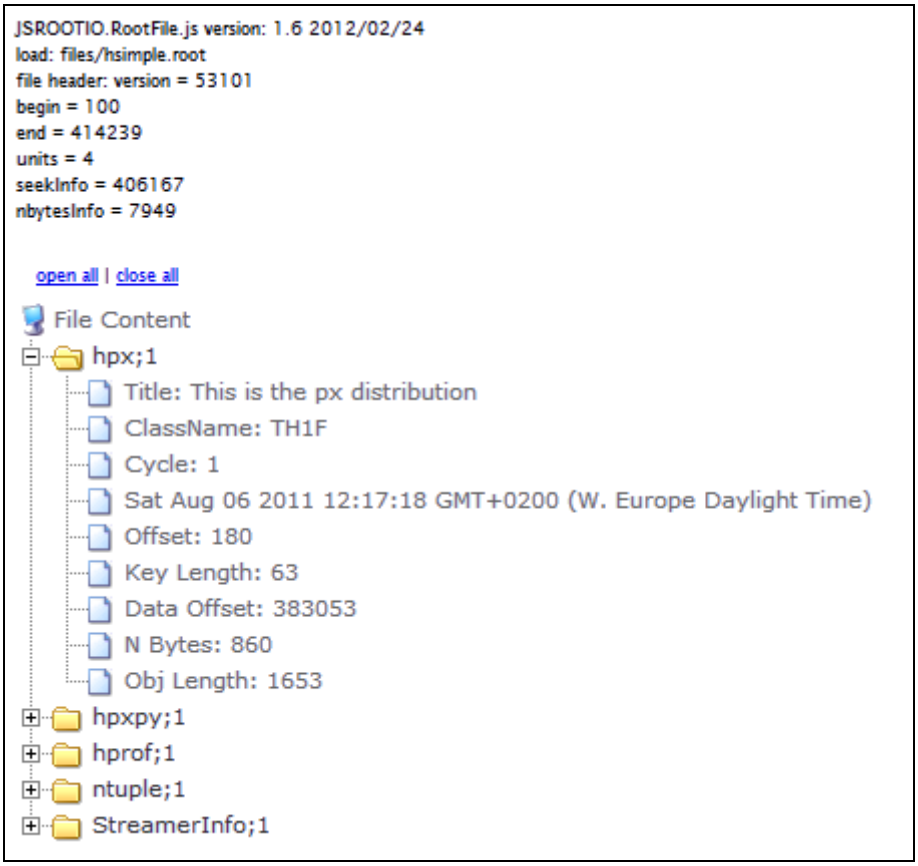

Figure 3. Example of file header and list of keys, showing some information about the 'hpx' object, like its title and its type.

In a first prototype, the classes' streamers were hard-coded, meaning there was one specific streamer written per class. The only implemented classes were *TH1*, *TH2*, *TGraph*, and *TProfile*. This was working well, but this approach has several issues:

- Streamers must be updated with every change in the original class.
- A new streamer must be implemented for every new class.
- The library is growing with every new streamer, in particular due to its complexity.

To avoid all those potential issues, we decided to use one of the nice features of JavaScript, which is the possibility to dynamically (at runtime) create objects (classes) with their data members. This allowed implementing dynamic streamers (automatically created from the *TStreamerInfos*). This also offers to potentially read any object from a ROOT file, as soon as we can read the *TStreamerInfo* of its class.

4. Display of histograms and graphs

The HighCharts [9] JavaScript charting library is used to display the histograms and graphs. It is released under the Creative Commons Attribution-Non Commercial 3.0 License [10], allowing us to adapt it to ROOT's needs. Some missing features (for example error bars, Lego plots) have to be implemented.

Figure 4 shows the traditional visualization of an histogram (*TH1F*) read from a local ROOT file in the ROOT Object Browser. We see the file and its content in the list tree, displayed on the left pane of the browser, and the histogram displayed in a canvas, on the right pane.

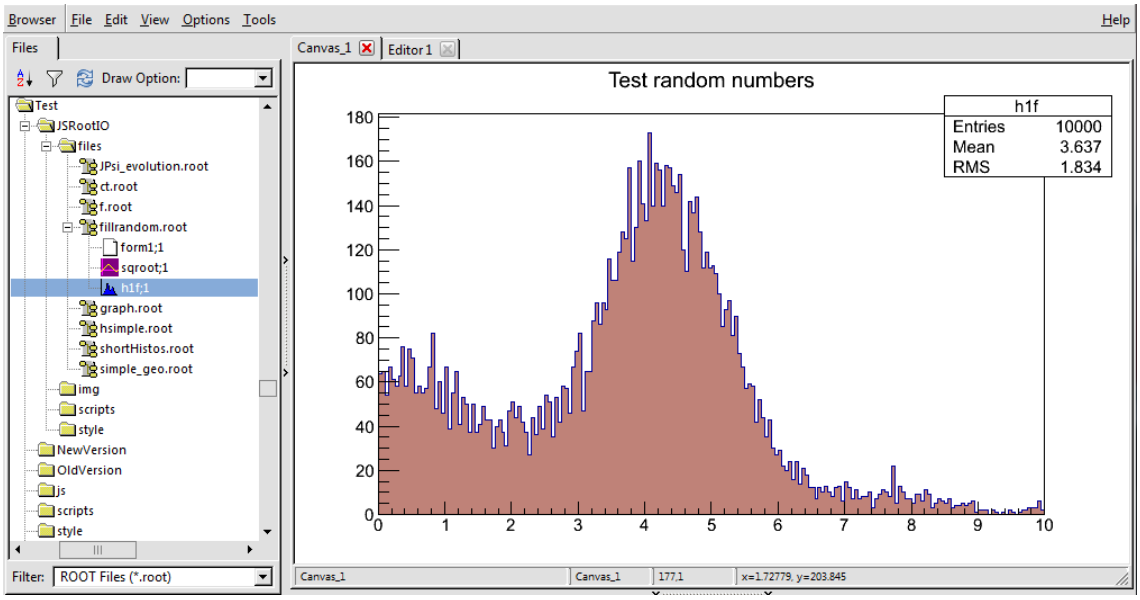

Figure 4. Traditional visualization of a local ROOT file in the ROOT browser.

Figure 5 shows the same histogram, from the same ROOT file, but now in an HTML page, implemented using the JSROOTIO library. We see more or less the same layout, with the list tree on the left, and the histogram on the right, keeping the same graphics attributes.

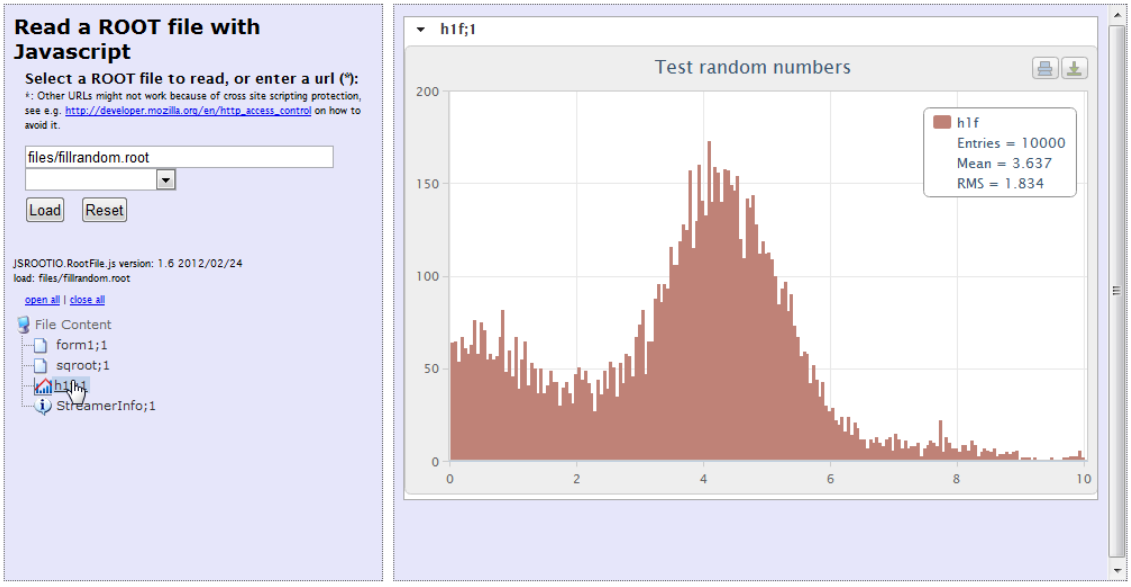

Figure 5. JSROOTIO visualization of identical histogram, in a web browser.

Figure 6 shows the directory navigation inside a ROOT file, which is a key feature. Again, the content of the directory (the list of keys) is read only on demand, i.e. when the user opens it using the left list tree.

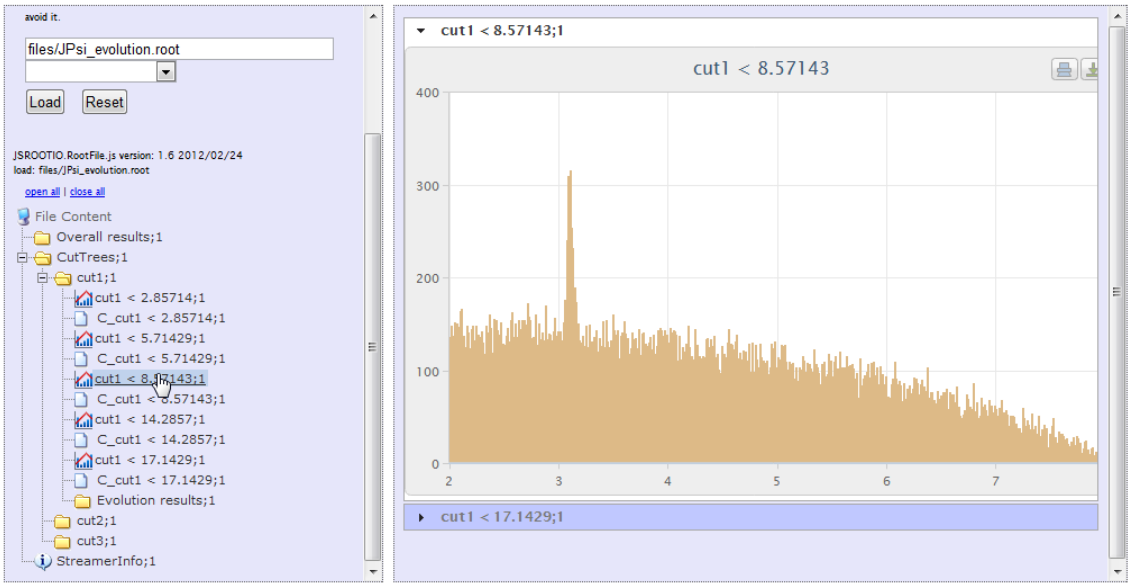

Figure 6. Example of directory navigation in a web browser.

Figure 7 shows the rendering of a two dimensional histogram *TH2F*. In this example, each cell is drawn with a colour proportional to the cell content.

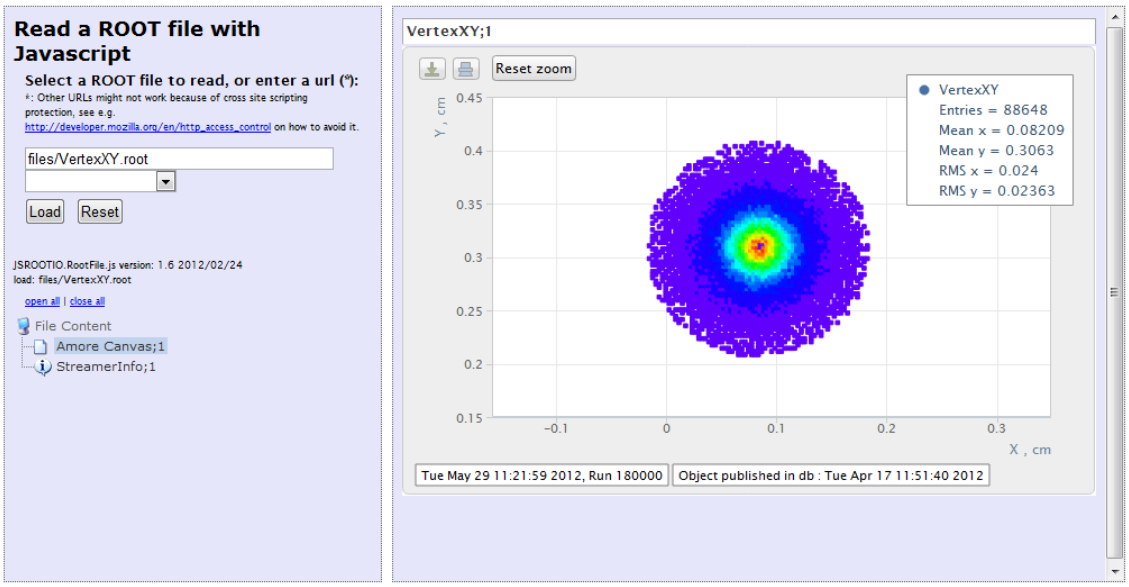

Figure 7. Visualization of a two dimensional histogram (*TH2F*).

5. Deployment

Using the library is simple; simply copy the ROOT files to be shared anywhere on the web, and create a simple HTML page next to the files. Only two lines have to be added in the `<head>`, and a few lines in the `<body>`. A complete, fully working HTML example is shown below. This is the recommended way of using the JSROOTIO library. It allows using the most up-to-date version of the code.

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
 "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
   <head>
      <title>Read a ROOT file in JavaScript (Demonstration)</title>
      <meta http-equiv="Content-type" content="text/html; charset=utf-8" />
      <link rel="stylesheet" type="text/css"
            href="http://root.cern.ch/js/style/JSRootInterface.css" />
      <script type="text/javascript"
            src="http://root.cern.ch/js/scripts/JSRootInterface.js"></script>
   </head>
   <body onload="BuildSimpleGUI()">
      <div id="simpleGUI" files="file_1.root;file_2.root;file_n.root;"></div>
   </body>
</html>
```

As previously said, there is no need to download or install anything to be able to use the JSROOTIO library. But if needed, the JavaScript, css, and HTML source code are available in subversion [11], and a working, tested, and up-to-date version is available online [12].

6. Outlook and Summary

The JSROOTIO library allows reading and displaying many ROOT objects in an efficient way, and is very easy to use. There is still missing functionality, but it is already working and usable, thanks to valuable feedback from early users. The actual status is encouraging, and implementing the missing parts seems easily feasible given the described experience. For example, some cases of automatic streaming are not fully implemented yet, and the *lzma* decompression of buffers still has to be investigated. Concerning the graphics part, the missing features still have to be implemented.

As the JSROOTIO library doesn't depend on any ROOT specific release, the development can proceed independently, and new features can be added in a transparent way (for the users).

Acknowledgments

I would like to thanks Axel Naumann, for his primary work on the JavaScript implementation of the *TKey* reading functions, Philippe Canal, for his valuable help explaining the *TStreamerInfo* and the complex internal structure of the ROOT I/O, John Harvey and Benedikt Hegner for their valuable comments and suggestions when writing this paper.

References

[1] R. Brun and al., "ROOT — A C++ framework for petabyte data storage, statistical analysis and visualization", Computer Physics Communications; Anniversary Issue; Volume 180, Issue 12, December 2009, Pages 2499-2512
[2] RFC 2616 Section 3.12: Range Units, http://tools.ietf.org/html/rfc2616#section-3.12
[3] JSROOTIO in subversion, http://root.cern.ch/svn/root/trunk/js/JSRootIO
[4] The ROOT Team, "Input/Output", http://root.cern.ch/download/doc/11InputOutput.pdf
[5] The ROOT Team, online reference guide, http://root.cern.ch/root/html/TStreamerInfo.html
[6] The ROOT Team, online reference guide, http://root.cern.ch/root/html/TKey.html
[7] The ROOT Team, online reference guide, http://root.cern.ch/root/html/TH1F.html
[8] Masanao Izumo, zlib's inflate function, http://www.onicos.com/staff/iz/amuse/javascript/expert/inflate.txt
[9] HighCharts JS, http://www.highcharts.com
[10] Creative Commons Attribution-NonCommercial 3.0 Unported,

http://creativecommons.org/licenses/by-nc/3.0/legalcode
[11] Source code availability, http://root.cern.ch/svn/root/trunk/js/JSRootIO
[12] Latest online working version, http://root.cern.ch/js