Mathematica with ROOT

Ken Hsieh,[1,*] Thomas G. Throwe,[2] and Sebastian White[3]

[1] *Wolfram Research, Inc., Champaign, IL 61820, USA*
[2] *Brookhaven National Laboratory, Upton, NY 11973, USA*
[3] *The Rockefeller University, NY, NY, 10065*

Abstract

We present an open-source *Mathematica* importer for CERN ROOT files. Taking advantage of *Mathematica*'s import/export plug-in mechanism, the importer offers a simple, unified interface that cleanly wraps around its *MathLink* -based core that links the ROOT libraries with *Mathematica*. Among other tests for accuracy and efficiency, the importer has also been tested on a large (5 Gbyte) file structure, D3PD, used by the ATLAS experiment for offline analysis without problems. In addition to describing the installation and usage of the importer, we discuss how the importer may be further improved and customized. The package may be downloaded here and a related presentation may be found here.

* kenh@wolfram.com

I. INTRODUCTION

Mathematica[1] and ROOT [2] are two powerful tools used in many technical fields. In the field of high-energy physics, the different needs of theorists and experimentalists have traditionally migrated the theorists towards Mathematica, while the experimentalists have relied on ROOT as one of the key tools of analysis. With the advent of the Large Hadron Collider (LHC), the need of collaboration between theorists and experimentalists is as great as ever.

With this in mind, we present *Mathematica* ROOT importer . Not only have we developed several functions able to import some data contained in ROOT files to *Mathematica*, we also offer a simple and unified interface to use these functions, taking advantage of the new features of `Import[]` and `Export[]` of *Mathematica* 8[3]. While it does not capture all the possible rich data contained in many root files, we present this program in the hope that it may be modified and tweaked to something more useful not only to the HEP community, but to the broader *Mathematica* and ROOT user bases.

It is important at the outset to list what the converter is, and is not, capable of. As provided, it can

- import the list of objects (and their types) stored in a ROOT file through iteration of `TKey`,
- import the list of `TLeaf` stored in a `TTree`
- import the data of single-leaf `TBranch` of the data type listed in Table I. Essentially, we support only the basic C-types and some C++ standard template library containers containing these basic types.
- import the bin and error data stored in `TH1F` and `TH2F` objects and create histograms from the bin data.

Some of the shortcomings include

- import data from `TBranch`es that contain multiple `TLeaf`s,
- import data from `TBranch`es whose data type are not listed in Table I,
- import other types of histogram `TH1*` and `TH2*` objects.

However, it should be noted that in some cases it is relatively straightforward to add additional data and histogram types.

We include a few example data files but this package has also been tested on a large (5 Gbyte) file structure, D3PD, used by the ATLAS experiment for offline analysis without problems.

In this initial release we are only distributing source files with executables for the Windows platform. For Mac and Linux machines, we include several makefiles for creating executables on the different platforms rather than the executables themselves.

Type (T)	T	v<T>	v<v<T> >	l<T>
int	x	x	x	x
unsigned int	x	x	x	x
short	x	x	x	x
unsigned short	x	x	x	x
double	x	x	x	x
float	x	x	x	x
bool	x	x	x	x
string	x	x	x	x
char*	x			

TABLE I. Supported TBranch data types. The abbreviation v<T> and l<t> stand respectively for std::vector<T> and std::list<T>, where T is a generic data type. The string listed here is std::string of C++. In some cases of std::vector<std::vector<T> > and std::list<T>, libraries may need to be generated.

II. INSTALLATION

A. Package Contents

The Mathematica ROOT importer package can be downloaded at http://library.wolfram.com/infocenter/Articles/7793/. In addition to the folder References containing several references and guides, the package contains the following files:

```
Examples/Mathematica_ROOT_M8_Usage.nb
Examples/Mathematica_ROOT_Tests.nb
Examples/basic.root
Examples/cernstaff.root
Examples/demo.root
Examples/l1.root
Examples/s1.root
Examples/th2f.root
Examples/v1.root
Examples/v2.root
ROOT/BuildMathLinkExecutable.nb
ROOT/Converter.m
ROOT/Import.m
ROOT/makefile
ROOT/makefile.linux32
ROOT/makefile.linux64
ROOT/makefile.mac32
ROOT/makefile.mac64
ROOT/root_interface.tm
ROOT/ROOT.sln
ROOT/ROOT.vcproj
ROOT/Binaries/*/
ROOT/Binaries/[Windows,Windows-x86-64]/ROOT.exe
```

where * spans those architectures supported by *Mathematica* 8{`Linux, Linux-x86-64, MacOSX-x86, MacOSX-x86-64, Windows, Windows-x86-64`} .

B. Requirements

The *Mathematica* ROOT importer requires *Mathematica* 8 and CERN ROOT[1]. The *MathLink* portion of the importer dynamically links the ROOT libraries at compile-time, and loads the libraries at run-time. At compile-time (only needed on Linux and Mac machines), the users have to supply the paths to the ROOT header and libraries files to `ROOT/BuildMathLinkExecutable.nb` or the appropriate `makefile` in order to compile successfully During run-time, the *MathLink* executable and the ROOT libraries (which may, in turn, load other ROOT libraries) depend on environment variables to locate the ROOT libraries.

On Windows machines, the path to the ROOT libraries should be included in the environment variables `$Path` and `$LIB`. On Mac and Linux machine, if the path to the ROOT

[1] The *Mathematica* ROOT importer is developed and tested with ROOT 5.28/00.

libraries is not explicitly compiled into the ROOT binaries and libraries, then, at run-time, the location of the libraries is made available through an environment variable. The user may check whether the ROOT library path is compiled into the binaries and libraries by running the `ldd` (Linux) or `otool -L` (Mac) command on the appropriate binaries and libraries. If these commands indicate that the location of the libraries is unknown, then the `$LD_LIBRARY_PATH` (Linux) and `$DYLD_LIBRARY_PATH` (Mac) are used to indicate the location. The environment variable is generally set or appended in the user's `.bash_profile` or equivalent file for the user's particular shell. Depending on the user's operating system and user interface, environment variables may or may not be available to applications launched through a menu or desktop shortcut. If the path to the ROOT libraries is not compiled into the binaries and libraries, and if evaluating `Environment["LD_LIBRARY_PATH"]` returns `$Failed` when *Mathematica* is launched in this way, then the user will be required to start *Mathematica* from a terminal session in order to have access to the environment and use the importer.

C. Installation

The installation process is divided into two main steps: generating the *MathLink* executable and copying the necessary files to a location where *Mathematica* may load it automatically.

1. *Generating the MathLink executable*

This step is needed only for users on Mac and Linux machines. For Windows users, we include pre-compiled *MathLink* executables for Windows platform (built with ROOT 5.28/00) and the users may skip directly to the next step.

A *MathLink* executable needs to compiled from the file `ROOT/root_interface.tm` included in the package. The compilation process requires first processing the `.tm` file into a `.cpp` file using the *Mathematica* utility `mprep`, and then compiling the resultant `.cpp` while linking against the ROOT libraries. We have included a Visual Studio project file for the Windows platform and a makefile for the Linux and Mac platforms to build the executables. One would have to modify a few items (such as a location of the local ROOT libraries) before being able to compile successfully. The executables can also be generated using *Mathematica* with the function `CreateExecutable`, and our implementation to build the executable using `CreateExecutable` [4] is included in `ROOT/BuildMathLinkExecutable.nb`.

The compilation target is a *MathLink* executable named `ROOT.exe`, and it should be placed appropriately, depending on the platform, inside one of the folders in `ROOT/Binaries/` as set up the makefiles. For example, the makefile for on 64-bit Linux machines would place `ROOT.exe` under the folder `ROOT/Binaries/Linux-x86-64/`.

2. Copy files to `$UserBaseDirectory/SystemFiles/Formats`

To install the *Mathematica* ROOT importer , simply copy the content of the `ROOT` folder into the directory `$UserBaseDirectory/SystemFiles/Formats`, where the *Mathematica* path variable `$UserBaseDirectory` can be found by evaluating `$UserBaseDirectory` in *Mathematica* . (In some cases, it may be necessary to create the folder `Formats` under `$UserBaseDirectory/SystemFiles`.) Similarly, the converter may be uninstalled by deleting this copy of the `ROOT` folder.

D. Library-Generation during first-time use

When the *Mathematica* ROOT importer is used for the first time, it generates the libraries needed by ROOT to support `vector<vector<*> >` types. The libraries are generated in the directory `$UserBaseDirectory/SystemFiles/Formats/ROOT` along with a lock file `CreateLibrary.m`, whose presence signals to *Mathematica* ROOT importer not to generate the libraries again in a new session.

When the user upgrades ROOT and needs to compile a new version of the *MathLink* executable, the libraries would also need to be generated. This can be done by the new *MathLink* executable once the libraries (files starting with the prefix `AutoDict*`) and the lock file `CreateLibrary.m` are removed.

III. USAGE

Once the ROOT import package is installed, there are several usages depending on the type of object stored in the ROOT file and the scope of information requested by the user. It is recommended that the users are familiar with the general syntax of the *Mathematica* `Import[]` function [5]. We list the possible usage in Table II before explaining each one in detail. It is important to note several things:

- Since `"ROOT"` is a user-defined format, we must explicitly denote the format name.
- Except for the `"Keys"` element, all the other elements require a sub-element: the name of the object to be inspected/imported.
- The returned structure and the options available to each element differ, and are explained in detail in the following subsections.
- The table only lists the native elements defined by the ROOT converter. The general, format-independent features of the *Mathematica* `Import[]`/`Export[]` framework (for example: importing multiple elements at once, the `"Elements"` element, etc.) are not listed here.

TABLE II. The element names and brief description of the *Mathematica* ROOT importer. The items in the left column are understood to be used as the second argument to the *Mathematica* `Import[]` function (with the first argument being the file name). The italicized keywords are the user-supplied variables, where *file* is the name of the imported ROOT file. The variable *tree* is the name of a `TTree` object in the ROOT file *file* and *branch* is the name of a `TBranch` object in *tree*. The variable *hist* is the name of a `TH1F` or `TH2F` object.

`Import[`*file*`, ___ ]`	Description
`{"ROOT", "Keys"}`,	`TKey` information, such as class names.
`{"ROOT", "TTreeMetadata",`*tree* `}`	`TTree` meta-info., such as `TBranch` names and data types
`{"ROOT", "TTreeData",`*tree*`}`	data from all `TBranch`'s contained in `TTree` *tree*
`{"ROOT", "TTreeData",`*tree*`,`*branch*`}`	data from a particular `TBranch` *branch* contained in `TTree` *tree*
`{"ROOT", "TH1FData",`*hist*`}`	data in a `TH1F` object given as a formatted list of numbers
`{"ROOT", "TH1FGraphics",`*hist*`}`	`TH1F` object rendered using the `Histogram[]`
`{"ROOT", "TH2FData",`*hist*`}`	data in a `TH2F` object given as a formatted list of numbers
`{"ROOT", "TH2FGraphics",`*hist*`}`	`TH2F` object rendered using the `Histogram3D[]`

A. Inspect of contents of a ROOT file through its keys

To inspect the content of a ROOT file, we may import its `"Keys"` element:

`Import[`*file*`, {"ROOT", "Keys"}],`

and since `"Keys"` is the default element for the `"ROOT"` format, we can equivalently specify only the format and drop the head `List`:

`Import[`*file*`, "ROOT"].`

The output of the function call is a list of `"TKey"` information $\{\text{KeyInfo}_1, \text{KeyInfo}_2, ...\}$, with each entry is itself a triplet:

$$\text{KeyInfo}_i = \{\text{Key Name}_i, \text{Key Title}_i, \text{Class Name}_i\}. \tag{1}$$

Some examples of inspecting ROOT files using the `"Keys"` element is given in Fig. 1.

B. Inspect the branch information of a `TTree` object

The data in a ROOT file is typically stored and organized in `TTree` objects, and we may use the `"TTreeMetadata"` element to import the metainformation of a `TTree` object. The function call

```
In[2]:= Import["cernstaff.root", {"ROOT", "Keys"}]

Out[2]= {{T, CERN 1988 staff data, TTree}}

In[3]:= Import["demo.root", "ROOT"]

Out[3]= {{h0, histo nr:0, TH1F}, {h1, histo nr:1, TH1F},
         {h2, histo nr:2, TH1F}, {h3, histo nr:3, TH1F}, {h4, histo nr:4, TH1F},
         {h5, histo nr:5, TH1F}, {h6, histo nr:6, TH1F}, {h7, histo nr:7, TH1F},
         {h8, histo nr:8, TH1F}, {h9, histo nr:9, TH1F}, {h10, histo nr:10, TH1F},
         {h11, histo nr:11, TH1F}, {h12, histo nr:12, TH1F},
         {h13, histo nr:13, TH1F}, {h14, histo nr:14, TH1F}}
```

FIG. 1. Examples of importing ROOT data with the `"Keys"` element. In these examples, we see that the file `cernstaff.root` contains a lone `TTree` object named `"T"` while the file `demo.root` contains a collection of `TH1F` objects with names of the form `"h`*i*`"`.

```
In[5]:= Import["cernstaff.root", {"ROOT", "TTreeMetadata", "T"}]

Out[5]= {{Category, Category, Int_t, 3354},
         {Flag, Flag, UInt_t, 3354}, {Age, Age, Int_t, 3354},
         {Service, Service, Int_t, 3354}, {Children, Children, Int_t, 3354},
         {Grade, Grade, Int_t, 3354}, {Step, Step, Int_t, 3354},
         {Hrweek, Hrweek, Int_t, 3354}, {Cost, Cost, Int_t, 3354},
         {Division, Division, Char_t, 3354}, {Nation, Nation, Char_t, 3354}}
```

FIG. 2. Example of importing ROOT data with the `"TTreeMetadata"` element. In this example, we see that the `TTree` object named `"T"` in the file `cernstaff.root` contains a collection of `TBranch`'s. The first `TBranch` has name `"Category"` (the first item in the list) with a title (second item) same as its name. It stores data in the form of 3354 `Int_t` objects.

`Import[`*file*`, {"ROOT", "TTreeMetadata",` *tree* `]`.

The output is a list of quartets of the form:

$$\{\{\text{branch name}_1, \text{branch title}_1, \text{data type}_1, N_T\}, \{\text{branch name}_2, \text{branch title}_2, \text{data type}_2, N_T\}, \quad (2)$$

where N_T is the number of data entries of each branch, which should be the same throughout a particular `TTree`. An example of importing `TTree` metainformation is given in Fig. 2.

C. Import a particular `TBranch` object

Knowing the name of a `TBranch` object, we can proceed to import the data inside the branch. We may do this via the import element `"TTreeData"`:

`Import[`*file*`,{"ROOT","TTreeData",` *tree*`,`*branch*`]`.

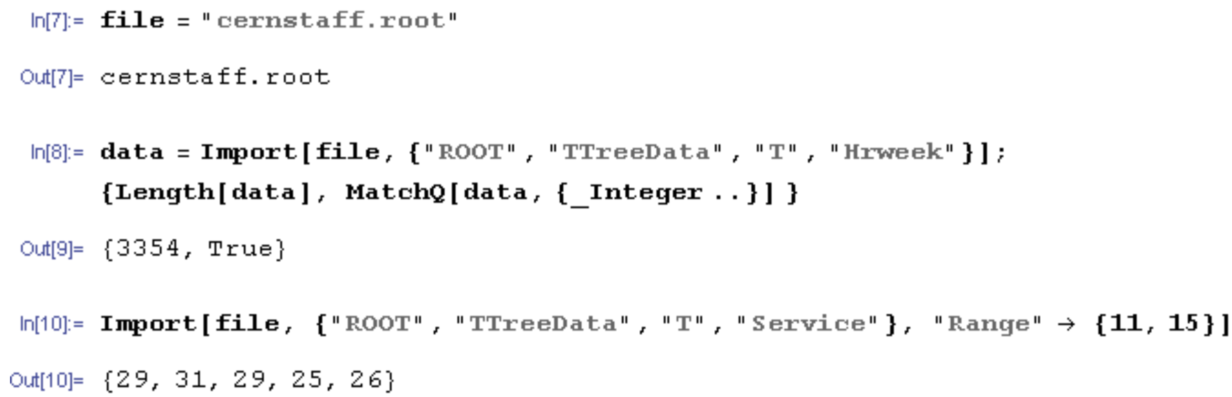

FIG. 3. Example of importing ROOT data with the `"TTreeData"` element. In the first example, we stored the data in the `TBranch` named `"Hrweek"`. As the data is too large to be shown here, we merely show that we indeed have a collection of integers. The second example uses the `"Range"` option to import only the 11th through 15th entries in the `TBranch` named `"Service"`

and *Mathematica* returns the data stored in the `TBranch` object named *branch*.

For large data sets, it may be particularly useful to import only parts of a branch. This may be accomplished with the `"Range"` option with the value in the form of $\{m, n\}$, with m and n both being positive integers:

`Import[`*file*`,{"ROOT","TTreeData",` *tree*`,`*branch* `},"Range"→{`m`,`n`}].`

In this case, *Mathematica* imports only the m^{th} through n^{th} entries, inclusively. Examples of importing data from `TBranch` objects are shown in Fig. 3.

This feature, full and partial import of the data in a `TBranch` is arguably the most important feature of the converter. However, it is limited to `TBranch`'s that contain basic types of data, listed in Table I. The users may extend the converter to work with additional types of data, as discussed in a later section.

D. Import all branches of a `TTree` object

If we import the `"TTreeData` without specifying a `TBranch` name, the importer will iterate through the list of `TBranch` names obtained via `"TTreeMetadata"`. In addition, the *Mathematica* `Import[]`/`Export[]`framework automatically parses its arguments and we may import several `TBranch`'s (as long as they belong to the same `TTree`) in a single `Import[]`call. As with the specific `TBranch` importer, the full `TTree` importer also takes a `"Range"` option, and imports only specified entries for all the `TBranch`'s. These features are illustrated in Fig. 4.

```
In[2]:= file = "cernstaff.root"

Out[2]= cernstaff.root

In[3]:= data = Import[file, {"ROOT", "TTreeData", "T"}, "Range" → {1, 2}]

Out[3]= {{202, 530}, {15, 15}, {58, 63}, {28, 33}, {0, 0}, {10, 9},
         {13, 13}, {40, 40}, {11975, 10228}, {PS, EP}, {DE, CH}}

In[4]:= Transpose[data]

Out[4]= {{202, 15, 58, 28, 0, 10, 13, 40, 11975, PS, DE},
         {530, 15, 63, 33, 0, 9, 13, 40, 10228, EP, CH}}

In[5]:= Import[file, {"ROOT", "TTreeData", "T", {"Service", "Division", "Children"}},
          "Range" → {11, 15}]

Out[5]= {{29, 31, 29, 25, 26}, {PS, PS, PS, PS, PS}, {0, 2, 0, 0, 1}}
```

FIG. 4. Examples of importing multiple `TBranch`'s with the `"TTreeData"` element. In the first example, we import the first two entries of each branch in `TTree "T"`. (Without the `"Range"` option, we would have imported all the entries.) It is often desirable to `Transpose[]` the result, giving us the first two records in `"T"` (note that the order of the elements are the same as that given by the `"TTreeMetadata"` element. In the last example, we import some specific entries from three selected `TBranch`'s.

E. Import the data of a `TH1F` object

To import the data contained in the histogram, we use the `"TH1FData"` element:

`Import[`*file*`,{"ROOT","TH1FData",` *hist*`}]`.

The function returns a list of bin data, with entry in the form of

$$\{x_i, \Delta x_i, c_i, \Delta c_i\}, \tag{3}$$

where x_i and Δx_i are respectively the lower edge and the width of bin i, and c_i and Δc_i are respectively bin content and its error.

F. Import the data of a `TH1F` object as a histogram

We can import the `TH1F` object directly as a *Mathematica* graphics via the `"TH1FGraphics"` element:

`Import[`*file*`,{"ROOT","TH1FGraphics",` *hist*`}]`.

The function directly returns the graphics rendered using the `Histogram` function of *Mathematica* .

The `"TH1FGraphics"` also takes as options those options available to `Histogram`. The list of option names and their default values can be retrieved by evaluating `Options[Histogram]`.

G. Import the data of a TH2F object

To import the data contained in a two-dimensional histogram, we use the `"TH2FData"` element:

`Import[`*file*`,{"ROOT","TH2FData",` *hist*`}].`

The function returns a list of bin data, with entry in the form of

$$\{x_i, \Delta x_i, y_i, \Delta y_i, c_i, \Delta c_i\}, \tag{4}$$

where $\{x_i, y_i\}$ and $\{\Delta x_i, \Delta y_i\}$ are respectively the coordinate of the lower edge and widths of bin i, and c_i and Δc_i are respectively bin content and its error.

H. Import the data of a TH2F object as a histogram

We can import the TH2F object directly as a *Mathematica* graphics via the `"TH2FGraphics"` element:

`Import[`*file*`,{"ROOT","TH2FGraphics",` *hist*`}].`

The function directly returns the graphics rendered using the `Histogram3D` function of *Mathematica* .

The `"TH2FGraphics"` also takes as options those options available to `Histogram3D`. The list of option names and their default values can be retrieved by evaluating `Options[Histogram3D]`.

IV. BRIEF REMARKS ABOUT THE INTERNAL WORKINGS AND EXTENSIBILITY

In this section we offer several remarks about the internal workings of the *Mathematica* ROOT importer that may be of interest to those users interested in extending the converter.

A. Internal workings

The *Mathematica* ROOT importer is based on three technologies: CERN ROOT to extract the data inside a ROOT file, *MathLink* to transmit the extracted data to Mathematica, and using *Mathematica* to post-process the data as necessary. The ROOT and *MathLink* portions of the code are contained in the file `root_interface.tm`, and this *MathLink* template file needs to be processed by the `mprep` utility included in *Mathematica* before it can be compiled using a C++ compiler. The *MathLink* executable defines several low-level functions defined under the `ROOTImport‘` context. The input signatures of these functions are defined in `root_interface.tm`. For example,

```
ROOTImport‘importKey["/home/kenh/Documents/ROOTConvert/Examples/cernstaff.root"]
```

returns the same output as `Import["cernstaff.root",{"ROOT","Keys"}]` because the "Keys" importer essentially wraps around the low-level function. There are, however, several importance advantages using the `Import[]` call:

- the `Import[]` framework automatically passes the full path to the converter functions,
- among other checks, the `Import[]` framework checks that the file actually exists before calling the low-level functions
- the `Import[]` framework provides an easy-to-use interface that may call several low-level functions in one input and allows users to customize the form of the output

B. Extensibility

Our current implementation likely needs to be extended before reaching its full potential as a bridge between Mathematica and ROOT. Recognizing this, we have taken the effort to document both the *MathLink* and *Mathematica* portions of the code, and the existing functions may serve as templates for further development.

Occasionally, the authors may also provide fixes and new features to the code, in which case the online version of this document will be correspondingly updated.

V. CONCLUSION

This paper presents a simple implementation of a *Mathematica* importer for ROOT that is able to import some data stored in `TTree` and the histogram objects `TH1F` and `TH2F`. Taking advantage of the import/export plug-in mechanism, the importer offers an easy-to-use interface while the core While it does not capture all the possible rich and flexible data types typically stored in the ROOT files, it is nonetheless an effective and useful first step, as evident in our usage/test in importing a large ATLAS data set. The open-source nature of the project opens doors for the importer to be improved and customized, and we hope this tool will be useful to broad communities of *Mathematica* and ROOT users.

ACKNOWLEDGMENTS

We are grateful to Stephen Wolfram and Peter Overmann for dedicating the resources that made this project possible. We also thank Philippe Canal, Valeri Fine, Pavel Nevski, and Fons Rademakers for helping us with various aspects of CERN ROOT. KH would also like to thank Abdul Dakkak and William Sehorn for assisting with parts of this project.

[1] http://www.wolfram.com

[2] http://root.cern.ch/

[3] http://reference.wolfram.com/mathematica/tutorial/DevelopingAnImportConverter.html

[4] http://reference.wolfram.com/mathematica/CCompilerDriver/ref/CreateExecutable.html

[5] http://reference.wolfram.com/mathematica/ref/Import.html