University of Saskatchewan
Subatomic Physics Internal Report
SPIR 151

RLucid: Lucid to Root conversion package

Version 1.0

Ward Wurtz and Rob Pywell

September 2016, Last Update: September 22, 2016

Abstract

This report describes the RLucid package. RLucid allows convenient conversion of data files written by the LUCID data acquisition system to be translated into ROOT files for further processing and analysis.

Contents

1 Introduction

Lucid [1] is a data acquisition system used by the experimental nuclear physics group at the University of Saskatchewan. ROOT [2] is a powerful data analysis program from CERN. The RLucid package is designed to allow a user to convert Lucid data to ROOT data in a user defined way. It is meant to provide the needed flexibility for this conversion while providing tools to simplify the work.

The main motivation for this package is the *Blowfish* neutron detector array. This array is housed at Duke University in Durham, NC, U.S.A. and is a joint project between the University of Saskatchewan in Saskatoon, SK, Canada and the University of Virginia in Charlottesville, VA, U.S.A. *Blowfish* produces large Lucid data files which one may wish to import into ROOT.

The RLucid package is designed to have uses beyond the *Blowfish* neutron detector array. RLucid requires one user defined class to facilitate data conversion and analysis. This class is responsible for creating and filling ROOT tree and histogram structures. RLucid has been designed to make the filling of trees as painless as possible.

The main body of this document consists of two parts. Section 3 discusses how to use RLucid. Section 4 explains how RLucid was programmed and how it can be modified. Additionally, there are three appendices. Appendix A lists functions of two classes that a user may find useful. Appendix E describes the Lesser GNU Public License, which governs redistribution of the RLucid package. Appendix D describes work that still needs to be completed.

2 Version 1.0

This is the first true release of RLucid.

Changes from earlier versions:

- This version does *not* need to have Lucid installed on the computer on which RLucid is to be installed and run. The needed files from Lucid are included in the package.
- The included Lucid files have been modified so that RLucid will correctly read Lucid files regardless of the word size of the computer operating system running RLucid. Normally Lucid (and hence RLucid) will only run correctly on 32 bit machines. This version will run on 32 and 64 bit architectures.
- Functionality has been added so the user can access the time data was taken. (This is the same as the `LUCIDtime` built-in variable in regular Lucid Looker code.) This time is the UNIX time (in UCT (GMT)), which is included in each Lucid data record.
- This distribution includes the `BlowfishLooker` files. This is the standard User Looker for translating *Blowfish* data (see Appendix B). (The User Looker is the code that describes which Lucid data will be stored in the root file (see Section 3)).
- This distribution also includes the program `lucidview` which allows viewing of lucid data files in a human readable text form (see Appendix C).

3 User's Manual

3.1 Downloading and Installing

RLucid can be obtained from `http://nucleus.usask.ca` under the "Software" tab. RLucid can be installed on any computer that has ROOT installed. (This version has only been tested using root 5 (not 6 as yet).

Once you have downloaded RLucid, you must unzip it

```
tar -zxvf RLucid-1.0.tgz
```

This will create the directory `RLucid-1.0`. You should then enter this directory.

```
cd RLucid-1.0
```

This version of RLucid includes the Lucid files needed to make the needed Lucid library `liblucid.a` and the needed include files; they are included in the subdirectory `lucidsrc`. The `Makefile` will automatically build the Lucid library as needed. It is prudent, after first unpacking the RLucid files, to ensure that there are no object files or libraries remaining from a different architecture computer. Therefore do a "clean" by

```
make clean
```

When you are ready to compile RLucid simply type

```
make
```

and RLucid will be compiled. During compilation of the Lucid files there may be a few warning error messages. These may be ignored safely. (The Lucid files include some legacy code that is not needed. Later versions will hopefully get rid of these.)

After compiling, there should be two new files of interests. The first, `libRLucid.so`, is a shared library containing the complete RLucid application. The second, `rxlucid`, is a sample application that demonstrates how to include RLucid in your stand-alone ROOT applications.

The library `libRlucid.so`, rootmap file `libRlucid.rootmap` and the needed header files can be installed in your ROOT directory by typing

```
make install
```

after you have ensured that you have permission to write to the ROOT directory on your machine.

3.2 Loading RLucid Libraries Into ROOT

We start by discussing how to load RLucid into ROOT. If you are using a version of ROOT before 5.16, read section 3.2.1 and then move on to 3.3. If you are using version 5.16 or later, read section 3.2.2 and then move on to 3.3.

3.2.1 For ROOT Versions Before 5.16

With the RLucid source code, there is a file called `rootlogon.C`. Starting ROOT in this directory by typing

```
root
```

will cause ROOT to run `rootlogon.C` and execute `rootlogon()` which is shown below.

```
//automatically run when root starts
rootlogon()
{
     //Load needed libraries
     gSystem->Load("libRLucid");
}
```

The command `gSystem->Load("libRLucid")` attempts to load `libRLucid.so` into ROOT.

3.2.2 For ROOT Versions Including and After 5.16

Starting ROOT in any directory by typing

```
root
```

will cause ROOT to read the `libRLucid.rootmap` file installed into ROOT's library directory with `libRLucid.so`. This file tells ROOT what libraries must be loaded in order to load the `libRLucid.so` library. All libraries will be loaded when one of the classes in `libRLucid.so` is called upon. Any warnings about libraries being already loaded can be safely ignored.

3.3 Running RLucid Graphically

Once ROOT is running, we can create a graphical user interface (GUI) to control RLucid.

```
root [0] new TRXLucid;
```

The RXLucid window will be created and should appear as in figure 1. To begin using RXLucid, select

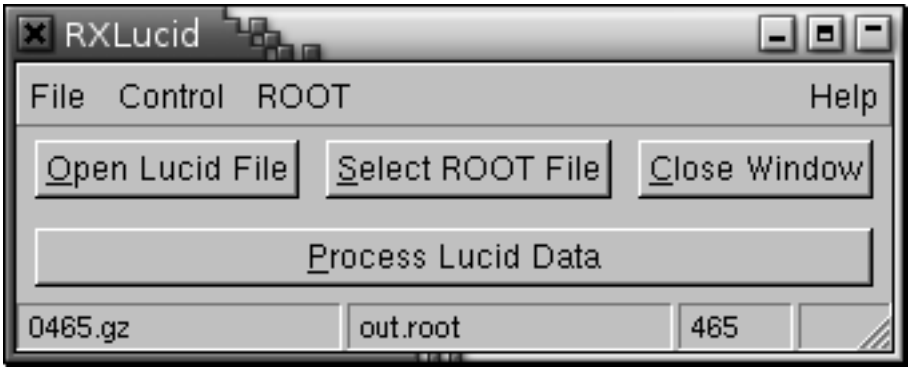

Figure 1: RXLucid Graphical Interface

a Lucid file to open by clicking on the `Open Lucid File` button. RLucid can open uncompressed Lucid files and Lucid files compressed with gzip or bzip2 that end in the extension `*.gz` and `*.bz2` respectively. Select a ROOT file to save the data by pressing the `Select ROOT File` button and typing in the name of the ROOT file. You will notice in figure 1 that the name of the Lucid file appears on the left part of the status bar and the name of the ROOT file appears on the next space. The run number appears on the third space after the data has been processed. If we wish to append data to a ROOT file, as will be discussed in section 3.4, an `(A)` will appear after the ROOT file name.

Pressing the button labelled `Process Lucid Data` will start the conversion and analysis of the Lucid data. Unfortunately, this will cause RLucid to block and become unresponsive. This problem could be fixed by using threads and ROOT does support threads. However, many ROOT classes, such as TTree in particular, are not thread safe. This results in unpredictable behaviour if a tree is accessed by more than one thread, as one might wish to do when filling trees from Lucid files. Therefore, RLucid does not support threads but provides a printout about every second to indicate that it is processing and has not crashed or entered an infinite loop:

```
TRLucidControl::Process() -> Lucid File: lucid-data.gz
TRLucidControl::Process() -> ROOT File: root-data.root
TRLucidControl::Process() -> User Looker: Copy User Looker
Lucid Event:     66000   120000   198000   277000   356000   435000
Lucid Event:    514000   563000   641000   719000   797000   874000
Lucid Event:    951000  1027000
Done Reading Lucid Data
```

After processing is complete a `(C)` will appear after the filename. This signifies that if we process another Lucid file, RLucid will continue adding the data onto the end of the current ROOT file.

Notice in the above output the line `User Looker: Copy User Looker`. A user looker is an object that a user can write in order to instruct RLucid on how to convert the Lucid data into ROOT data. A user looker is any class that inherits the pure virtual class `TVUserLooker`. RLucid comes with two user lookers built in. The first, `TCopyUserLooker` makes a direct copy of the Lucid data without any interpretation. The second, `TDummyUserLooker` does no conversion and is used for debugging purposes. We will discuss user lookers in depth later. Let us see what `TCopyUserLooker` has output.

By selecting the menu item `ROOT` then `Browser` in the RXLucid window, we can open the ROOT browser. From the Browser we can view the contents of the ROOT file by choosing `ROOT Files` and then our file name. Notice that for each Lucid event, `TCopyUserLooker` has created a tree and for each Lucid variable a branch. Lucid variables that contained arrays are stored in the branches as arrays due to the performance gain realized over storing different channels in different branches.

Now that we have a basic grasp on using the RXLucid GUI, we will want to explore some of the more advanced features of the GUI and start building user looker classes.

3.4 Graphical User Interface Features

Returning to the main RXLucid window, we notice a menu called `File`. In this menu there are many options and these are shown in figure 2. The items `Open Lucid File` and `Select ROOT File` perform the same operations as their respective buttons. The item `Input From Program` allows us to get Lucid input from a program. The item `Options` contains many useful options. Figure 3 shows the options window There are three sets of options in this window.

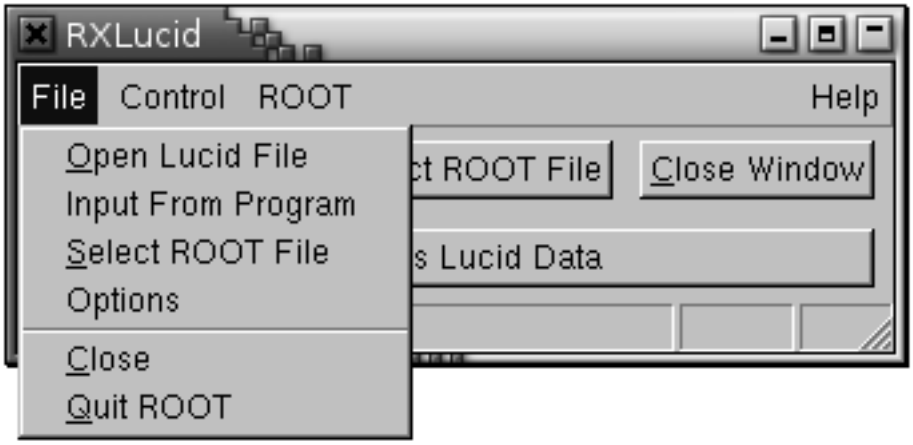

Figure 2: File Menu

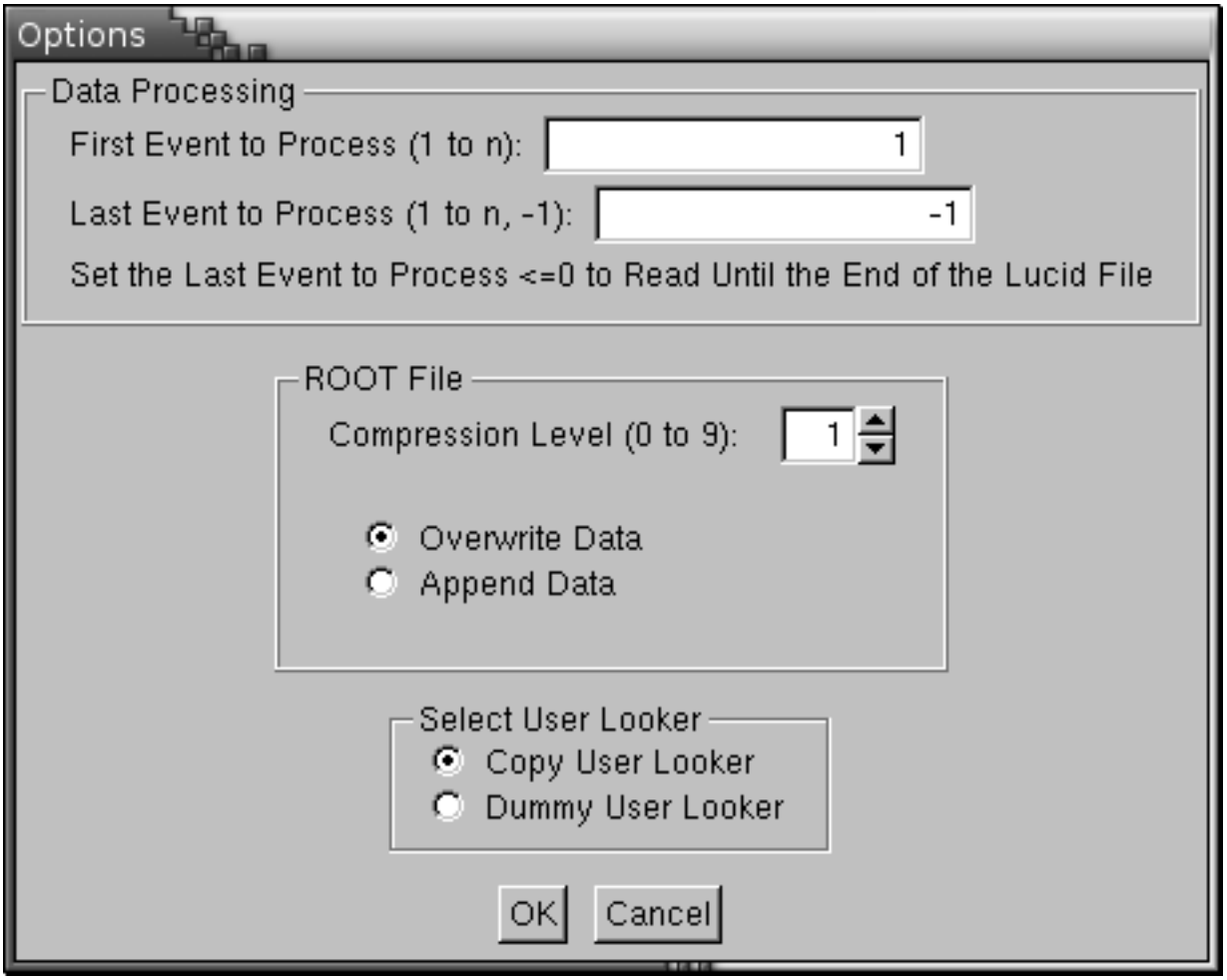

Figure 3: Options Window

The first set is data processing options. These options control the event that Lucid starts and stops processing data. These are useful if a Lucid file contains a section with bad data or for limiting the amount of data read for debugging purposes. Setting the stop event to -1 will cause RLucid to read the entire file.

The second set of data options controls the compression level to use when compressing the data in the ROOT file. It is recommended that we use a compression level greater than 0, which is no compression at all. However, the principle of diminishing returns suggests that high levels of compression are not practical due to the increased processor usage. A compression level of 1 is usually sufficient. We can also choose whether we want to overwrite or append data to the ROOT file if it already exists.

The third set of options allows us to chose what user looker to use. The two default user lookers are show in figure 3. If we had added our own user looker to RLucid, it would appear here.

It is possible to add a user looker to RLucid using the graphical interface. In the drop-down menu of the main RLucid GUI select `Control` as in figure 4. Selection `Load User Looker` will bring up a

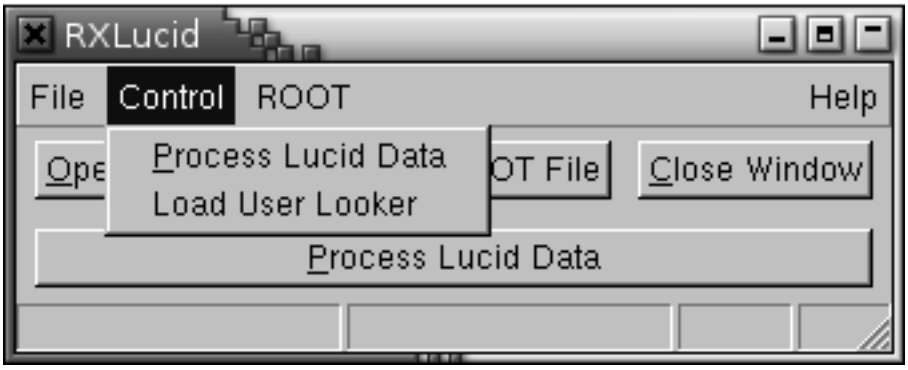

Figure 4: Control Menu

file dialogue box. You can select either a C++ source file or a `.so` shared library that was previously compiled by ROOT. For example, select the file `TExampleUserLooker.cxx` which is supplied with RLucid for instructional purposes.

```
TRXLucid::LoadUserLooker() -> Will attempt to compile and load file
     /home/me/mydir/TExampleUserLooker.cxx...
Info in <TUnixSystem::ACLiC>: creating shared library
     /home/me/mydir/TExampleUserLooker_cxx.so
TRXLucid::LoadUserLooker() -> Compilation of /home/me/mydir/TExampleUserLooker.cxx
     succeeded.
TRXLucid::LoadUserLooker() -> Will now try to create a new user looker
     TExampleUserLooker...
TRXLucid::LoadUserLooker() -> Success: Example User Looker
```

and the user looker is now loaded and ready to go. Note that RLucid assumes that the name of the user looker class is the same as the name of the file without the `.cxx` extension.

If ROOT is not able to compile the user looker or RLucid is not able to add the user looker to its list of user lookers, an error message will be displayed and the user looker will not be added. Note that once a user looker is added to RLucid, it cannot be removed without ending the ROOT session. Therefore, this method of loading user lookers is not good for debugging but is meant as a convenient

way to add known good user lookers through the graphical interface. See section 3.6.7 for more details on how to load a user looker without the GUI.

The user looker will now be selectable from the options menu and `Example User Looker` will appear above `Copy User Looker` in figure 3. The new user looker will be selected by default.

3.5 Direct Control of RLucid

For those who do not wish to use the RXLucid GUI, one can control RLucid directly. The GUI classes are simply graphical tools to control the class `TRLucidControl`. This class is responsible for coordinating the activity of all other classes.

To access the `TRLucidControl` class, we must call the function
`static TRLucidControl * GetPointer(void)` as in the following example.

```
TRLucidControl * fLucidControl = TRLucidControl::GetPointer();
```

This class has a private constructor to ensure that only one instance of this class can exist at any time. The static function `GetPointer()` can be called at any time to get a pointer to the one-and-only instance of `TRLucidControl`. The first call to `GetPointer()` will call the constructor and all subsequent calls will simply return a pointer to this class.

Once we have the pointer, we are able to perform all the functions the GUI performs. The functions `SetLucidFileName(const char *)` and `SetROOTFileName(const char *)` can be used to set the Lucid and ROOT filenames respectively. The start and stop events can be set by calling the functions `SetStartEvent(Long64_t)` and `SetStopEvent(Long64_t)` respectively. You can start processing the data by calling the function `Process()`. See appendix A.1 or the file `TRLucidControl.h` for a full list of the available functions.

3.6 Creating a User Looker Class

The class `TVUserLooker` defines an interface for the user looker through five pure virtual functions. The control class, `TRLucidControl`, uses the five pure virtual functions to perform the user defined conversion and analysis of the Lucid data. Any class that inherits `TVUserLooker` is forced to define these five functions and can be passed to `TRLucidControl` through the function `static void AddUserLooker(TVUserLooker * ul)`. We first discuss how the Lucid data is made available for the `TVUserLooker` class through the `TRLucidEvent` class. The five pure virtual functions and their usage is then discussed. We finish by discussing how to compile and load a user looker from the ROOT command line or a script.

3.6.1 The Class `TRLucidEvent`

The class `TRLucidEvent` is the stepping stone between Lucid and ROOT. It would not be appropriate for a user looker class to access the Lucid records directly; therefore `TRLucidEvent` provides the needed middle ground.

The class `TRLucidEvent` stores the Lucid data in a series of buffers. The buffers are defined so that their address will not change, even when processing multiple Lucid files, as long as the Lucid symbol table remains the same. Therefore, a ROOT tree can use this buffer directly as its input buffer. A pointer to an array of short integers can be obtained by using the function `GetShortPointer()`

which can take either the variable name or an index number as an argument. Likewise, there exist the functions `GetLongPointer()`, `GetFloatPointer()`, `GetDoublePointer()` and `GetCharPointer()` for getting pointers to their respective data types. See appendix A.2 or the file `TRLucidEvent.h` for a full list of the functions available for manipulating events.

The following is a simplified excerpt from `TCopyUserLooker` and shows how to create a branch on a tree named `theTree` using information obtained from `TRLucidEvent`:

```
//for all short integers in the Lucid Event
TString name, leafList;
Int_t size;
Short_t * shortPtr;
Int_t ctr = 0;
while(1)
{
    //try to get the next short pointer.  If none exists, we are done
    shortPtr = theEvent->GetShortPointer(ctr);
    if(shortPtr == 0){ break; }

    //create the name and the leaf list (which is based on the name)
    name = theEvent->GetShortName(ctr);
    size = theEvent->GetShortSize(ctr);
    leafList = name;
    if(size > 1) {  //[size] for arrays
        leafList += "[";
        leafList += size;
        leafList += "]"; }
    leafList += "/S"; //16 bit, signed integer

    //Create a new branch in this tree to hold all the data in this short.
    //One branch can hold an array of data.  This is preferable over having many
    //branches for performance reasons.
    theTree->Branch(name,shortPtr,leafList);

     ctr ++;
}
```

Once the branch has been created it can be filled by simply using the `Fill()` function.

```
theTree->Fill();
```

For more information on this example, see the file `TCopyUserLooker.cxx`.

The above example is only one way to create and fill branches of a tree. The `TRLucidEvent` class provides functions that allow for a variety of operations. The `TRLucidEvent` class allows us to do the data conversion and analysis how we want. It's functions are listed in appendix A.2.

3.6.2 `TVUserLooker::InitROOTFile()`

The first function to be called in a user looker class is `InitROOTFile()`. This function is called once for each ROOT file created by RLucid.

The new ROOT file is created by `TRLucidControl` when the `Process()` function is called or `Process Lucid Data` button is pressed. Right after, `InitROOTFile()` is called. The purpose of `InitROOTFile()` is to create objects, such as trees and histograms, that will be saved to the newly opened ROOT file, which can be accessed by using the `gFile` global pointer.

Notice that while it is mandatory to define the function, it need not perform any operation. The class `TCopyUserLooker` does not create trees here since it does not have previous knowledge of the Lucid symbol table.

3.6.3 `TVUserLooker::InitLucidFile()`

The `InitLucidFile(TRLucidEvent * eventArray[], Int_t noEvents, Int_t runNumber)` function is called whenever a new Lucid file is about to be processed. This function is called right after the symbol table is read and is passed an array of events, the number of events in the array and the Lucid run number. Note that `InitROOTFile()` will be called before `InitLucidFile()` for a newly opened or created ROOT file, but only `InitLucidFile()` will be called if we are continuing to add data to the current ROOT file for many Lucid files.

Since the array `eventArray` contains pointers to all the events found in the Lucid symbol table, this function can be used to set up ROOT trees and histograms when the symbol table is not known beforehand. The user looker `TCopyUserLooker` makes use of this function.

3.6.4 `TVUserLooker::ProcessEvent()`

The function `ProcessEvent(TRLucidEvent * theEvent)` performs the data conversion and analysis. It takes as arguments a pointer to the event to be processed. This is the function that should call the `Fill()` function on the appropriate trees and histograms. If ADC or TDC information needs to be uncompressed, this is where it should be done.

3.6.5 `TVUserLooker::FinishRun()`

The function `FinishRun()` is used to save the trees and histograms to the file after the Lucid run has finished. This is a good place to call the `Write()` function in one of two ways. First, we can call `Write()` individually for each object we wish to write to the ROOT file. Second, we can call

```
gFile->Write("",TObject::kOverwrite);
```

which will write all trees and histograms to the file.

3.6.6 `TVUserLooker::GetULName()`

The function `GetULName()` is used to return a string that identifies the user looker to human beings. For instance the `GetULName()` function of `TCopyUserLooker` returns `"Copy User Looker"`. This function should not be confused with `GetName()` which is a member of `TObject`. For `TCopyUserLooker` the function `GetName()` returns `"TCopyUserLooker"`.

3.6.7 Loading a User Looker

We have already seen how to load a user looker using the GUI. However, if we are not using the GUI or debugging compile errors, we may not wish to load the user looker manually. In this section, we will examine how to load the example user looker `TExampleUserLooker` that came with RLucid.

The first thing we must do is load and compile the example user looker. At the ROOT prompt we can use the `.L +` command.

```
root [0] .L TExampleUserLooker.cxx+
Info in <TUnixSystem::ACLiC>: creating shared library
     /home/me/mydir/./TExampleUserLooker_cxx.so
```

Thus we have successfully compiled and loaded the user looker into ROOT. If errors occurred during compilation, the user looker will not be loaded. Alternatively the command

```
root [0] gSystem->CompileMacro("TExampleUserLooker.cxx");
```

can be used to compile and load the user looker. This command is easier to put into a script, such as `rootlogon.C`, than the `.L +` command.

Next we need to make RLucid aware of this user looker.

```
TRLucidControl::AddUserLooker(new TExampleUserLooker);
```

Now the user looker has been added to the list of user lookers maintained by RLucid. Note that once it has been added, it cannot be removed until the ROOT session ends.

When an instance of `TRLucidControl` has been created, the last added user looker will be the one used by default. To change the user looker, use the function `SetCurrentUserLooker(Int_t no)` where `no` is the number of the user looker. The user lookers with numbers 0 and 1 are `TDummyUserLooker` and `TCopyUserLooker` respectively. Therefore, our new user looker has number 2. See appendix A.1 or the file `TRLucidControl.h` for more functions concerning the adding of user lookers.

4 Programmer's Manual

The RLucid program can be divided into three layers: The GUI layer, the control layer and the Lucid layer. These layers are the subjects of sections 4.1, 4.2 and 4.3 respectively. Figure 5 shows how the classes are divided amongst the three layers.

There is a class not depicted in figure 5 and that is the `TRLucidEvent` class. Instances of this class are created in the Lucid layer and filled with data from the Lucid file. The instances of `TRLucidEvent` are then passed into the control layer so that the user looker can use the Lucid data to create ROOT data. This class is described in section 4.4.

The file `liblucid.a` featured in figure 5 is the main Lucid library and is supplied by Lucid. It is used by the Lucid layer to extract the data from a Lucid file.

4.1 The GUI Layer

The GUI layer exists to provide a graphical interface on top of the control layer. The class `TRXLucid` is the main GUI class and `TRLucidOptions`, `TRLucidProgram` and `TRLucidAbout` are only accessed

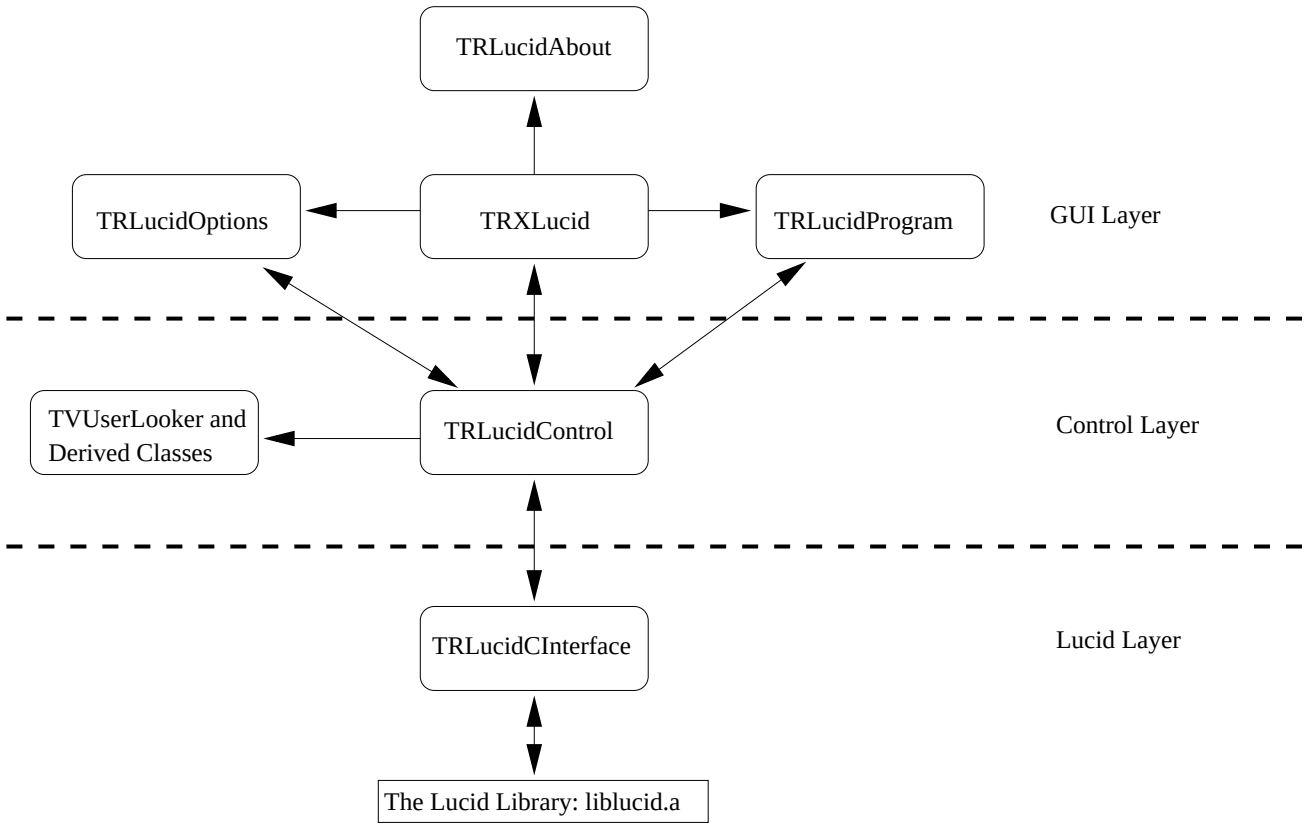

Figure 5: Relationships Between the RLucid Classes Excluding the Class `TRLucidEvent`

through it. The GUI classes use the standard ROOT GUI system for simplicity. Buttons are connected to publicly available functions and the menu items in `TRXLucid` are controlled using messages by the `ProcessMessages()` function. `TRXLucid` also has the ability to handle shortcut keys, handled by the `HandleKey()` function.

The class `TRLucidAbout` does not take any input but shows information about RLucid. It is the most simple graphical interface and has one button that closes it.

The class `TRLucidProgram` provides a text box to enter a program name and 'OK' and 'Cancel' buttons. The program name is sent to `TRLucidControl` when the 'OK' button is pressed and RLucid will get its data from a program.

The class `TRLucidOptions` has several entry fields which can set several options. These options all correspond to options available in `TRLucidControl`. Pressing the 'OK' button will cause the options to be sent to `TRLucidControl` and used when processing data.

4.2 The Control Layer

The control layer consists of two parts: The classes `TRLucidControl` and the user looker. The user looker is described in the user's manual, section 3.6 as it must be defined by the user. The class `TRLucidControl` will be described in this section and its public functions are listed in appendix A.1.

Most of the function that a user will use are associated with setting and checking options. The only exception to this is the function `Process()` which takes these options and uses them to process

the Lucid file and create the new ROOT file. With the exception of adding a user looker, the options are set and checked using functions that read and write class variables. These class variables store the information so that it can be used in the `Process()` function.

The variables for storing user lookers are slightly different. The array of user lookers is stored as static, which means that it will not be created or destroyed with the `TRLucidControl` object. This allows one to load a user looker before any instance of `TRLucidControl` is created and the list of user lookers will survive even if the instance of `TRLucidControl` is destroyed.

The constructor for `TRLuicdControl` is private. In order to ensure that there can exist only one instance of `TRLucidControl`, one must call the `GetPointer()` function. If an instance of `TRLucidControl` does not exist, `GetPointer()` will create one. If an instance does exist, `GetPointer()` will return a pointer to this instance. It is because of the old C Lucid library that we can have only one instance of `TRLucidControl`. Since the Lucid library has several internal variables and is not object oriented, a ROOT session can only access one Lucid file at a time. Having only one instance of `TRLucidControl` makes this easy to enforce.

The class `TRLucidControl` creates an instance of `TRLucidCInterface`. All of the communication with the Lucid C library is done with `TRLucidCInterface` and `TRLucidControl` controls an instance of `TRLucidCInterface`.

The `Process()` function of `TRLucidControl` attempts to open a Lucid file via `TRLucidCInterface` and read its symbol table. If it is successful, it will then open a ROOT file or prepare to continue writing to a previously opened ROOT file. It calls the user looker function `InitROOTFile()` if we have a new ROOT file and it always calls `InitLucidFile()`.

After the two files have been initialised, `Process()` enters a loop. It will stay in this loop until either `TRLucidCInterface` reports that it is at the end of the Lucid file or we reach the last desired event, as set by the user.

The first action of the loop is to get an event from the Lucid file through `TRLucidCInterface`. If this event is within the range of events we wish to convert, it is sent to the user looker's function. Unfortunately, in order to find the first event to process, assuming it is not the first event of the Lucid file, one must read all previous events in the Lucid file. If the event is to be read, it is then passed to the user looker function `ProcessEvent()`.

Once we are done reading the Lucid file, the loop exits and `Process()` calls the user looker function `EndRun()`.

The control layer manages the work of converting the Lucid data into ROOT data. It is the class `TRLucidCInterface`, to be described in the next section, that reads the Lucid data. Also, it is the user looker class that saves the data to the ROOT file. The class `TRLucidControl` passes information from one to the other.

4.3 The Lucid Layer

In the Lucid layer there is the class `TRLucidCInterface` which provides a C++ interface to the Lucid C library `liblucid.a`. The Lucid data format is designed to be fault tolerant for experimental situations. This is not the most convenient for data analysis, so we use this class to hide the details of Lucid's file format from ROOT. Thus, ROOT does not know about this class and there is no dictionary generated for it. This class should only be accessed by `TRLucidControl`. While `TRLucidControl` managed the Lucid-to-ROOT conversion, this class is responsible only for organizing the Lucid data without understanding the ROOT framework.

Lucid can get data from either a file or a program and TRLucidCInterface comes with the functions OpenFile() and OpenProgram() which open a file and program respectively. To close a file, TRLucidCInterface has the function Close(). Closing a file is straight forward. However, Lucid does not handle closing a program well. Therefore, Close() closes the program itself, then creates a new file descriptor to /dev/null for Lucid to close. Lucid closes this file descriptor without noticing that it has been tricked. For more discussion on this, please see the comments in the function TRLucidCIntergace::Close() in the file TRLucidCInterface.cxx. This trick, while certainly not the best programming practice, has been extensively tested and been verified to work.

The function ReadSymbolTable() processes the first few Lucid records until it reads the symbol table and encounters a 'Begin Run' record. When a symbol record is read, the function calls the private, helper function ReadRecordSYMBOL(). The function ReadRecordSYMBOL() examines the record and creates a TRLucidEvent object to correspond with this event. The TRLucidEvent object is told how many variables of each type are stored in the event and what their names are. It is then added to the appropriate lists. The first list contains the various TRLucidEvent objects sorted by their event type number. This list is used for filling the events with data when reading records from the Lucid file. The second list stores the TRLucidEvent objects in order and is passed to the user looker's InitLucidFile() function.

Note that when processing several Lucid files, they must have the same symbol table. The function ReadSymbolTable() checks to ensure that the symbol tables are the same. If not, it outputs a warning. In order to process multiple files with different symbol tables, use the function TRLucidControl::ResetSymbolTable() in the control layer.

The function ReadEvent() reads an event from the Lucid file and returns it to TRLucidControl. First, ReadEvent() checks to see if there are any events still in the current Lucid record. If not, it opens the next record in the Lucid file. Once ReadEvent() has a record with unread events, it calls the private, helper function ReadRecordDATA(). The function ReadRecordDATA() reads the next event from the record and saves the information in the appropriate instance of TRLucidEvent. When ReadEvent() encounters an 'End Run' record or encounters an unexpected 'End of File', it returns a value to let TRLucidControl know that it is done processing data.

It is important to note that during the compile process the Makefile extracts several objects from the statically linked library liblucid.a. These objects are then placed in the libRLucid.so library so that they may be accessed easily when the libRLucid.so library is loaded. Thus, there is no need to try to load liblucid.a into ROOT.

4.4 The Class TRLucidEvent

The class TRLucidEvent is used to pass data from the Lucid layer to the control layer. The instances of TRLucidEvent are created and maintained by TRLucidCInterface. Thus, TRLucidCInterface is given the special distinction of being a friend class to TRLucidEvent. Only TRLucidCInterface can set data in a TRLucidEvent object, but any class can read it.

When creating a new TRLucidEvent object, TRLucidCInterface first tells TRLucidEvent how many of each variable type is in the event through SetNumberVariables() so that it can create the memory structures required to store them. Then TRLucidCInterface informs the TRLucidEvent object of the variables using the functions AddShort(), AddLong(), AddFloat(), AddDouble() and AddChar(). For each variable in the symbol table, TRLucidCInterface will call the appropriate 'Add' function once. Then TRLucidEvent will create an instance of TRLucidEventInfo, which is

a structure for keeping track of the information concerning variables in the event. The instances of `TRLucidEventInfo` are used to extract the Lucid data by the user looker. When there are no more variables, the `DoneAdd()` function is called.

The class `TRLucidEvent` stores information in a series of buffers. Once `TRLucidCInterface` is done reading the symbol table, these buffers will reside in memory at the same location as long as the symbol table remains the same. The data that is not compressed is stored all in the same buffer with different pointers pointing to the correct places. The compressed data, since it has a variable length, gets its own buffer. It is copied into this buffer so that the position of the buffer in memory never changes. The `SetData()` function of `TRLucidEvent` is used by `TRLucidCInterface` to set the data. It copies the fixed length data directly into the buffer and each bit of compressed data into its buffer.

Once the `TRLucidEvent` object has been filled with data, it is passed to `TRLucidControl` which passes it to the user looker. Data can then be easily extracted from it by using the functions found in appendix A.2.

A Available Functions

This appendix is to be used as a reference for the most useful functions of RLucid. Functions that use the `inline` keyword are not denoted as such.

A.1 Class `TRLucidControl`

Tables 1 and 2 describe the functions of the class `TRLucidControl`. For more information, see the header file `TRLucidControl.h`.

Table 1: Public functions for the class `TRLucidControl` (more in table 2)

Function	Description
`static TRLucidControl * GetPointer()`	Get the pointer to the one-and-only instance of `TRLucidControl`.
`virtual ~TRLucidControl()`	Destructor.
`static void AddUserLooker(TVUserLooker * ul)`	Adds a user looker to the static user looker list.
`static const char * GetUserLookerName(Int_t no)`	Get the name of the user looker stored in the list with index `no`.
`static Int_t GetNoUserLookers()`	Get the number of available user lookers.
`Int_t SetCurrentUserLooker(Int_t no)`	Set the current user looker by index number from the list of user lookers.
`const char * GetCurrentUserLookerName()`	Get the name of the current user looker.
`Int_t GetCurrentUserLookerNumber()`	Get the number of the current user looker.
`Int_t Process()`	Convert the Lucid file into a ROOT file.
`void SetLucidFileName(const char * fn)`	Set the file name for the Lucid data.
`void SetLucidFileName(TString fn)`	Set the file name for the Lucid data.
`TString GetLucidFileName()`	Get the file name for the Lucid data.
`TString GetRelLucidFileName()`	Get the relative file name for the Lucid data.
`void SetROOTFileName(const char * fn)`	Set the file name for the ROOT data.
`void SetROOTFileName(TString fn)`	Set the file name for the ROOT data.
`TString GetROOTFileName()`	Get the file name for the ROOT data.
`TString GetRelROOTFileName ()`	Get the relative file name for the ROOT data.

Table 2: Public functions for the class TRLucidControl (continued from table 1)

Function	Description
`Bool_t OverwriteROOT()`	Check to see if we are set to overwrite the ROOT file.
`Bool_t AppendROOT()`	Check to see if we are set to append data to the ROOT file.
`Bool_t ContinueROOT()`	Check to see if we are set to continue writing to the ROOT file.
`void SetOverwriteROOT()`	Set to overwrite any existing ROOT files.
`void SetAppendROOT()`	Set to append data to any existing ROOT files.
`void SetLucidProgram(const char * command)`	Set the command used to get Lucid data.
`void SetLucidProgram(TString cm)`	Set the command used to get Lucid data.
`TString GetLucidProgram()`	Get the command used to get Lucid data.
`Bool_t LucidDataFromProgram()`	Returns `kTRUE` if we are getting data from a program, `kFALSE` if from a file.
`void SetCompression(Int_t comp)`	Set the compression level for the ROOT file.
`Int_t GetCompression()`	Get the compression level for the ROOT file.
`Int_t GetRunNumber()`	Get the Lucid run number.
`Long64_t GetStartEvent()`	Get the Lucid event to start reading data.
`Long64_t GetStopEvent()`	Get the Lucid event to stop reading data.
`void SetStartEvent(Long64_t start)`	Set the Lucid event to start reading data.
`void SetStopEvent (Long64_t stop)`	Set the Lucid event to stop reading data.
`void ResetSymbolTable()`	Reset the symbol table.

A.2 Class TRLucidEvent

Table 3 describes the functions of the class TRLucidControl. For more information, see the header file TRLucidControl.h. Also the « operator is overloaded for this class.

Table 3: Public functions for the class TRLucidEvent

Function			Description
Int_t	GetType()		Get the event's type number.
TString	GetName()		Get the event's name.
Long64_t	GetEventNumber()		Get the number of this event.
ULong64_t	GetLucidTime()		Get the Lucid time (from last Lucid record).
void	Print	(ostream *os)	Print useful event information.
Short_t *	GetShortPointer	(TString name)	A pointer to the named array of shorts.
Int_t *	GetLongPointer	(TString name)	A pointer to the named array of longs.†
Float_t *	GetFloatPointer	(TString name)	A pointer to the named array of floats.
Char_t *	GetCharPointer	(TString name)	A pointer to the named array of chars.
Short_t *	GetShortPointer	(Int_t numb)	A pointer to an array of shorts by number.
Int_t *	GetLongPointer	(Int_t numb)	A pointer to an array of longs by number.†
Float_t *	GetFloatPointer	(Int_t numb)	A pointer to an array of floats by number.
Char_t *	GetCharPointer	(Int_t numb)	A pointer to an array of chars by number.
Int_t	GetShortSize	(Int_t numb)	The size of an array of shorts (in elements).
Int_t	GetLongSize	(Int_t numb)	The size of an array of longs (in elements).
Int_t	GetFloatSize	(Int_t numb)	The size of an array of floats (in elements).
Int_t	GetCharSize	(Int_t numb)	The size of an array of chars (in elements).
TString	GetShortName	(Int_t numb)	The name of an array of shorts by number.
TString	GetLongName	(Int_t numb)	The name of an array of longs by number.
TString	GetFloatName	(Int_t numb)	The name of an array of floats by number.
TString	GetCharName	(Int_t numb)	The name of an array of chars by number.

† Note the apparent inconsistent data type. The 'Long' in GetLongPointer refers to the Lucid data type long within the Lucid data file, which is 4 bytes. A Root Int_t appears to always be 4 bytes regardless of whether Root is running on a 32 bit or 64 bit operating system. A Root Long_t can be different on different operating systems. See Appendix D.

B BlowfishLooker

Included in the RLucid package is a subdirectory named `BlowfishLooker` which includes the files `BlowfishLooker.cxx` and `BlowfishLooker.h`. This is the current version of the user looker used for translating Lucid data from *Blowfish* experiments. This may need to be modified for specific experiments or analysis needs.

To use it you need to copy the two files into the directory where you intend to run Root for data analysis. After starting `rxlucid` you can load the `BlowfishLooker` by using the `Load User Looker` function which in in the `Control` menu.

Alternatively you can have it included automatically when you start Root by including it in the `rootlogon.C` file. To do this it needs to be compiled. e.g. in Root by typing

```
root [1] .L BlowfishLooker.cxx+
```

Then the required commands can be included in the `rootlogon.C` file. e.g.

```
rootlogon()
{
   //Load needed libraries
   gSystem->Load("libRLucid");

   //Add the User Looker for the Blowfish-ROOT Analysis Package
   gSystem->Load("BlowfishLooker_cxx.so");
   TRLucidControl::AddUserLooker( new BlowfishLooker );
}
```

Then, after starting Root, typing

```
root [2] new TRXLucid
```

will start the RXLucid Gui.

The `BlowfishLooker` needs a few additional files to run. In the directory you will run Root (and `BlowfishLooker`) there needs to be a subdirectory named `param`. In that subdirectory there needs to be a few subdirectories in which some parameters files are stored.

In the directory `param/Cells` there must be a file named `Cell_Map.dat` which will be read by `BlowfishLooker` and contains infomation that allows translation of module addresses to cell numbers. This file is generated from the information in a cell map file which lists all the cells and which CFD, Scalers, VME ADC and TDC channels, etc they are connected to. The latest version of that file is called `cellmap-June2015.dat`. The `Cell_Map.dat` file can be generated from the `cellmap-June2015.dat` file by using the Perl script `makecellmap.pl`.

There needs to exist a directory `param/Ped`. During the processing of a Lucid file `BlowfishLooker` will automatically generate a set of pedestals for the run and will store a human readable pedestal file in the `param/Ped` directory.

A copy of the subdirectory `param` is included in the distribution.

C Lucidview

Lucidview is a program that print the contents of a Lucid data file in a human readable form. It is included in the subdirectory `lucidsrc/util`.

`lucidview` may be built by (after building `RLucid`) going into that directory and typing `make`. i.e.

```
>cd lucidsrc/util
>make depend
>make
```

If desired, typing "`make install`" will copy the `lucidview` program to the user's `bin` directory (i.e. `$(HOME)/bin`).

To run type:

```
> lucidview <filename>
```

where `<filename>` is the name of the Lucid data file. If no filename is given `lucidview` will expect input from the standard input. There are several options to `lucidview`. Typing `lucidview -o` will print a list of the options. The options allow filtering of the output for just the data records you want to see. The default is to print the full contents of the Lucid file. Since most files are long it is usual to redirect the output to a pager like `less`. Most Lucid data files are compressed with `gzip` so they must be uncompressed before viewing with `lucidview`. e.g. to view the contents of the data file `0535.gz` do the following...

```
> zcat 0525.gz | lucidview | less
```

D Work Remaining

- Decompress compressed Lucid data with channel/value format. At present only the simplest form of data compression available in Lucid is supported. Fortunately that is all we use for data from the VME modules.
- For speed and efficiency the underlying Lucid data structure present in the Lucid data file is not completely shielded from the end user. Specifically, in the user looker only a pointer to the Lucid data is passed to the user. Therefore the user must know the length of the variables in memory. For example, a `long` in the Lucid data is always 4 bytes while in Root a `Long_t` can be 4 or 8 bytes depending on the computer architecture. Therefore a `GetLongPointer()` must be copied to a `Int_t *` pointer which in root appears to be always 4 bytes. This is noted in the documentation and in the `BlowfishLooker` files. It would be nice if the end user, who is writing the "User Looker" root code did not need to know the structure of how the original Lucid data is stored. To do this however would require a re-write of some of the RLucid code to copy variables one-by-one. (This would incidentally result in slightly slower executing code.)
- Test with ROOT 6.
- Test, test and test some more.

E Lesser GNU Public License

GNU LESSER GENERAL PUBLIC LICENSE
Version 2.1, February 1999

[This is the first released version of the Lesser GPL. It also counts as the successor of the GNU Library Public License, version 2, hence the version number 2.1.]

Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public Licenses are intended to guarantee your freedom to share and change free software–to make sure the software is free for all its users.

This license, the Lesser General Public License, applies to some specially designated software packages–typically libraries–of the Free Software Foundation and other authors who decide to use it. You can use it too, but we suggest you first think carefully about whether this license or the ordinary General Public License is the better strategy to use in any particular case, based on the explanations below.

When we speak of free software, we are referring to freedom of use, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish); that you receive source code or can get it if you want it; that you can change the software and use pieces of it in new free programs; and that you are informed that you can do these things.

To protect your rights, we need to make restrictions that forbid distributors to deny you these rights or to ask you to surrender these rights. These restrictions translate to certain responsibilities for you if you distribute copies of the library or if you modify it.

For example, if you distribute copies of the library, whether gratis or for a fee, you must give the recipients all the rights that we gave you. You must make sure that they, too, receive or can get the source code. If you link other code with the library, you must provide complete object files to the recipients, so that they can relink them with the library after making changes to the library and recompiling it. And you must show them these terms so they know their rights.

We protect your rights with a two-step method: (1) we copyright the library, and (2) we offer you this license, which gives you legal permission to copy, distribute and/or modify the library.

To protect each distributor, we want to make it very clear that there is no warranty for the free library. Also, if the library is modified by someone else and passed on, the recipients should know that what they have is not the original version, so that the original author's reputation will not be affected by problems that might be introduced by others.

Finally, software patents pose a constant threat to the existence of any free program. We wish to make sure that a company cannot effectively restrict the users of a free program by obtaining a restrictive license from a patent holder. Therefore, we insist that any patent license obtained for a version of the library must be consistent with the full freedom of use specified in this license.

Most GNU software, including some libraries, is covered by the ordinary GNU General Public License. This license, the GNU Lesser General Public License, applies to certain designated libraries, and is quite different from the ordinary General Public License. We use this license for certain libraries in order to permit linking those libraries into non-free programs.

When a program is linked with a library, whether statically or using a shared library, the combination of the two is legally speaking a combined work, a derivative of the original library. The ordinary General Public License therefore permits such linking only if the entire combination fits its criteria of freedom. The Lesser General Public License permits more lax criteria for linking other code with the library.

We call this license the "Lesser" General Public License because it does Less to protect the user's freedom than the ordinary General Public License. It also provides other free software developers Less of an advantage over competing non-free programs. These disadvantages are the reason we use the ordinary General Public License for many libraries. However, the Lesser license provides advantages in certain special circumstances.

For example, on rare occasions, there may be a special need to encourage the widest possible use of a certain library, so that it becomes a de-facto standard. To achieve this, non-free programs must be allowed to use the library. A more frequent case is that a free library does the same job as widely used non-free libraries. In this case, there is little to gain by limiting the free library to free software only, so we use the Lesser General Public License.

In other cases, permission to use a particular library in non-free programs enables a greater number of people to use a large body of free software. For example, permission to use the GNU C Library in non-free programs enables many more people to use the whole GNU operating system, as well as its variant, the GNU/Linux operating system.

Although the Lesser General Public License is Less protective of the users' freedom, it does ensure that the user of a program that is linked with the Library has the freedom and the wherewithal to run that program using a modified version of the Library.

The precise terms and conditions for copying, distribution and modification follow. Pay close attention to the difference between a "work based on the library" and a "work that uses the library". The former contains code derived from the library, whereas the latter must be combined with the library in order to run.

GNU LESSER GENERAL PUBLIC LICENSE
TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

0. This License Agreement applies to any software library or other program which contains a notice placed by the copyright holder or other authorized party saying it may be distributed under the terms of this Lesser General Public License (also called "this License"). Each licensee is addressed as "you".

A "library" means a collection of software functions and/or data prepared so as to be conveniently linked with application programs (which use some of those functions and data) to form executables.

The "Library", below, refers to any such software library or work which has been distributed under these terms. A "work based on the Library" means either the Library or any derivative work under copyright law: that is to say, a work containing the Library or a portion of it, either verbatim or with modifications and/or translated straightforwardly into another language. (Hereinafter, translation is included without limitation in the term "modification".)

"Source code" for a work means the preferred form of the work for making modifications to it. For a library, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the library.

Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running a program using the Library is not restricted, and output from such a program is covered only if its contents constitute a work based on the Library (independent of the use of the Library in a tool for writing it). Whether that is true depends on what the Library does and what the program that uses the Library does.

1. You may copy and distribute verbatim copies of the Library's complete source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and distribute a copy of this License along with the Library.

You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Library or any portion of it, thus forming a work based on the Library, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

a) The modified work must itself be a software library.

b) You must cause the files modified to carry prominent notices stating that you changed the files and the date of any change.

c) You must cause the whole of the work to be licensed at no charge to all third parties under the terms of this License.

d) If a facility in the modified Library refers to a function or a table of data to be supplied by an application program that uses the facility, other than as an argument passed when the facility is invoked, then you must make a good faith effort to ensure that, in the event an application does not supply such function or table, the facility still operates, and performs whatever part of its purpose remains meaningful.

(For example, a function in a library to compute square roots has a purpose that is entirely well-defined independent of the application. Therefore, Subsection 2d requires that any application-supplied function or table used by this function must be optional: if the application does not supply it, the square root function must still compute square roots.)

These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Library, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Library, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Library.

In addition, mere aggregation of another work not based on the Library with the Library (or with a work based on the Library) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

3. You may opt to apply the terms of the ordinary GNU General Public License instead of this License to a given copy of the Library. To do this, you must alter all the notices that refer to this License, so that they refer to the ordinary GNU General Public License, version 2, instead of to this License. (If a newer version than version 2 of the ordinary GNU General Public License has appeared, then you can specify that version instead if you wish.) Do not make any other change in these notices.

Once this change is made in a given copy, it is irreversible for that copy, so the ordinary GNU General Public License applies to all subsequent copies and derivative works made from that copy.

This option is useful when you wish to copy part of the code of the Library into a program that is not a library.

4. You may copy and distribute the Library (or a portion or derivative of it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange.

If distribution of object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place satisfies the requirement to distribute the source code, even though third parties are not compelled to copy the source along with the object code.

5. A program that contains no derivative of any portion of the Library, but is designed to work with the Library by being compiled or linked with it, is called a "work that uses the Library". Such a work, in isolation, is not a derivative work of the Library, and therefore falls outside the scope of this License.

However, linking a "work that uses the Library" with the Library creates an executable that is a derivative of the Library (because it contains portions of the Library), rather than a "work that uses the library". The executable is therefore covered by this License. Section 6 states terms for distribution of such executables.

When a "work that uses the Library" uses material from a header file that is part of the Library, the object code for the work may be a derivative work of the Library even though the source code is not. Whether this is true is especially significant if the work can be linked without the Library, or if the work is itself a library. The threshold for this to be true is not precisely defined by law.

If such an object file uses only numerical parameters, data structure layouts and accessors, and small macros and small inline functions (ten lines or less in length), then the use of the object file is unrestricted, regardless of whether it is legally a derivative work. (Executables containing this object code plus portions of the Library will still fall under Section 6.)

Otherwise, if the work is a derivative of the Library, you may distribute the object code for the work under the terms of Section 6. Any executables containing that work also fall under Section 6, whether or not they are linked directly with the Library itself.

6. As an exception to the Sections above, you may also combine or link a "work that uses the Library" with the Library to produce a work containing portions of the Library, and distribute that work under terms of your choice, provided that the terms permit modification of the work for the customer's own use and reverse engineering for debugging such modifications.

You must give prominent notice with each copy of the work that the Library is used in it and that the Library and its use are covered by this License. You must supply a copy of this License. If the work during execution displays copyright notices, you must include the copyright notice for the Library among them, as well as a reference directing the user to the copy of this License. Also, you must do one of these things:

a) Accompany the work with the complete corresponding machine-readable source code for the Library including whatever changes were used in the work (which must be distributed under Sections 1 and 2 above); and, if the work is an executable linked with the Library, with the complete machine-readable "work that uses the Library", as object code and/or source code, so that the user can modify the Library and then relink to produce a modified executable containing the modified Library. (It is understood that the user who changes the contents of definitions files in the Library will not necessarily be able to recompile the application to use the modified definitions.)

b) Use a suitable shared library mechanism for linking with the Library. A suitable mechanism is one that (1) uses at run time a copy of the library already present on the user's computer system, rather than copying library functions into the executable, and (2) will operate properly with a modified version of the library, if the user installs one, as long as the modified version is interface-compatible with the version that the work was made with.

c) Accompany the work with a written offer, valid for at least three years, to give the same user the materials specified in Subsection 6a, above, for a charge no more than the cost of performing this distribution.

d) If distribution of the work is made by offering access to copy from a designated place, offer equivalent access to copy the above specified materials from the same place.

e) Verify that the user has already received a copy of these materials or that you have already sent this user a copy.

For an executable, the required form of the "work that uses the Library" must include any data and utility programs needed for reproducing the executable from it. However, as a special exception, the materials to be distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

It may happen that this requirement contradicts the license restrictions of other proprietary libraries that do not normally accompany the operating system. Such a contradiction means you cannot use both them and the Library together in an executable that you distribute.

7. You may place library facilities that are a work based on the Library side-by-side in a single library together with other library facilities not covered by this License, and distribute such a combined library, provided that the separate distribution of the work based on the Library and of the other library facilities is otherwise permitted, and provided that you do these two things:

a) Accompany the combined library with a copy of the same work based on the Library, uncombined with any other library facilities. This must be distributed under the terms of the Sections above.

b) Give prominent notice with the combined library of the fact that part of it is a work based on the Library, and explaining where to find the accompanying uncombined form of the same work.

8. You may not copy, modify, sublicense, link with, or distribute the Library except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense, link with, or distribute the Library is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

9. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Library or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Library (or any work based on the Library), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Library or works based on it.

10. Each time you redistribute the Library (or any work based on the Library), the recipient automatically receives a license from the original licensor to copy, distribute, link with or modify the Library subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties with this License.

11. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Library at all. For example, if a patent license would not permit royalty-free redistribution of the Library by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Library.

If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply, and the section as a whole is intended to apply in other circumstances.

It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

12. If the distribution and/or use of the Library is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Library under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

13. The Free Software Foundation may publish revised and/or new versions of the Lesser General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Library specifies a version number of this License which

applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Library does not specify a license version number, you may choose any version ever published by the Free Software Foundation.

14. If you wish to incorporate parts of the Library into other free programs whose distribution conditions are incompatible with these, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

NO WARRANTY

15. BECAUSE THE LIBRARY IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE LIBRARY, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE LIBRARY "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE LIBRARY IS WITH YOU. SHOULD THE LIBRARY PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

16. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE LIBRARY AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE LIBRARY (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE LIBRARY TO OPERATE WITH ANY OTHER SOFTWARE), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

END OF TERMS AND CONDITIONS

References

[1] D. Murray, et al., *LUCID User's Guide.*
available at http://nucleus.usask.ca under "Reports".

[2] ROOT Collaboration, *The ROOT System Home Page.* http://root.cern.ch/.