University of Saskatchewan
Subatomic Physics Internal Report

SPIR 147.5

The G4Lucid Software Package

Version 1.5

Ward Andrew Wurtz and Rob Pywell

June 2013, Last Update: July 20, 2017

Abstract

The report describes the G4Lucid package. G4Lucid allows a Geant4 simulation to write data in the same format as written by the Lucid data acquisition system. This allows simulation data to be analyzed with the same software as real data.

Contents

1 Introduction

GEANT4 [G4] is a simulation toolkit for tracking particles through matter. Lucid [SPIR-100] is a data acquisition and analysis system for use in nuclear experiments. In an experiment, the University of Saskatchewan experimental subatomic physics group uses the Lucid data acquisition system to accumulate data. Lucid is also used to perform histogramming and other data analysis during online data taking and offline. As well, software exists to analyze Lucid data files using the Root [Root] data analysis framework (see RLucid [SPIR-151]). Therefore, it is desirable to have data generated by a GEANT4 simulation be written in the same format that is written by the data acquisition system. Then the *same* software can be used to analyze both real and simulated data. This is the purpose of the G4Lucid package.

The G4Lucid package is an extension to GEANT4 to make interfacing with Lucid easier. GEANT4 is built in the object-oriented style of programming while Lucid is written sequentially. The G4Lucid package is built using object-oriented methods and it takes care of the memory allocation and structure building that Lucid requires. To the GEANT4 user, the G4Lucid package appears as a collection of objects. To the Lucid user, the data output from the G4Lucid package appears as an ordinary data file.

The main body of this document consists of two parts. Section 3 discusses how to use the G4Lucid package. Section 4 explains how the G4Lucid package was programmed and how it can be modified. Additionally, there are three appendices. Appendix A list the functions and classes that are available to the user through the G4Lucid package. Appendix B describes the Lesser GNU Public License, which governs redistribution of the G4Lucid package.

2 Version 1.5

This version of the G4Lucid package does *not* require that the Lucid data acquisition or analysis system be installed on the computer on which you wish to use G4Lucid. The needed files from Lucid are included in the package.

As well, the G4Lucid code, and the included Lucid code, have been modified so that it will produce correct Lucid data files on computers with either 32-bit or 64-bit architecture.

Version 1.5 has been modified so that it compiles (almost) cleanly on the most recent version of Geant (tested on Geant4-10-3.1) and with recent gcc versions (tested using gcc 4.8.5). There are still a couple of warning messages when compiling the lucid library. These are related to code supporting tape drives. (I hope to eventually purge Lucid of all references to tape drives!)

Other functionality is the same as in previous versions.

3 User's Manual

This section assumes the reader is familiar with using GEANT4 and object-oriented programming. It also assumes a basic knowledge of GXLucid and writing Looker code. Understanding the Lucid Reader and Writer is not important for using the G4Lucid package.

The G4Lucid package is simply a collection of classes. Section 3.1 describes how to compile and install these classes. Section 3.2 discusses the basic hierarchy of the classes in the G4Lucid package. Sections 3.3 and 3.4 construct a simple example program using the G4Lucid package. These techniques

are expanded upon in section 3.5 which describes how to go from the simple example to using the G4Lucid package in your GEANT4 application.

3.1 Installing the G4Lucid Package

The G4Lucid package requires that GEANT4 is already installed on the computer. The G4Lucid package is designed to run on a Linux system and has not been tested on any other configuration. This release of the G4Lucid package has been tested with Geant4.10.00.p03 running on a CentOS 6 32-bit Linux operating system with gcc 4.4.7, and with Geant4.10.3.p01 running on a CentOS 7 64-bit Linux operating system with gcc 4.8.5. This version of G4Lucid does not require that Lucid be installed the computer. The needed files from lucid are included in the package.

This guide assume you will have installed Geant4 on a Linux operating system using the CMake build system. It assumes you have CMake version 2.8 or higher. More recent versions of Geant4 require CMake 3 but that is not needed for compiling G4Lucid.

Get the package `G4Lucid-1.5.tgz` from `http://nucleus.usask.ca/` under the "Software" tab. Unpack the tar file into a suitable directory in your geant4 tree.

e.g. suppose you have installed geant4 in a directory structure

```
${HOME}/geant4
        /geant4-10.0.00.p03
        /geant4-install
        /geant4-build
```

then

```
cd ${HOME}/geant4
tar  zxvf G4Lucid-1.5.tgz
```

This will create the directory `G4Lucid-1.5`. Change to the created directory

```
cd ${HOME}/geant4/G4Lucid-1.5
```

which contains several files and sub-directories. The file `LICENSE` describes the license details for the G4Lucid package's code. It has been released under the Lesser GNU Public License (LGPL) which allows redistribution and modification of the code. The license text is included in Appendix B. The directories `src` and `include` contain the source code and header files respectively. They conform to the GEANT4 standard for source and header directories. The directory `latex` contains the LaTeXsource for this manual. This manual can be compiled by executing `make` three times in the `latex` directory. The sub-directory `lucidlib` contains the files needed from Lucid. The directory also includes a `CMakeLists.txt` file to simplify the installation using `cmake`.

3.1.1 Compiling the Lucid library

The first step is to compile the needed Lucid library `liblucid.a`.
Change to the directory `lucidlib/src`.

```
cd ${HOME}/geant4/G4Lucid-1.5/lucidlib/src
```

The compilation of the lucid library does not use `cmake` but uses a regular makefile. In that directory execute the following commands:

```
make clean
make depend
make
make install
```

During compilation there will be some warning messages. For now, these may safely be ignored. (Later versions will hopefully clean up the lucid code to comply with modern gcc standards.) The last step copies `liblucid.a` to `${HOME}/geant4/G4Lucid-1.5/lucidlib`.

The `G4Lucid` package may now be compiled.

3.1.2 Creating the Makefile using CMake

The package is designed to be compiled using CMake. Change to the `G4Lucid-1.5` directory.

```
cd ${HOME}/geant4/G4Lucid-1.5
```

To avoid grief, it is advisable on a new installation to first ensure that there are no CMake files left behind from a previous installation. Remove all CMake files except `CMakeLists.txt`. i.e.

```
rm CMakeCache.txt
rm cmake_install.cmake
rm -r CMakeFiles
```

The `CMakeLists.txt` needs to be modified to find your installation of Geant4. Edit the `CMakeLists.txt` file with a text editor and locate the following lines.

```
# ========================================================
# You will need to modify the following lines as needed
# This is where the geant4 files are
set(GEANT4_LOCAL /home/rob/geant4)
# The Geant4 installation
#set(Geant4_DIR ${GEANT4_LOCAL}/geant4-install/lib64/Geant4-10.3.1 )
set(Geant4_DIR ${GEANT4_LOCAL}/geant4-install/lib/Geant4-10.0.3 )
# ========================================================
```

Change the lines that set the variables `GEANT4_LOCAL` and `Geant4_DIR` so that `Geant4_DIR` points to the directory in your particular GEANT4 installation directory which contains the `Geant4Config.cmake` file.

Then to run CMake simply type

```
cmake CMakeLists.txt
```

or simply

```
cmake .
```

This should create the file `Makefile`.

3.1.3 Creating the Makefile using the CMake GUI

Alternatively, to use the CMake GUI, start the GUI and

> Browse Source and set it to `${HOME}/geant4/G4Lucid-1.5`.
> Browse Build and set it to `${HOME}/geant4/G4Lucid-1.5`.
> Click "Configure".
> Choose the default `Unix Makefiles` and `Use Native Compilers` in the pop-up and click OK.

If it complains that it cannot find the package configuration file for Geant4. Set `Geant4_DIR` by:

> Clicking on `Geant4_DIR-NOTFOUND` then clicking on the `[...]` button.
> Choose the geant4 directory containing the `Geant4Config.cmake` file.
> e.g. `${HOME}/geant4/geant4-install/lib/Geant4-10.0.0.3`.
> or, on a 64-bit system `${HOME}/geant4/geant4-install/lib64/Geant4-10.0.0.3`.
> Click `Open`.

Click "Configure" again. It should simply report "Configuring done". (You do not need to change `CMAKE_BUILD_TYPE` or `CMAKE_INSTALL_PREFIX`. The G4Lucid Library can stay in this directory.) You will note that the variable `Lucid_lib` will appear (often highlighted in red). It should automatically be set to `${HOME}/geant4/G4Lucid-1.5/lucidlib/liblucid.a` which is the location of the lucid library. If it is not set you may need to set it manually. (If you click "Configure" again the red highlighting should go away.)

Click "Generate". It should simply report "Generating done". A `Makefile` will have been created in the `G4Lucid-1.5` directory.

3.1.4 Building the G4Lucid Library

After running cmake, and while still in the `G4Lucid-1.5` directory type `make`. This will create the library `libG4Lucid.so` with the following printout.

```
Scanning dependencies of target G4Lucid
[  7%] Building CXX object CMakeFiles/G4Lucid.dir/src/G4Lucid.cc.o
[ 14%] Building CXX object CMakeFiles/G4Lucid.dir/src/G4LucidADC.cc.o
[ 21%] Building CXX object CMakeFiles/G4Lucid.dir/src/G4LucidV792.cc.o
[ 28%] Building CXX object CMakeFiles/G4Lucid.dir/src/G4LucidDouble.cc.o
[ 35%] Building CXX object CMakeFiles/G4Lucid.dir/src/G4VLucidVariable.cc.o
[ 42%] Building CXX object CMakeFiles/G4Lucid.dir/src/G4LucidEvent.cc.o
[ 50%] Building CXX object CMakeFiles/G4Lucid.dir/src/G4LucidFloat.cc.o
[ 57%] Building CXX object CMakeFiles/G4Lucid.dir/src/G4LucidLong.cc.o
[ 64%] Building CXX object CMakeFiles/G4Lucid.dir/src/G4LucidMessenger.cc.o
[ 71%] Building CXX object CMakeFiles/G4Lucid.dir/src/G4LucidV775.cc.o
[ 78%] Building CXX object CMakeFiles/G4Lucid.dir/src/G4LucidCompLong.cc.o
[ 85%] Building CXX object CMakeFiles/G4Lucid.dir/src/G4LucidShort.cc.o
[ 92%] Building CXX object CMakeFiles/G4Lucid.dir/src/G4LucidChar.cc.o
[100%] Building CXX object CMakeFiles/G4Lucid.dir/src/G4LucidTDC.cc.o
```

```
Linking CXX shared library libG4Lucid.so
[100%] Built target G4Lucid
```

If no errors occurred during compilation, the G4Lucid package should now be installed. (No make install is needed, the G4Lucid library can stay where it is.) We can now learn the structure of the G4Lucid package and use it in an example.

3.2 Understanding the Classes

The first step in using the G4Lucid package is to understand its data structure hierarchy. Figure 1 shows how a user can visualize the data structures that make up the G4Lucid package.

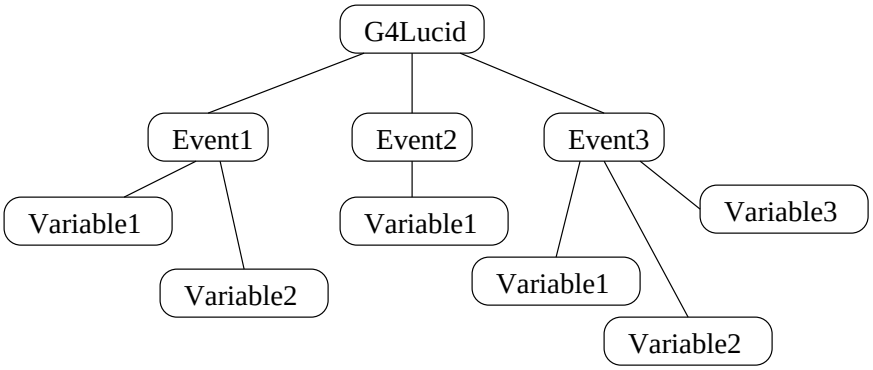

Figure 1: Map of the GEANT4 data-structures.

At the top of the hierarchy, there is a single, static instance of `G4Lucid`. In any GEANT4 program, there cannot be more than one instance of `G4Lucid`, and this is rigidly enforced. The `G4Lucid` object takes care of global things such as the experiment name, description and run number. Next, there is a class called `G4LucidEvent`. Each Lucid event needs a separate instance of `G4LucidEvent`. In figure 1 there are three Lucid events. These events are associated with the `G4Lucid` object using the function `AddLucidEvent(G4LucidEvent *)`. Inside a Lucid event there are variables. Variables can be floats, short integers or any other variable defined in the Lucid data stream. Variables are instances of a class, such as `G4LucidShort` or `G4LucidFloat`. These variables can be added to an event using the functions `AddLucidVariable(G4LucidShort *)` or `AddLucidVariable(G4LucidFloat *)`.

Variables can be set using the `Set()` function. When a variable is set, its value is stored in memory. The value is saved to the disk when a Lucid event is written. Writing a Lucid event with `WriteEvent()` will write all variables associated in that event to a file.

With an understanding of the data structure hierarchy in the G4Lucid package, we can now begin using the classes. This is the next step in our goal of constructing a working example.

3.3 Using the Classes

Using the G4Lucid package is straight forward for anyone familiar with C++ and object oriented programming. One simply has to get a pointer to the `G4Lucid` object, set some strings, create some events and associate variables with the events. Then one can begin a run, set the variables and write them to the Lucid data file.

Before we can begin using the G4Lucid package, we need to include the necessary header files. This can be done with the line

```
#include "G4LucidInclude.hh"
```

which will include the header files for `G4Lucid`, `G4LucidEvent` and all of the Lucid variables. You may also need to tell your compiler where to find these header files. See the makefile in section 3.4 for more information on passing information to the compiler.

The first step in using the G4Lucid package within a GEANT4 application is to get a pointer to the `G4Lucid` object.

```
G4Lucid * lucidPtr = G4Lucid::GetLucidPointer();
```

Since this is the first call to `GetLucidPointer()` the singleton `G4Lucid` object will be created automatically by the function `GetLucidPointer()`. All subsequent calls to `GetLucidPointer()` simply return a pointer to the instance that has already been created. In this way, we are assured that only one instance of `G4Lucid` exists at any given time.

Next, we should set the experiment name and description and change the filename prefix.

```
lucidPtr -> SetExpName("My Really Neat Experiment");
lucidPtr -> SetExpDescription("GEANT 4 Simulation");
lucidPtr -> SetFilenamePrefix("lucid");
```

We are now ready to create and add Lucid events.

```
G4LucidEvent * eventHit = new G4LucidEvent("hit");
lucidPtr -> AddLucidEvent(eventHit);
```

The above command creates a new `G4LucidEvent` called "hit" and adds it to `lucidPtr`. One does not need to worry about deleting `eventHit` later since deleting `lucidPtr` will recursively delete all events and variables.

We can now add variables to `eventHit`. Let us add a `float`[1] to keep track of the energy deposited and a `short` to represent an ADC.

```
G4LucidFloat * hitEng = new G4LucidFloat("energy");
G4LucidShort * hitADC = new G4LucidShort("adc");
```

Now that we have defined two variables, we must assign them to an event.

```
eventHit -> AddLucidVariable( hitEng );
eventHit -> AddLucidVariable( hitADC );
```

Notice that the function `AddLucidVariable()` is overloaded so it can handle both data types.

The setup step is now complete. To use the G4Lucid package to collect data, a run must be started.

```
lucidPtr -> BeginRun();
```

Variables may now be given data.

[1] We must use `float` as `double` does not work properly. See section 4.7 for more details.

```
hitEng -> Set( 3.14*MeV );
hitADC -> Set( 137 );
```

The data in the G4Lucid package's variables can now be written to the Lucid file. This is done by writing an event.

```
eventHit -> WriteEventClear();
```

All variables in `eventHit` have been written to the Lucid file and cleared. Data has been written so the run can be stopped.

```
lucidPtr -> EndRun();
```

Lucid has stopped accepting new data. One can now begin another run or can end the program. To free the memory taken by the G4Lucid package, simply delete `lucidPtr`. This will recursively delete all the events and variables defined above.

```
delete lucidPtr;
```

The data written during this run should be output to the file `lucid0001`. It can be viewed with `lucidview` and all the Lucid tools. (Note that a version of `luciview`, that does not need `Lucid` to be installed, and will work on both 32-bit and 64-bit systems, is included in the `RLucid` package.)

3.4 A Simple Example

While the previous section presents the basic techniques for building a program that uses the G4Lucid package, an example is always helpful. This section provides a complete working example that can be found in `examples/simple`. It contains a CMake file, `CMakeList.txt` and a program, `exampleL01.cc`.

The following code can be found in `exampleL01.cc` and demonstrates the usage of the G4Lucid package:

```
/*************************************************************
 * exampleL01.cc
 *
 * First Started 31 May 2005 by Ward Wurtz
 * Last Modified October 2016 by Rob Pywell
 *
 * This is a very simple example to demonstrate how to use the
 * basic functionality of the G4Lucid extension in your
 * GEANT4 simulation.
 *
 *************************************************************/

//GEANT4 generic includes
#include "globals.hh"

//G4Lucid includes
```

```
#include "G4LucidInclude.hh"

//The only method
int main(void)
{
    //Get a pointer to the G4Lucid instance
    G4Lucid * lucidPtr = G4Lucid::GetLucidPointer();

    //Set experiment name and description
    lucidPtr->SetExpName("My Really Neat Experiment");
    lucidPtr->SetExpDescription("GEANT 4 Simulation");
    lucidPtr->SetFilenamePrefix("lucid");

    //Create an event and register it with G4Lucid
    G4LucidEvent * eventHit = new G4LucidEvent("hit");
    lucidPtr->AddLucidEvent(eventHit);

    //Create variables and add them to the event
    G4LucidFloat * hitEng = new G4LucidFloat("energy");
    G4LucidShort * hitADC = new G4LucidShort("adc");
    G4LucidLong * hitLong = new G4LucidLong("long",3);
    eventHit -> AddLucidVariable( hitEng );
    eventHit -> AddLucidVariable( hitADC );
    eventHit -> AddLucidVariable( hitLong );

    //Try another event with compressed data
    G4LucidEvent * eventComp = new G4LucidEvent("comp");
    lucidPtr->AddLucidEvent(eventComp);
    G4LucidV792 * vmeadc = new G4LucidV792("vmeadc",0.002, 100);
    eventComp->AddLucidVariable(vmeadc);
    vmeadc->SetZeroSuppression(true);

    //Begin the run
    lucidPtr -> BeginRun();

    //Set the variables
    hitEng -> Set( 3.14*MeV );
    hitADC -> Set( 137 );
    for(G4int i = 0; i < 3; i++)
        {
        hitLong->Set(i*1000,i);
        }
    //Write the event and clear its variables
    eventHit->WriteEventClear();
```

```
    //Another event
    hitEng -> Set( 2*3.14*MeV );
    hitADC -> Set( 2*137 );
    for(G4int i = 0; i < 3; i++)
        {
        hitLong->Set(2*i*1000,i);
        }
    eventHit->WriteEventClear();

    // add data to the vmeadc
    for(int i = 0; i < 32; i++)
        vmeadc->SetQDC(0., i);
    vmeadc->SetQDC(1*MeV, 4);
    vmeadc->SetQDC(2*MeV, 6);
    vmeadc->SetQDC(2.5*MeV, 9);

    eventComp->WriteEventClear();

    //End the run
    lucidPtr->EndRun();

    //Free all memory allocated to G4Lucid
    delete lucidPtr;

    return 0;
}

//End of file exampleL01.cc
```

This example may be compiled using CMake in the same way as for other Geant4 applications. The following `CMakeLists.txt` file is used to compile the program:

```
cmake_minimum_required(VERSION 2.6 FATAL_ERROR)
project(exampleL01)

# ========================================================
# You will need to modify the following lines as needed
# This is where the geant4 files are
set(GEANT4_LOCAL /home/rob/geant4)
# G4Lucid location
set(G4LUCID_DIR ${PROJECT_SOURCE_DIR}/../../)
# This is where liblucid.a is located
set(LUCIDLIB_DIR ${G4LUCID_DIR}/lucidlib )
# The Geant4 installation
```

```
set(Geant4_DIR ${GEANT4_LOCAL}/geant4-install/lib/Geant4-10.0.3 )
# ========================================================

IF(NOT CMAKE_BUILD_TYPE)
  SET(CMAKE_BUILD_TYPE Release CACHE STRING
      "Choose the type of build, options are: None Debug Release
RelWithDebInfo MinSizeRel."
      FORCE)
ENDIF(NOT CMAKE_BUILD_TYPE)

# Find Geant4 package, activating all available UI and Vis drivers by default
# You can set WITH_GEANT4_UIVIS to OFF via the command line or ccmake/cmake-gui
# to build a batch mode only executable

option(WITH_GEANT4_UIVIS "Build example with Geant4 UI and Vis drivers" ON)
if(WITH_GEANT4_UIVIS)
  find_package(Geant4 REQUIRED ui_all vis_all)
else()
  find_package(Geant4 REQUIRED)
endif()

# Setup Geant4 include directories and compile definitions
include(${Geant4_USE_FILE})

include_directories("${G4LUCID_DIR}/include")

# get the extra libraries needed
find_library(G4Lucid_lib G4Lucid ${G4LUCID_DIR} )
find_library(Lucid_lib lucid ${LUCIDLIB_DIR})

# Add the executable, and link it to the Geant4 libraries
add_executable(exampleL01 exampleL01.cc  )
target_link_libraries(exampleL01 ${Geant4_LIBRARIES} ${G4Lucid_lib} ${Lucid_lib} )
```

You will need to modify the lines near the beginning of the `CMakeLists.txt` file as appropriate so that set the variables `GEANT4_LOCAL` and `Geant4_DIR` so that `Geant4_DIR` points to the directory in your GEANT4 installation directory that contains the `Geant4Config.cmake` file. The other variables should not need to be changed.

Once again, to avoid grief, it is advisable on a new installation to first ensure that there are no CMake files left behind from a previous installation. Remove all CMake files except `CMakeLists.txt`. i.e.

```
rm CMakeCache.txt
rm cmake_install.cmake
rm -r CMakeFiles
```

After running `cmake` in the usual way, by using the CMake GUI or the command line, the program can be compiled by running `make` and run using the command `exampleL01`. It will create a Lucid data file named `lucid0001`. We can now use the command

```
lucidview lucid0001
```

to view the Lucid file. The `lucidview` output is presented below.

```
Oct 25 15:20:57 2016         1  New session by rob.
        Experiment 'My Really Neat Experiment'
        Build name 'GEANT 4 Simulation'
Oct 25 15:20:57 2016         1  Format 1.25 1.25.
Oct 25 15:20:57 2016         1  Event 1 (hit) symbol table.
            integer*4 long[3]
               real*4 energy
            integer*2 adc
Oct 25 15:20:57 2016         2  Event 2 (comp) symbol table.
            integer*4 vmeadc[34] (compressed, channel offset:0 length:0, value offset:0 l
          character*2 <<pad>>
Oct 25 15:20:57 2016         1  Begin run 1 by *OFFLINE*.
Oct 25 15:20:57 2016         1  84 data bytes.  Run 1.
        Event 1 (hit)
                long:          0       1000       2000
                energy:           3.14
                adc:   137
        Event 1 (hit)
                long:          0       2000       4000
                energy:           6.28
                adc:   274
        Event 2 (comp)
                vmeadc:  33554432     262743
                           394315     591173
                          67108864
                <<pad>>: [2]
Oct 25 15:20:57 2016         1  End run 1 by *OFFLINE*.
                2 event 1 (hit).
                1 event 2 (comp).
```

Note that a version of `luciview`, that does not need `Lucid` to be installed, and will work on both 32-bit and 64-bit systems, is included in the `RLucid` package. The above output is included in the file `lucidview-lucid0001-output.txt` that is included in the `simple` directory.

We have now successfully written and run a program relying on the G4Lucid package. The next section contains hints for taking our usage further. It describes ways to easily incorporate the G4Lucid package into a full GEANT4 program.

3.5 Advanced Usage

The previous sections discuss the basics required to use the G4Lucid package in an application. This section discusses techniques that will make it easier to write and use GEANT4 applications that rely on the G4Lucid package. Such techniques include where objects are created and used, the different types of variables and using the commands added to the GEANT4 user interface.

This manual does not list all the functions contained in the classes that comprise the G4Lucid package. For more information, it is informative to view the `.hh` files in the `include` directory. They are well commented and describe many useful functions.

3.5.1 Advanced Usage: Application Organization

This section describes a method for organizing the creation and use of the G4Lucid package's different classes in an actual GEANT4 program. Since a GEANT4 program can be large, it is nice to have a systematic method for defining all events and variables.

It is useful to set the experiment name and description and define all events in the `main()` method. A shortcut command to creating an event is demonstrated below:

```
lucidPtr->AddLucidEvent(new G4LucidEvent("hit"));
```

From anywhere in the program, a pointer to the event can be recovered with the following code:

```
G4Lucid * lucidPtr = G4Lucid::GetLucidPointer();
hitEvent = lucidPtr -> GetLucidEvent("hit");
if( hitEvent == 0 ){
     G4cerr << "FATAL ERROR: hitEvent was given the null pointer." << G4endl;
     exit(1); }
```

An event may be used in two different files. The above technique ensures that the event is defined before it is used. It also makes the symbol table easier to analyze since the Lucid symbol table is built in the same order as events are added to `G4Lucid`.

Likewise, variables can be obtained from an event in a similar fashion.

```
G4LucidShort * myShort = hitEvent -> GetLucidShort("myShortName");
G4LucidLong  * myLong  = hitEvent -> GetLucidLong ("myLongName" );
G4LucidFloat * myFloat = hitEvent -> GetLucidFloat("myFloatName");
```

Each variable type has a corresponding function to get it from the event.

With proper organization, and using the above functions, one can make their simulation easier to write, understand and maintain. Since any class and function can get a pointer to the instance of the `G4Lucid` object, the events and variables can be easily accessed from any piece of code.

3.5.2 Advanced Usage: The G4Lucid Package's Variables

Variables are the objects that contain data. If they are associated with an event, they are written to a Lucid file when the event is written. This section describes the variables and some techniques to use them better.

All variables are children of the parent class `G4VLucidVariable`. This class takes care of functions common to all variable types. Such functions include storing the name and number of the variable and defining pure virtual functions such as `Clear()`. Variables that are derived from `G4VLucidVariable` are listed below:

- `G4LucidChar`
- `G4LucidShort`
- `G4LucidLong`
- `G4LucidFloat`
- `G4LucidCompLong`

The class `G4LucidCompLong` stores long integers in a Lucid event's compressed memory space.

The class `G4LucidShort` is often used to store data from an analogue-to-digital converter (ADC) or a time-to-digital converter (TDC). Rather than having to implement these common conversion manually, special classes were built:

- `G4LucidADC`
- `G4LucidTDC`

These classes inherit all the functionality of `G4LucidShort` and add specific routines to make `double`-to-`short` conversions.

Using the extensions is very easy. The `G4LucidADC` class uses the method `SetADC()` to take a `double` and convert it to a `short` according to user-set options. Likewise `G4LucidTDC` can take a start and stop time and convert them into a `short`. For more details on the methods available to `G4LucidADC` and `G4LucidTDC` please see Appendix A.

Also, the class `G4LucidCompLong` is meant to be used by a derived class that implements its own compression algorithm. It is used by the following classes:

- `G4LucidV775`
- `G4LucidV792`

These classes represent the CAEN V775 TDC and V792 ADC, which are VME modules that use zero suppression. The classes provide the ADC and TDC data in the format expected from the CAEN modules.

Any variable can be added to more than one event. This is useful for two events that read out similar, but not the same, data.

Lucid is able to handle arrays of variables. Therefore, an index can be added to the variables of the G4Lucid package. For instance, if we have three detectors we might want to keep track of the energy deposited in each detector. To create an array of variables we must tell the constructor how many elements will be in the array.

```
G4LucidFloat * hitEng = new G4LucidFloat( "energy" , 3 );
```

When we add data to the variable we must supply an index.

```
hitEng -> Set( 3.14*MeV , 0);
hitEng -> Set( 6.28*MeV , 1);
hitEng -> Set( 9.42*MeV , 2);
```

When the event that contains `hitEng` is written, the array is written just as a single value was written before.

3.5.3 Advanced Usage: Clearing Variables

There are several different ways of clearing the G4Lucid package's variables. The first and most simple way is to call the variable's `Clear()` function directly. Alternatively, the `ConditionalClear()` function can be used to clear the variable. The difference between `Clear()` and `ConditionalClear()` is that `Clear()` will always clear the variable, but `ConditionalClear()` will only clear the variable if conditional clearing is enabled. Conditional clearing is enabled by default, but its status can be changed by using the functions `EnableConditionalClear()` and `DiableConditionalClear()`.

By calling the `Clear()` function for a `G4LucidEvent`, one can clear the entire event. By clearing an entire event, each variable will have its `ConditionalClear()` function called. Also, the function `WriteEventClear()` will clear the event after a write has occurred. Notice that the above functions conditionally clear the event's variables, which means that they will not clear any variable which has conditional clear disabled. It is possible to override the conditional clear and force all variables in the event to be cleared. This is done using the function `ForceClear()`.

The difference between `Clear()` and `ForceClear()` is useful for variables that should be written at each event but only reset at the end of a run. Take, for example, a `G4LucidEvent` with two variables: a counter (a `G4LucidLong`) and an ADC. The ADC should be written and cleared at the end of each event and this can be accomplished by using the function `WriteEventClear()` of the `G4LucidEvent`. However, if we do not want to reset the counter for each event, we can call `DisableConditionalClear()` which is a member function of `G4LucidLong`. At the end of the run, we can call `ForceClear()` of the `G4LucidEvent` to clear both the ADC and the counter.

It is also possible to clear all events using the `Clear()` and `ForceClear()` member functions of the `G4Lucid` class. These functions operate in the same manner as those in `G4LucidEvent`.

3.5.4 Advanced Usage: Command-Line Interface

GEANT4 provides many tools for a user to use in an application. One of the most useful is the command-line user interface. The G4Lucid package allows a user to control many options from the command line. The directory `/lucid/` in the root directory of the GEANT4 command line contains all of the commands. The contents of the directory are listed below:

```
Command directory path : /lucid/


Guidance :
Lucid Data Acquisition System
```

```
 Sub-directories :
   /lucid/event/   Lucid Event Settings
 Commands :
   print * Print all Lucid events and variables.
   clear * Clear all Lucid events and variables.
   name * Set the experiment name.
   description * Set the experiment description.
   filename * Set the filename prefix.
   about * Print information about G4Lucid.
```

The function of the commands should be clear from their description. For more information on a command type `help command_name`. The directory `/lucid/event` contains commands for enabling and disabling the writing of individual events. The directory's contents are listed below:

```
Command directory path : /lucid/event/


Guidance :
Lucid Event Settings

 Sub-directories :
 Commands :
   list * List all Lucid events.
   enable * Enable writing for a Lucid event.
   disable * Disable writing for a Lucid event.
```

These commands allow a user to deactivate uninteresting events in order to reduce the size of outputted Lucid files.

3.6 Conclusion

The G4Lucid package has been installed and is ready for use. With the example of section 3.4 and the information of section 3.5, you should have all the tools required to write GEANT4 packages that take advantage of G4Lucid. For more information, see the `.hh` files in the `include` directory and appendix A. They are well commented and are very informative about options that are not covered in this short manual. For those wishing to understand and modify the sources of the G4Lucid package, section 4 describes how the G4Lucid package was designed.

4 Programmer's Manual

This manual is intended to make it easier to maintain the G4Lucid package. It describes how the classes themselves work and how they work together. Users of the G4Lucid package should not have to read this section unless modifications need be made to the software or a greater understanding of its internal workings is desired.

4.1 Introduction to the Classes

Even though there are many classes making up the G4Lucid package, we can group them into four groups. The first group has only the class `G4Lucid`. This class is responsible for containing the events and controlling options for the run. The second group also has only one class, `G4LucidEvent`. This class creates symbol tables entries for Lucid from the variables that it contains. It also writes the data to the Lucid file. The third group of classes are those classes the directly or indirectly inherit the functionality of `G4VLucidVariable`. The user gives these classes data to pass to `G4LucidEvent` in order to write them to the Lucid file. The forth group of classes consists of any class that is not in the first three groups.

4.2 Events and `G4Lucid`

The class `G4Lucid` is a singleton. This means that only one instance of this class can exist at any time. On the other hand, many events are required in order to create simulated Lucid data. Thus, many instances of the class `G4LucidEvent` can exist and pointers to these instances are stored in the singleton instance of `G4Lucid`.

The singleton status of `G4Lucid` is enforced by using a private constructor and the static member function `GetLucidPointer()`. A pointer to the singleton instance of `G4Lucid` can be obtained at any time by using the command

```
G4Lucid * lucidPtr = G4Lucid::GetLucidPointer();
```

which will return a pointer to an already existing instance, or create a new instance using the private constructor.

The class `G4Lucid` is used to store pointers to events, control the beginning and end of a data run and to store general information about the Lucid run. Pointers to the events are added by using the function `AddLucidEvent(G4LucidEvent *)` and retrieved by using the function `GetLucidEvent()` with the latter being overloaded to take either the event's name or number as an argument. These functions store the events in an array. When an event is added, the array size is increased by one and the event is added to the end of the array. Events cannot be removed except by deleting the `G4Lucid` instance

```
delete lucidPtr;
```

and starting fresh by using `AddLucidEvent(G4LucidEvent *)` to create a new instance. Deleting the instance of `G4Lucid` will delete all of the events.

At the beginning of a run, the user should call the function `BeginRun()`. This function interfaces with the Lucid library `liblucid.a`. The function `BeginRun()` first declares that we will define our own symbol table.

```
LF_rsymtab( NULL );
```

It then opens the Lucid file for writing using the filename, the name of the experiment and a description.

```
if( LF_open( file, name, description ) ) {
   G4cerr << "G4Lucid::BeginRun() -> FATAL ERROR: Error in LF_open; filename = "
      << file << G4endl;
   return -1; }
```

These can be set by using the functions `SetFilenamePrefix()`, `SetExpName()` and `SetExpDescription()` and are stored in class variables as instances of `G4String`. Last, `G4Lucid` builds the symbol tables for all of the events by calling their respective `MakeLucidVarList()` functions in sequence.

```
for(G4int i = 0; i < numberEvents; i++) {
   eventArray[i] -> MakeLucidVarList(); }
```

This completes the process of opening the Lucid file and building the symbol table. Now data can be written to the Lucid file.

To close the Lucid file and stop the run, the user should call `EndRun()`. This will call the Lucid function required to close the Lucid file.

```
LF_close();
```

After this operation is complete, data can no longer be written to the Lucid file.

To finish this discussion, one should make the observation that `G4Lucid` is declared to be a friend of `G4LucidEvent`. This is so that `G4Lucid` can access two functions of `G4LucidEvent` that no one else should be able to access. These functions are `MakeLucidVarList()`, which has been described above, and `SetNumber(short numb)`, which sets the event's identification number. The function `SetNumber(short numb)` is called only when the event is added to `G4Lucid`.

4.3 Variables and Inheritance

Before we can understand how variables and events interact, we should discuss the structure of the variables. All variables inherit the class `G4VLucidVariable`. Some functions, such as `GetName()` and `GetArraySize()` are implemented here as all variables will have names and array sizes. Other functions, such as `Print()` and `Clear()` are defined here but not implemented. These functions are pure virtual functions, meaning that every class that inherits `G4VLucidVariable` must implement them. However, since the implementations are different, it is impossible to implement a generic function for all variables.

Having the parent class `G4LucidVariable` allows for better abstraction and code reuse as it implements some basic features of all variables and introduces uniformity. However, it may not be sufficient to inherit `G4LucidVariable`. Some classes, like `G4LucidADC` have special requirements. Since an ADC outputs data in short integers the class `G4LucidADC` inherits `G4LucidShort`. Since `G4LucidShort` inherits `G4VLucidVariable`, `G4LucidADC` indirectly inherits `G4VLucidVariable`. Thus `G4LucidADC` is a variable, but as far as Lucid is concerned, it can be treated as a short integer. This allows for code reuse, making the code easier to build and maintain. See figure 2 for the complete inheritance tree of Lucid variables.

4.4 Variables and `G4LucidEvent`

The class `G4LucidEvent` is by far the largest class in the G4Lucid package making up about one third of the total code. This size is due to the tasks of storing each type of Lucid variable, creating the symbol table entry for each Lucid event and outputting data each time a Lucid event occurs.

Events can be added by using the function `AddLucidVariable()`. This function is overloaded and handles each type of Lucid event differently. Different types of events are placed in different

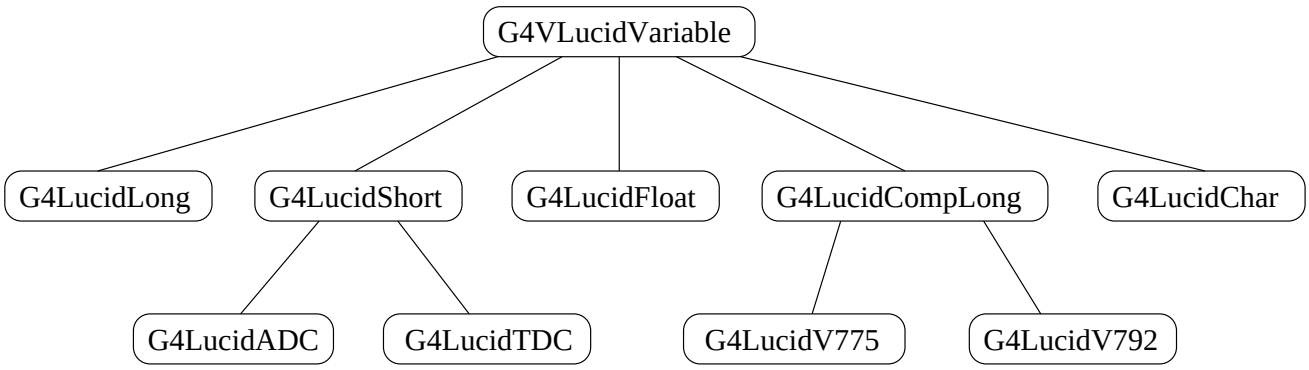

Figure 2: Inheritance map for variables.

arrays. Each time a variable is added, the array size is increased by one and the variable is added to the end of the array. Since different types of variables are stored in different arrays, we can set pointers to variables by using the 'get' functions such as `GetLucidFloat()` and `GetLucidADC()`. These functions take either the variable's name or number as an argument.

The function `GetLucidFloat()` will return a pointer to an object of type `G4LucidFloat`. However, internally `G4LucidEvent` stores all variables as `G4VLucidVariable` and only performs a cast as the pointer is being returned. There are two reasons for storing all variables as pointers to a `G4VLucidVariable`. The first is code simplification in instances where the same operation is performed on all variables. An example is clearing all variables in the event. The following code clears all the variables in this event.

```
for(G4int i = 0; i < numb_chains; i ++){
   for(G4int j = 0; j < numbVar[i]; j++) {
      varArray[i][j]->Clear(); }}
```

The first loop iterates through all different types of variables. The second loop iterates through all variables of the type. In this way, the clear operation is performed in a very simple way. Also, this scheme makes it easier to expand the G4Lucid package by adding different types of variables. Section 4.5 is dedicated to this topic.

The pointers to the variables are stored in an array and only variables of the same type are stored in the same array. The declaration for the arrays that store the variables are as follows.

```
G4VLucidVariable  ** varArray[numb_chains];
G4int numbVar[numb_chains];
```

The array of integers `numbVar` is used to store the number of variables of each type. An array of double pointers may appear confusing so let us take a closer look. First, let us define the following values

```
static const G4int numb_chains     = 10;
static const G4int float_chain_no  = 0;
static const G4int double_chain_no = 1;
static const G4int long_chain_no   = 2;
static const G4int short_chain_no  = 3;
```

```
static const G4int tdc_chain_no    = 4;
static const G4int adc_chain_no    = 5;
static const G4int char_chain_no   = 6;
static const G4int clong_chain_no  = 7;
static const G4int v792_chain_no   = 8;
static const G4int v775_chain_no   = 9;
```

that can be used to index the arrays varArray and numbVar. If we want to select the third short integer in the event, we can use

```
G4VLucidVariable * thirdShort = varArray[short_chain_no][2];
```

where short_chain_no is used to select the variable type and 2 is used to select the third variable as arrays are indexed from 0. To get the eighth long as an instance of G4LucidLong, we can use a cast.

```
G4LucidLong * eighthLong = (G4LucidLong *) varArray[long_chain_no][7];
```

The cast is safe as there should only be instances of G4LucidLong in the array indexed with long_chain_no. Now reexamine the example of clearing all of the variables.

```
for(G4int i = 0; i < numb_chains; i ++){
   for(G4int j = 0; j < numbVar[i]; j++) {
      varArray[i][j]->Clear(); }}
```

Once again, the first loop loops through all of the different variable types and the second loop loops through all individual variables.

There is also a master variable list.

```
static G4VLucidVariable ** masterVarList;
static G4int masterVarListSize;
```

This list is static so only one such list exists and it is shared by all events. The only purpose of this list is to delete the variables stored by the events. Since variables can be placed within multiple events, a naive deletion scheme may delete a variable twice. Therefore, we maintain a master variable list which is deleted when the first event is deleted. Thus, deleting one event deletes all of the variables in all events.

Once all of the variables have been added and the user begins the run, G4Lucid will request symbol table information from the event as described in section 4.2. The function MakeLucidVarList() is tasked with making the variable list for the symbol table. It creates an array of Lucidvar structures. The structure Lucidvar is part of the Lucid library liblucid.a. It then fills the array with information about the variables stored in the event. The function then computes the number of padding bytes required so that the number of bytes in the buffer is a multiple of LUCIDALIGN_RECORD, which currently has a value of 4. Last, the function uses LF_symbol() to tell Lucid about the new symbol table.

```
LF_symbol(GetNumber(), charName, lucidVarList, arrayElements);
```

The first argument is the event's number. The second argument is the event's name. The third argument is a pointer to the array of `Lucidvar` structures that contain the information about the variables and the last argument is the number of elements in this array.

The other time that `G4LucidEvent` interacts with the Lucid file is during the writing of data. The function `WriteEvent()` writes the data in the event's variables to the file. It creates an array of characters in order to store the data.

```
char * lucidData = new char[ bufferSize ];
```

where `bufferSize` is chosen to be large enough to store all of the variable's information, the padding bytes and any compressed information. The first part is the regular, fixed length part of the Lucid event. The second part is the variable length part of the event, sometimes called the compressed data area.

The array is filled with the data from the fixed part for all of the events. The padding bytes are set to zero and the compressed data area is filled. Once the array is filled, `LF_event()` is called to store the data.

```
LF_event( GetNumber(), lucidData, bytesInBuffer  );
```

The first argument is the event's identification number. The second argument is the array containing the Lucid data and the third argument is the number of bytes in this array.

The class `G4LucidEvent` may be a little complex. However, it has been designed to make use of the inheritance structure of the variable classes. This allows for more code reuse and, in many cases, simpler code. It has been designed to make the addition of new types of variables as simple as possible. This topic is treated more in the next section.

4.5 Creating a New Lucid Variable

It may be desirable to create a new type of Lucid variable to model a piece of electronics or to make the taking of data more simple. The classes `G4LucidEvent`, `G4LucidShort` and `G4LucidCompLong` have been designed to make the addition of new variables easy. There are two ways to add a new variable: the easy, temporary way and the more difficult, permanent way.

To easily add a variable, create a class, `G4LucidMyVar`, which is the child of an already existing variable. For example, the declaration of `G4LucidMyVar` may look as follows.

```
class G4LucidMyVar : public G4LucidShort
```

Once you have created your class, simply add your new variable to an event using the function `AddLucidVariable()`. Since `AddLucidVariable()` is overloaded, it will default to the parent's version. In the case of a variable that uses `G4LucidShort` as the parent, the C++ compiler will use `G4LucidVariable(G4LucidShort *var)`. When you want to extract the variable from the event, you can use the following statement.

```
G4LucidMyVar * theVar = (G4LucidMyVar *) theEvent->GetLucidShort("the_var_name");
```

This method has the disadvantage that it requires a cast when extracting the variable as `G4LucidEvent` treats an instance of `G4LucidMyVar` as an instance of `G4LucidShort`. However, it is very simple to implement new variables this way because we do not need to modify `G4LucidEvent`.

To permanently add a new variable based on G4LucidShort or G4LucidCompLong we must modify G4LucidEvent. Start by creating your variable G4LucidMyVar which we will assume is a child of G4LucidShort for discussion purposes. The first thing we do is add a new array by incrementing numb_chains by one. We then must add a new chain identification number myvar_chain_no which is not the same as one of the existing chain numbers. We must add the following functions to add and get the new types of variables.

```
G4int AddLucidVariable(G4LucidMyVar *var);
G4LucidMyVar * GetLucidMyVar(G4int intNo);
G4LucidMyVar * GetLucidMyVar(G4String name);
inline G4int GetNumberMyVar(void)   {return numbVar[myvar_chain_no]; }
```

Due to the inheritance structure of variables and the amount of code reuse we should be able to implement these functions easily by copying and pasting existing functions, then changing the function name and the chain number.

Now that the event has pointers to the new variables, we need to include the new variable type in the creation of the Lucid symbol table and the writing of data to the Lucid file. In the function MakeLucidVarList() go to the loop where we take care of all of the classes derived from G4LucidShort. Increment the number of iterations by one and add a statement similar to the following one.

```
else if(j== 3){
   shortArray = varArray[myvar_chain_no];
   numbShorts = numbVar[myvar_chain_no];}
```

This will cause G4LucidEvent to process your new variable as a G4LucidShort, which is its parent class.

Now that we have added the new variable to the symbol table, we must add it to the function that writes the data to the Lucid file. This operation is performed by the function WriteEvent(). We simply must find the loop for short integers and ensure that we add the same code as before to this loop.

For a class that inherits G4LucidCompLong there is one more step that does not apply to G4LucidShort. We must find the place where the data is added to the compressed area of the Lucid data. Once again, we add another iteration to the loop and add code similar to the above code to the loop.

Because of the inheritance structure of variables and the amount of code reuse in G4LucidEvent, adding new variables is relatively easy. There are two ways to add variables which differ by whether or not we modify G4LucidEvent. Each of these different ways is applicable to different situations.

4.6 The Class G4LucidMessenger

While the class G4LucidMessenger is an important part of the G4Lucid package, it has not been discussed much before now. This is because the user should have no direct interaction with this class. Instead, this class is used to create the /lucid/ directory on the GEANT4 command line interface, defines the commands in this directory and performs actions when one of those commands are used.

This class inherits the GEANT4 user interface messenger class G4UImessenger. It implements a function void SetNewValue(G4UIcommand*, G4String) which GEANT4 uses to execute commands.

Only one instance of G4LucidMessenger should exist at any given time. This is enforced by having the singleton class G4Lucid create an instance of G4LucidMessenger in its constructor and delete the instance in its destructor.

4.7 The Class G4LucidDouble

The class G4LucidDouble is partially implemented. Base functionality exists, however there are problems with its use. When using G4LucidDouble reading the output Lucid file with lucidview shows the following error message.

```
Aug 23 17:52:38 2004         1  New session by ward.
        Experiment 'Cell Efficiency'
        Build name 'GEANT 4 Simulation'
Aug 23 17:52:38 2004         1  Format 1.25 1.25.
Aug 23 17:52:38 2004         1  Event 1 (startup) symbol table.
          character*2 <<pad>>
Aug 23 17:52:38 2004         2  Event 2 (neutron_emitted) symbol table.
            integer*4 demo
               real*8 init_energy
          character*2 <<pad>>
 *****SNIP******
Aug 23 17:52:38 2004         7  Event 7 (windup) symbol table.
          character*2 <<pad>>
Aug 23 17:52:38 2004         1  Begin run 1 by *OFFLINE*.
Aug 23 17:52:38 2004         1  120 data bytes.  Run 1.
        Event 2 (neutron_emitted)
                demo:          0
                init_energy:              0
                <<pad>>: [2]
                *** 4 bytes left.  No more variable descriptors.
Aug 23 17:52:38 2004         1  End run 1 by *OFFLINE*.
                1 event 2 (neutron_emitted).
```

Therefore G4LucidFloat should be used in place of G4LucidDouble until this error can be corrected. This is not a hindrance since a 32 bit floating point number should retain enough decimal places for any data.

A Available Classes and Functions

This appendix is to be used as a reference for the classes and functions in the G4Lucid package. It contains all of the classes that are useful to the user and their public member functions. Each class also has a destructor, but these are to be taken for granted. Also, functions that use the `inline` and `virtual` keywords are not denoted as such. For each class listed the « operator is overloaded and will call the `Print(std::ostream *os)` function.

A.1 Class `G4Lucid`

Table 1: Public functions for the class `G4Lucid`

Function	Description
`static G4Lucid * GetLucidPointer(void)`	Gets a pointer to the one-and-only G4Lucid object.
`G4int BeginRun(G4int runNo)`	Begin run using run number runNo. Returns the run number or `-1` on error.
`G4int BeginRun(void)`	Begin run using the last run number plus one. Returns the run number or `-1` on error.
`G4int EndRun(void)`	Ends run and returns the run number.
`G4int AddLucidEvent(G4LucidEvent *lucidEvt)`	Adds a Lucid Event and returns the event number. Returns `-1` if there was an error.
`G4LucidEvent * GetLucidEvent(G4String eventName)`	Gets Lucid event by name. Returns `0` on error with error message.
`G4LucidEvent * GetLucidEvent(G4int eventNo)`	Gets Lucid event by number. Returns `0` on error with no error message.
`void Clear(void)`	Conditionally clears all variables.
`void ForceClear(void)`	Forcefully clears all variables.
`G4String GetExpName(void)`	Get the experiment name.
`G4String GetExpDescription(void)`	Get the experiment description.
`void SetExpName(G4String name)`	Set the experiment name.
`void SetExpDescription(G4String name)`	Set the experiment description.
`G4bool IsRunning(void)`	Returns `true` if a run is in progress.
`void SetFilenamePrefix(G4String prefix)`	Sets the filename prefix.
`G4String GetFilenamePrefix(void)`	Gets the filename prefix.
`void Print(std::ostream *os)`	Prints information about events and variables.

A.2 Class G4LucidEvent

Table 2: Public functions for the class G4LucidEvent which do not deal with individual variables.

Function	Description
G4LucidEvent(G4String eventName)	Constructor.
G4int WriteEvent(void)	Writes the event to the Lucid file. Returns 0 on success, -1 on failure and 1 if writing is disabled.
G4int WriteEventClear(void)	Same as above but performs a conditional clear.
void Clear(void)	Conditionally clears all variables.
void ForceClear(void)	Forcefully clears all variables.
G4String GetName(void)	Gets the name of the event.
short GetNumber(void)	Gets the number of the event.
void EnableWriting(void)	Enable writing the event to the file.
void DisableWriting(void)	Disable writing the event to the file.
G4bool QueryWriting(void)	Query whether writing is enabled.
G4int GetNumberVariables(void)	Get the number of variables in this event.
size_t GetNumberBytes(void)	Get the number of bytes required to store the variables.
size_t GetMaxCompressBytes(void)	Get the maximum number of bytes required to store the data which goes into the compressed memory area of the Lucid event.
void Print(std::ostream *os)	Prints information about the event and its variables.

Table 3: Public functions for the class G4LucidEvent which get the number of each type of variable.

Function	Function	Description
G4int GetNumberShort(void)	G4int GetNumberLong(void)	Get the number of variables of the given type from the event.
G4int GetNumberDouble(void)	G4int GetNumberFloat(void)	
G4int GetNumberChar(void)	G4int GetNumberCompLong(void)	
G4int GetNumberADC(void)	G4int GetNumberTDC(void)	
G4int GetNumberV792(void)	G4int GetNumberV775(void)	

Table 4: Public functions for the class G4LucidEvent which add variables to the event.

Function	Description
G4int AddLucidVariable(G4LucidShort *var) G4int AddLucidVariable(G4LucidLong *var) G4int AddLucidVariable(G4LucidFloat *var) G4int AddLucidVariable(G4LucidDouble *var) G4int AddLucidVariable(G4LucidTDC *var) G4int AddLucidVariable(G4LucidADC *var) G4int AddLucidVariable(G4LucidChar *var) G4int AddLucidVariable(G4LucidCompLong *var) G4int AddLucidVariable(G4LucidV792 *var) G4int AddLucidVariable(G4LucidV775 *var)	Adds a Lucid variable to the event. Returns the variable's number on success and -1 on failure.

Table 5: Public functions for the class G4LucidEvent which get variables from the event.

Function	Description
G4LucidShort * GetLucidShort (G4int intNo) G4LucidLong * GetLucidLong (G4int intNo) G4LucidFloat * GetLucidFloat (G4int intNo) G4LucidDouble * GetLucidDouble (G4int intNo) G4LucidChar * GetLucidChar (G4int intNo) G4LucidTDC * GetLucidTDC (G4int intNo) G4LucidADC * GetLucidADC (G4int intNo) G4LucidCompLong * GetLucidCompLong (G4int intNo) G4LucidV792 * GetLucidV792 (G4int intNo) G4LucidV775 * GetLucidV775 (G4int intNo) G4LucidShort * GetLucidShort (G4String name) G4LucidLong * GetLucidLong (G4String name) G4LucidFloat * GetLucidFloat (G4String name) G4LucidDouble * GetLucidDouble (G4String name) G4LucidChar * GetLucidChar (G4String name) G4LucidTDC * GetLucidTDC (G4String name) G4LucidADC * GetLucidADC (G4String name) G4LucidCompLong * GetLucidCompLong (G4String name) G4LucidV792 * GetLucidV792 (G4String name) G4LucidV775 * GetLucidV775 (G4String name)	Gets a Lucid variable from the event. Returns 0 on failure. Functions taking the variable's name print an error message on failure. Functions taking the variable's number return silently on failure.

A.3 Class G4VLucidVariable

These functions are inherited by all variables. This class is a virtual class so no instances of it can exist.

Table 6: Public functions for the class G4VLucidVariable

Function	Description
`void Clear(void)`	Clears the variable.
`void ConditionalClear(void)`	Conditionally clears the variable.
`void EnableConditionalClear(void)`	Enable conditional clearing.
`void DisableConditionalClear(void)`	Disable conditional clearing.
`G4bool IsConditionalClearEnabled(void)`	Query conditional clearing.
`G4String GetName(void)`	Gets the name of the variable.
`G4int GetArraySize(void)`	Gets the array size for this variable.
`size_t GetNumberBytes()`	Gets the number of bytes required to store the variable's data in the Lucid file.
`void Print(std::ostream *os)`	Print information about the variable.

A.4 Classes Derived from G4VLucidVariable

These classes inherit `G4VLucidVariable` and all of its functions, which are listed in table 6.

Table 7: Public functions for the class G4LucidShort

Function	Description
`G4LucidShort(G4String varName, G4int size = 1)`	Constructor takes the name of the variable and its array size.
`G4int Set(short value,G4int index = 0)`	Sets the variable at index `index`. Returns `index` on success and `-1` on failure.
`short Get(G4int index)`	Returns the value for the specified index.
`short Get(void)`	Returns the value for `index` = 0.

Table 8: Public functions for the class G4LucidLong

Function	Description
`G4LucidLong(G4String varName, G4int size = 1)`	Constructor takes the name of the variable and its array size.
`G4int Set(G4long value,G4int index = 0)`	Sets the variable at index `index`. Returns `index` on success and `-1` on failure.
`G4long Get(G4int index)`	Returns the value for the specified index.
`G4long Get(void)`	Returns the value for `index` = 0.
`G4int Increment(G4int index = 0)`	Increments the variable by one.

Table 9: Public functions for the class G4LucidFloat

Function	Description
`G4LucidFloat(G4String varName, G4int size = 1)`	Constructor takes the name of the variable and its array size.
`G4int Set(G4float value,G4int index = 0)`	Sets the variable at index `index`. Returns `index` on success and `-1` on failure.
`G4float Get(G4int index)`	Returns the value for the specified index.
`G4float Get(void)`	Returns the value for `index` = 0.

Table 10: Public functions for the class G4LucidChar

Function	Description
`G4LucidChar(G4String varName, G4int size)`	Constructor takes the name of the variable and the size of the string.
`G4int Set(G4String value)`	Set the variable using a `G4String` object.
`G4int Set(const char * value)`	Set the variable using an array of characters.
`G4String Get(void)`	Get the variable's data in a `G4String`.
`char GetChar(G4int index)`	Get the character at `index`.

Table 11: Public functions for the class G4LucidCompLong

Function	Description
`G4LucidCompLong(G4String intName, G4int size)`	Constructor the takes name of the variable and its array size.
`G4int Set(G4long value,G4int index)`	Sets the variable at index `index`. Returns `index` on success and `-1` on failure.
`G4long Get(G4int index)`	Returns the value for the specified index.
`size_t GetNumberBytes(void)`	Gets the size of the structure `Lucidcompressoffset` which is the only part placed into the fixed-size area of the Lucid event.
`G4int GetMaxCompressArraySize(void)`	Gets the maximum size of the array needed to store the compressed data.
`size_t GetMaxCompressBytes(void)`	Gets the maximum number of bytes needed to store the compressed data.
`size_t GetCompressBytes(void)`	Gets the actual number of bytes needed to store the compressed data.
`const void * GetCompressData(void)`	Returns a pointer to the compressed data.

A.5 Classes Derived from `G4LucidShort`

These classes inherit `G4LucidShort` and all of its functions, which are listed in tables 6 and 7.

Table 12: Public functions for the class `G4LucidADC`

Function	Description
`G4LucidADC(G4String name, G4int bins, G4double perBin, G4String units = "", G4int size = 1)`	Constructor takes the name of the variable, the number of bins and resolution of the ADC, the type of units (i.e. "Energy") to use and the number of channels.
`G4int SetADC(G4double value, G4int index = 0)`	Set the ADC using an analogue value.
`G4double GetADC(G4int index)`	Gets the analogue value from the ADC.
`G4double GetADC(void)`	Same as above, but for channel 0.
`void SetSquishOverflow(G4bool arg)`	Squish overflow data into last bin.
`G4bool GetSquishOverflow(void)`	Determine if overflow squishing is on.
`G4bool GetOverflow(G4int index)`	Returns `true` if there is an overflow in the specified channel.
`G4bool GetOverflow(void)`	Same as above, but for channel 0.
`G4String GetUnits(void)`	Returns the unit system in use.
`void SetUnits(G4String new_units)`	Set unit system to use (i.e. "Energy").
`G4int GetBins(void)`	Get the number of ADC bins.
`G4int SetBins(G4int bins)`	Set the number of ADC bins.
`G4int SetQuantityPerBin(G4double perBin)`	Set the ADC resolution.
`G4double GetQuantityPerBin(void)`	Get the ADC resolution.

Table 13: Public functions for the class G4LucidTDC

Function	Description
`G4LucidTDC(G4String name, G4int bins, G4double timePerBin, G4int size = 1)`	Constructor takes the name of the variable, the number of bins and resolution of the TDC, and the number of channels.
`void SetStartStopTime(G4double start, G4double stop, G4int index = 0)`	Set both the TDC start and stop times.
`void SetStartTime(G4double time, G4int index = 0)`	Set TDC start time.
`void SetStopTime (G4double time, G4int index = 0)`	Set TDC stop time.
`G4double GetStartTime(G4int index)`	Get the start time.
`G4double GetStartTime(void)`	Same as above but for channel 0.
`G4double GetStopTime(G4int index)`	Get the stop time.
`G4double GetStopTime(void)`	Same as above but for channel 0.
`G4int GetBins(void)`	Get the number of TDC bins.
`G4int SetBins(G4int bins)`	Set the number of TDC bins.
`G4int SetTimePerBin(G4double perBin)`	Set the TDC resolution.
`G4double GetTimePerBin(void)`	Get the TDC resolution.
`void SetSquishOverflow(G4bool arg)`	Squish overflow data into last bin.
`G4bool GetSquishOverflow(void)`	Determine if overflow squishing is on.
`G4bool GetOverflow(G4int index)`	Returns `true` if there is an overflow in the specified channel.
`G4bool GetOverflow(void)`	Same as above, but for channel 0.

A.6 Classes Derived from G4LucidCompLong

These classes inherit G4LucidCompLong and all of its functions, which are listed in tables 6 and 11.

Table 14: Public functions for the class G4LucidV775

Function	Description
`G4LucidV775(G4String varName, G4double tdcRes = 0.1*ns)`	Constructor takes the name of the variable and the TDC resolution.
`G4int SetStartStopTime(G4double start, G4double stop, G4int index)`	Sets the TDC start and stop times for channel `index`.
`G4int SetStartTime(G4double time, G4int index)`	Sets the TDC start time for channel `index`.
`G4int SetStopTime(G4double time, G4int index)`	Sets the TDC stop time for channel `index`.
`void SetCommonStartTime(G4double time)`	Sets start time for all channels.
`void SetCommonStopTime(G4double time)`	Sets stop time for all channels.
`G4double GetStartTime(G4int index)`	Gets start time for channel `index`.
`G4double GetStopTime(G4int index)`	Gets stop time for channel `index`.
`void HexDump(std::ostream *os)`	Similar to `Print()` but in hexadecimal.
`void HexDump(void)`	Same as above, but prints to standard output.
`unsigned short GetChannels(void)`	Gets the number of channels in the TDC.
`unsigned short GetBins(void)`	Gets the number of bins in a TDC channel.
`G4double GetTimePerBin(void)`	Gets the TDC resolution.

Table 15: Public functions for the class G4LucidV792

Function	Description
`G4LucidV792(G4String varName, G4double qdcRes = 1.0*keV, G4long pedestalBin = 0)`	Constructor takes the name of the variable, the QDC resolution and the pedestal bin.
`G4int SetQDC(G4double value, G4int index)`	Sets the analogue value for channel `index`. Returns `index` on success and `-1` on failure.
`G4double GetQDC(G4int index)`	Returns the analogue value for channel `index`.
`void HexDump(std::ostream *os)`	Similar to `Print()` but in hexadecimal.
`void HexDump(void)`	Same as above, but prints to standard output.
`void SetPedistal(G4long ped)`	Sets the pedestal bin for all channels.
`G4long GetPedistal(void)`	Gets the pedestal bin.
`void SetZeroSuppression(G4bool zsup)`	Turn zero suppression on or off.
`G4bool GetZeroSuppression(void)`	Determine if zero suppression is on or off.
`unsigned short GetChannels(void)`	Gets the number of channels in the QDC.
`unsigned short GetBins(void)`	Gets the number of bins in a QDC channel.
`G4double GetQuantityPerBin(void)`	Gets the QDC resolution.

B 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

[SPIR-100] D. Murray, et al., *LUCID User's Guide.* "http://nucleus.usask.ca" under "Reports".

[G4] GEANT4 Collaboration, *Nucl. Inst. and Meth. A* 503 (2003) 250.

[Root] Rene Brun and Fons Rademakers, ROOT - An Object Oriented Data Analysis Framework, Proceedings AIHENP'96 Workshop, Lausanne, Sep. 1996, Nucl. Inst. & Meth. in Phys. Res. A 389 (1997) 81-86. See also http://root.cern.ch/

[SPIR-151] W.A. Wurtz and Rob Pywell, RLucid: Lucid to Root Conversion Package "http://nucleus.usask.ca under "Reports"