Qt-Based Implementation of Low Level ROOT Graphical Layer

By V.Fine

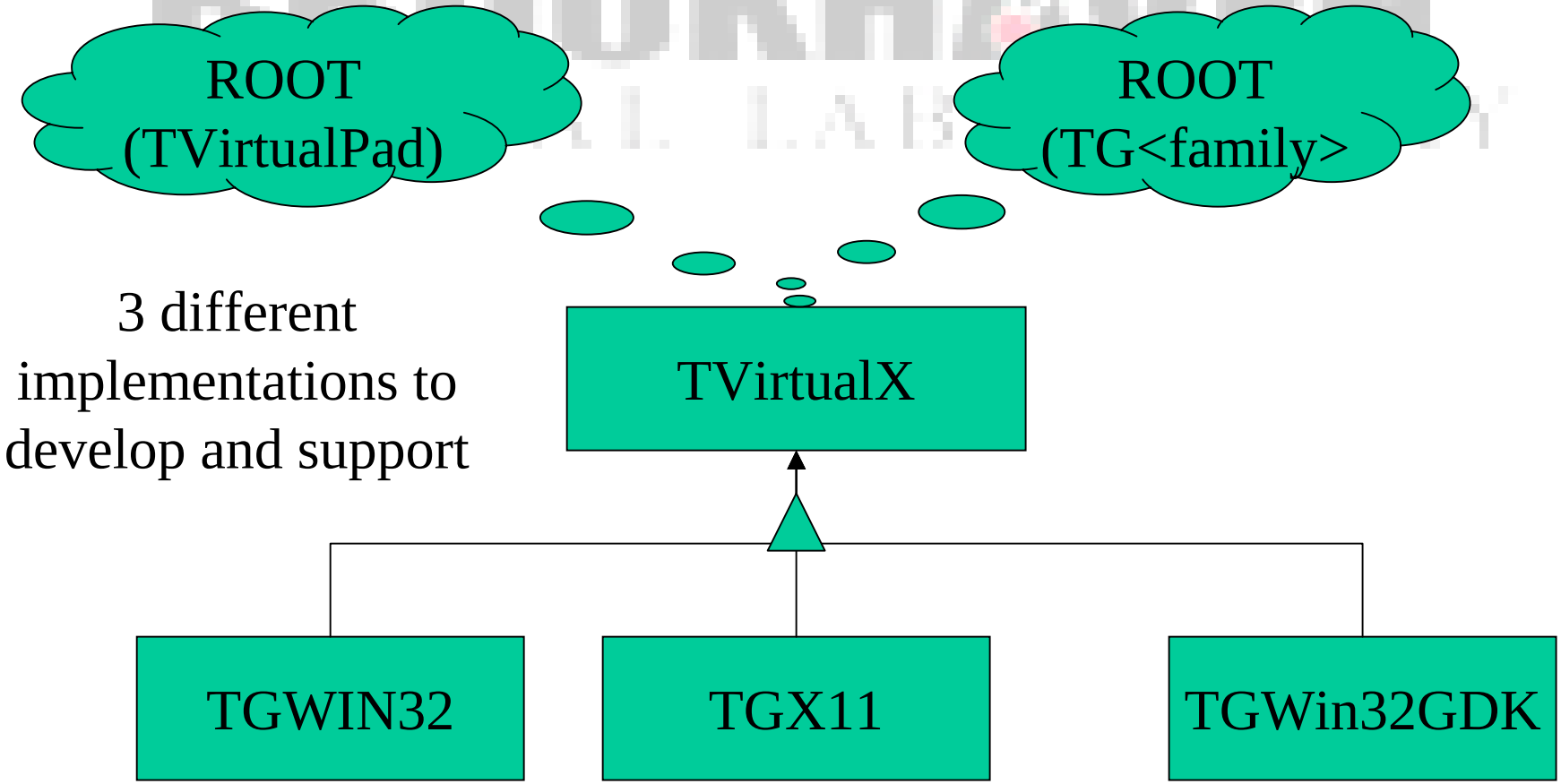
ROOT Low Level Graphics Level

It is well-known that ROOT package has been ported to many different platforms which include the various flavors of UNIX as well as Windows.

Such portability is provided thanks a few built-in “abstract” interfaces that separate the core ROOT logic from particular operating systems and low-level graphics layer.

Even though such architecture allows keeping ROOT platform independent, it still needs to contain the separate platform depended (X11 or Win32) parts for each platform one wants to port ROOT to. That is difficult to maintain and error prone.

ROOT low level GUI interface



Why Qt?

On the other hand, most tasks to be performed have no ROOT specific and have been successfully solved by other packages.

- *Qt package from TrollTech AS* was especially attractive not only due to its superior quality and high level technical support but because it comes with the source code and tools to build it in place (including a commercial version for Windows).
- The rich set of Qt documentation can be found on Web and available from the leading publishers as well.
- Qt is a multi-platform C++ application framework that developers can use to write single-source applications that run-natively-on Windows, Linux, Unix, Mac OS X and embedded Linux.
- Qt has been used to build thousands of successful commercial applications worldwide,
- and is the basis of the open source KDE desktop environment.

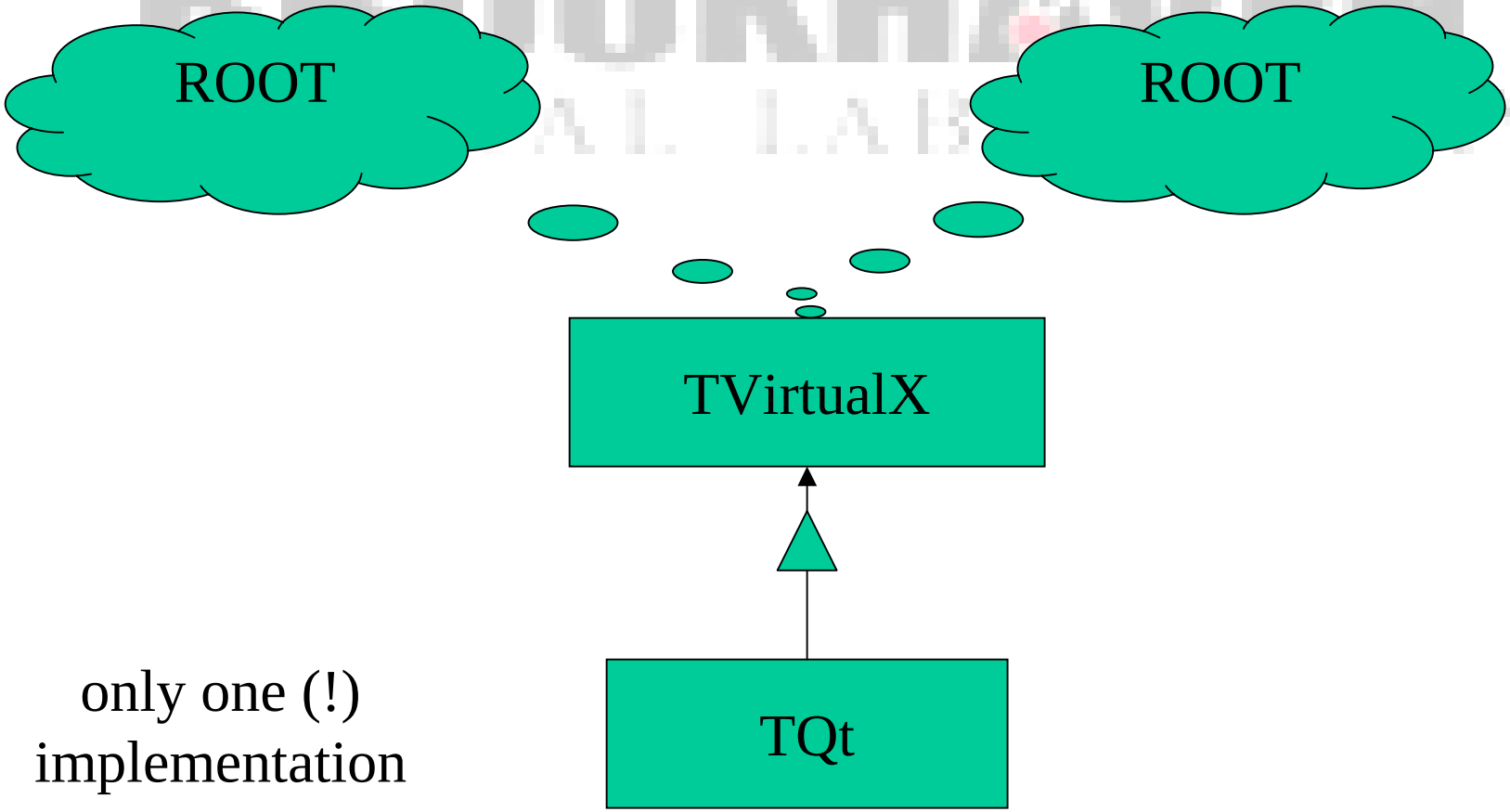
Why Qt ?

It does what it promises !

Myself have been confused no time with Qt documentation and functionality

The plain implementation of TVirtualX that doesn't bring any new feature can not justify the efforts to introduce a new extra layer and external dependency of the ROOT package

ROOT Qt-based GUI interface



only one (!)
implementation

Qt-based implementation

- The Qt-based layer comes with 2 separate subpackages those are to be compiled into two separate share libraries (DLLs), **`libQt`** and **`libQtGui`**.
- The first is mandatory and it provides the C++ class **`TQt.`**
- The second class is to provide the concrete implementations of the abstract interfaces defined by **`TGuiFactory`** class, namely:
  - **`TCanvasImp`,**
  - **`TBrowserImp`,**
  - **`TContextMenuImp`,**
  - **`TControlBarImp`,**
  - **`TInspectorImp`**
  - **`TGLViewerImp etc etc`**

The final goal here is to create a set of simple Qt-widgets those can be used to create sophisticated GUI interfaces with the standard components (for example a custom version of **`TBrowser`**)

ROOT vs Qt

- To run ROOT
  - Create **`TApplication`**
  - Enter ROOT event loop **`TApplication::Run`**
- To run Qt
  - Create **`QApplication`**
  - Enter Qt event loop **`QApplication::exec()`**

This implementation:

`TApplication` → **`QApplication`**

`TApplication::Run()` → **`QApplication::exec()`**

Bottom line: `No need to deal with QApplication!`

Qt Root “Hello Word”

1. `t1.pro` – Qt “Hello Word” project file

```
TEMPLATE = app
CONFIG           += qt warn_on release
HEADERS          =
SOURCES          = main.cpp
TARGET           = t1
REQUIRES=small-config
```

2. `main.cxx` – Qt “Hello Word” source file

```
#include <qapplication.h>
#include <qpushbutton.h>
int main( int argc, char **argv )
{
    QApplication a( argc, argv );
    QPushButton hello( "Hello world!", 0 );
    hello.resize( 100, 30 );
    a.setMainWidget( &hello );
    hello.show();
    return a.exec();
}
```

Change source to make C++ class:

```
#include <qapplication.h>
#include <qpushbutton.h>
int main( int argc, char **argv )
{
 QApplication a( argc, argv );
 QPushButton hello("Hello world!",0);
 hello.resize( 100, 30 );
 a.setMainWidget( &hello );
 hello.show();
 return a.exec();
}
```

helloWord.h:

```
class helloWord {
  public:
     helloWord(){ main();}
     int main();
};
```

helloWord.cxx:

```
#include “helloWord.h”
#include <qwidget.h>
#include <qpushbutton.h>
static helloWord  t1;
int helloWord::main()   {
     //  QApplication a( argc, argv );
     static QWidget a;
     static QPushButton hello( "Hello world!", &a );
      a.resize( 100, 30 );
      // a.setMainWidget( &hello );
      a.show();
      // return a.exec();
 }
```

Adjust Qt project file:

```
TEMPLATE = app
CONFIG           += qt warn_on release
HEADERS          =
SOURCES          = main.cpp
TARGET           = t1
REQUIRES=small-config
```

```
TEMPLATE = lib
CONFIG           += qt warn_on release dll
HEADERS          =
SOURCES          = helloWord.cxx
TARGET           = t1
win32:INCLUDEPATH = %ROOTSYS%/include
unix:INCLUDEPATH  = $(ROOTSYS)/include
win32:LIBS        = %ROOTSYS%\lib\libCore.lib %ROOTSYS%\lib\libQt.lib
```

Now we can build and execute it:

- Build makefile (for either platform):

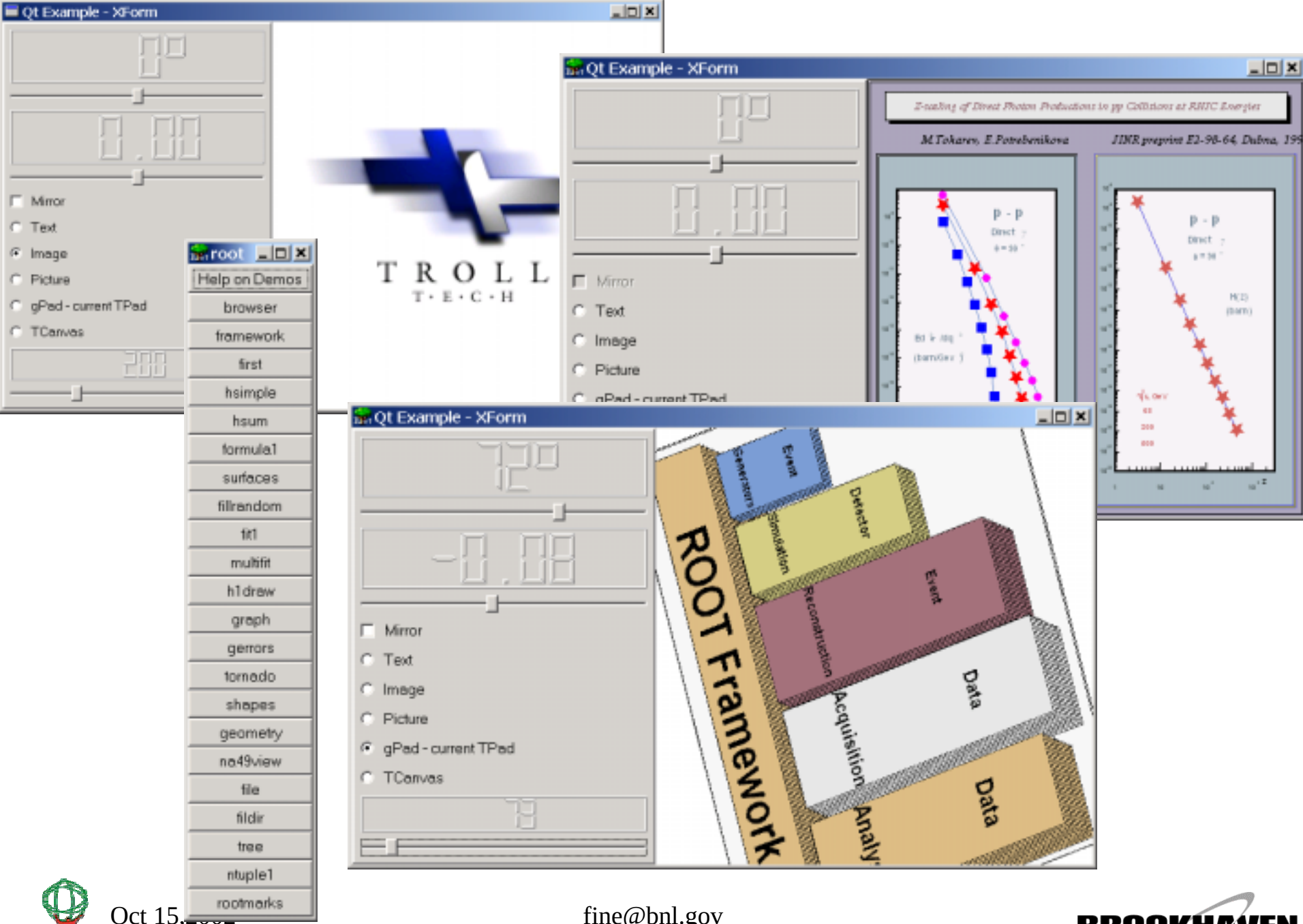
```
qmake
helloWord.pro
```

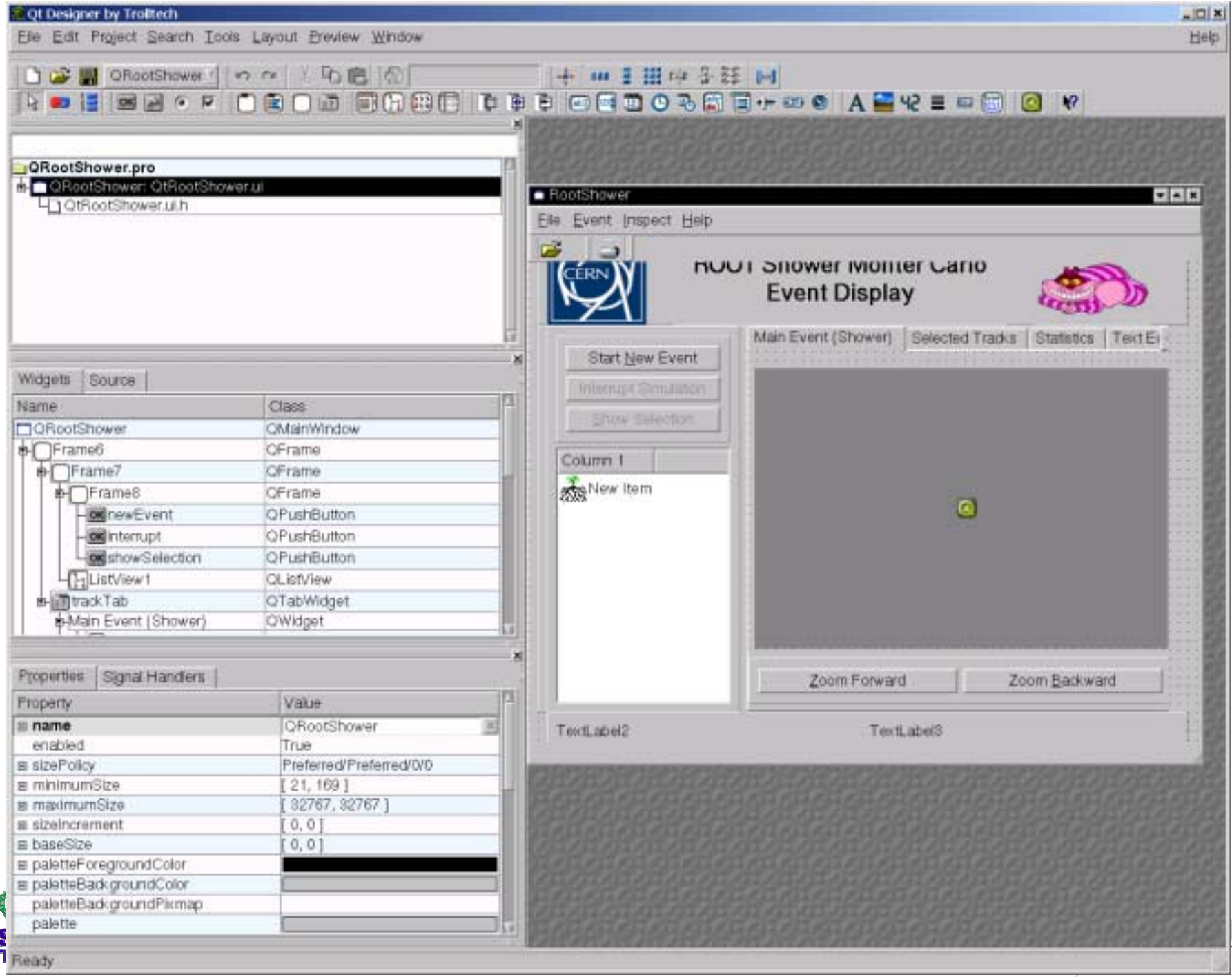
- Create a share library or DLL (for either platform):

```
make
```

- Start ROOT and load it:

```
root<cr>
root [0] gQt->LoadQt("libt1")
```


Qt Example - XForm
Mirror
Text
Image
Picture
gPad - current TPad
TCanvas
root
Help on Demos
browser
framework
first
hsimple
hsum
formula1
surfaces
fillrandom
fit1
multifit
h1draw
graph
gerrors
tornado
shapes
geometry
na49view
file
fildir
tree
ntuple1
rootmarks
TROLL
T · E · C · H
Qt Example - XForm
Mirror
Text
Image
Picture
Qt Example - XForm
Mirror
Text
Image
Picture
gPad - current TPad
TCanvas
ROOT Framework
Event Generators
Detector Simulation
Event Reconstruction
Data Acquisition
Data Analy


Qt Designer by Trolltech
File Edit Project Search Tools Layout Preview Window
Help
QRootShower
QRootShower.pro
QRootShower: QtRootShower.ui
QtRootShower.ui.h
Widgets
Source
Name
Class
QRootShower
QMainWindow
Frame6
QFrame
Frame7
QFrame
Frame8
QFrame
newEvent
QPushButton
Interrupt
QPushButton
showSelection
QPushButton
ListView1
QListView
trackTab
QTabWidget
Main Event (Shower)
QWidget
Properties
Signal Handlers
Property
Value
name
QRootShower
enabled
True
sizePolicy
Preferred/Preferred/0/0
minimumSize
[21, 169]
maximumSize
[32767, 32767]
sizeIncrement
[0, 0]
baseSize
[0, 0]
paletteForegroundColor
paletteBackgroundColor
paletteBackgroundPixmap
palette
Ready
RootShower
File Event Inspect Help
CERN
Event Display
Main Event (Shower)
Selected Tracks
Statistics
Start New Event
Interrupt Simulation
Show Selection
Column 1
New Item
Zoom Forward
Zoom Backward
TextLabel2
TextLabel3

Where?
http://root.bnl.gov

Download it right NOW.
And try !!!

Just click “Download” and “Run” from there

Qt-ROOT CVS

The source code is available via CVS repository as well.

To check out the sources, you need to be running CVS 1.10 or later (check by doing cvs -v), and have your **$CVSROOT** set to

:pserver:cvs@usatlas.org:/rootbnl/cvs

The password for user **cvs** is **cvs**.

Two cvs commands will do the job:

- **cvs -d :pserver:cvs@usatlas.org:/rootbnl/cvs login**
- **cvs -d :pserver:cvs@usatlas.org:/rootbnl/cvs co root**

Conclusion

- The present approach allows the ROOT developer as well as ROOT user to work with code that has no X11/ WIN32 graphics subsystem dependencies
- and at the same time opens unrestricted access to a rich set of ready-to-use commercial and
- free GUI Qt-based widgets.
- The Qt-based version was tested on Unix and Windows and available
- from the Physics Application Software (PAS) group of the Brookhaven National Laboratory CVS repository (http://root.bnl.gov).

Bottom line:

Now we can create a multi-platform ROOT-based applications with advanced GUI Qt-based interface