JÜLICH
FORSCHUNGSZENTRUM

CUBE 4.3.1 – Cube GUI Plugin Developer Guide

Commented example of a CUBE Gui Plugin

March 19, 2015

The Scalasca Development Team
scalasca@fz-juelich.de

Copyright

Copyright © 1998–2015 Forschungszentrum Jülich GmbH, Germany

Copyright © 2009–2015 German Research School for Simulation Sciences GmbH, Jülich/Aachen, Germany

All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- Neither the names of Forschungszentrum Jülich GmbH or German Research School for Simulation Sciences GmbH, Jülich/Aachen, nor the names of their contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Contents

Copyright iii

1 Cube Plugin Types 1
- 1.1 Cube Plugins . . . 1
- 1.2 Context Free Plugins . . . 1

2 Step by step example for CubePlugin 3
- 2.1 Qt project file . . . 3
- 2.2 SimpleExample.h . . . 3
- 2.3 SimpleExample.cpp . . . 5

3 Step by step example for ContextFreePlugin 7
- 3.1 Qt project file . . . 7
- 3.2 ContextFreePluginExample.h . . . 7
- 3.3 ContextFreeExample.cpp . . . 9

4 Developing plugins 11
- 4.1 Extensive example . . . 11
- 4.2 Problems loading plugins . . . 11

1 Cube Plugin Types

1.1 Cube Plugins

Plugins that derive from CubePlugin depend on a loaded cube file. They can react on user actions, e.g. tree item selection, and may insert a context menu or add a new tab next to the tree views. Examples for this type of plugins are the System Topology Plugin or the Statistics Plugin which are part of the Cube installation.

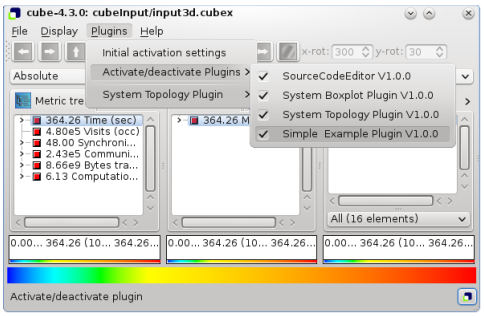

Figure 1.1: plugin menu

1.2 Context Free Plugins

Plugins that derive from ContextFreePlugin are only active if no cube file is loaded. These plugins create or modify Cube objects, which can be loaded and displayed.

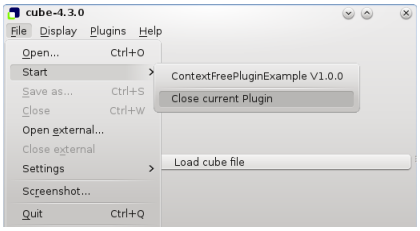

Figure 1.2: context free plugin menu

2 Step by step example for CubePlugin

The following sections describe the steps that are required to create a plugin. For simplicity, a separate project is created and the generated binary will to be copied to the plugin directory of the given cube installation.

2.1 Qt project file

To create a cube plugins, a makefile and source files have to be generated. The makefile can be generated automatically from a Qt project file

First we specify the path to the "cube-config" script of the cube installation. This script delivers correct flags for compiling and linking.

```
CUBE_CONFIG = /opt/cube/bin/cube-config

INCLUDEPATH    += $$system($$CUBE_CONFIG --gui-include-path) $$system($$CUBE_CONF
      IG --cube-include-path)
LIBS           += $$system($$CUBE_CONFIG --gui-ldflags) $$system($$CUBE_CONFIG --
      cube-ldflags)
DESTDIR         = $$system($$CUBE_CONFIG --cube-dir)/plugins

TEMPLATE        = lib
CONFIG         += plugin
HEADERS         = ExampleSimple.h
SOURCES         = ExampleSimple.cpp
TARGET          = $$qtLibraryTarget(ExamplePluginSimple)
```

qmake && make will build the first plugin example libExamplePluginSimple.so. The plugin will be copied to the plugin directory, e.g. /opt/cube/lib64/plugins.

2.2 SimpleExample.h

The example describes a minimal cube plugin, which is inserted as an additional tab next to the SystemTree. It shows the text of the recently selected tree item. The complete source of the example can be found in `$CUBE_INSTALL_PREFIX/share/doc/cube/example/gui/plugin-example`.

Every cube plugin has to derive from CubePlugin. To use Qt's signal and slot mechanism it also has to derive from QObject. If the plugin should be added as a tab next to a tree widget, it has to derive from TabInterface.

```
class SimpleExample : public QObject, public CubePlugin, TabInterface
```

The class header is followed by the following macro definitions:

- `Q_OBJECT` is required to handle signals and slots.
- `Q_INTERFACES( CubePlugin )` tells Qt that the class implements the CubePlugin interface and generates the method `qt_metacast(char*)` to cast the plugin object to CubePlugin using the class name given as as character array.
- For Qt versions >= 5.0 the plugin has to be exported using the `Q_PLUGIN_METADATA()` macro. The unique plugin name "SimpleExamplePlugin" is assigned. For Qt versions < 5.0, `Q_EXPORT_PLUGIN2` has be be used (see Section'SimpleExample.cpp').

```
class SimpleExample : public QObject, public CubePlugin, TabInterface
{
    Q_OBJECT
    Q_INTERFACES( CubePlugin )

#if QT_VERSION >= 0x050000
    Q_PLUGIN_METADATA(IID "SimpleExamplePlugin") // unique plugin name
#endif
```

The class SimpleExample has to implement all pure virtual methods from CubePlugin and TabInterface.

```
public:
    // CubePlugin implementation
    virtual bool cubeOpened( PluginServices* service );
    virtual void cubeClosed();
    virtual QString name() const;
    virtual void version( int& major, int& minor, int& bugfix ) const;
    virtual QString getHelpText() const;

    // TabInterface implementation
    virtual QString label() const;
    virtual QWidget* widget();

private slots:
    void treeItemIsSelected( TreeWidgetType type, TreeItem* item );
private:
    QWidget*        widget_;
    QLabel*         qlabel_;
    PluginServices* service;
};
```

2.3 SimpleExample.cpp

For Qt versions < 5.0, `Q_EXPORT_PLUGIN2` is used to export the plugin. The first argument is a unique name for the plugin, the second the name of the class.

```
#if QT_VERSION < 0x050000
Q_EXPORT_PLUGIN2( SimpleExamplePlugin, SimpleExample );
#endif

#include <QVBoxLayout>
#include <QtPlugin>
#include "ExampleSimple.h"
#include "PluginServices.h"
```

The function `cubeOpened(PluginServices* service)` is the starting point of our plugin. Here we create the main widget, which should be added as a system tab. As our plugin derives from TabInterface, this is done by `service->addTab(SYSTEMTAB, this)`.

If the user selects a tree item, service will emit a corresponding signal. To react on this event, the signal has to be connected to the slot `treeItemIsSelected()` of our plugin class.

The function returns true, if the plugin should be started. It it returns false, the plugin is closed and deleted.

The function `cubeClosed()` is called if the cube file is closed or if the plugin is unloaded by the user. All resources which have been allocated in `cubeOpened` have to be deleted here.

```
bool SimpleExample::cubeOpened( PluginServices* service )
{
    this->service = service;

    // create widget_ and place a label on it
    widget_ = new QWidget();
    qlabel_ = new QLabel( "example string" );
    QVBoxLayout* layout = new QVBoxLayout();
    widget_->setLayout( layout );
    layout->addWidget( qlabel_ );

    service->addTab( SYSTEMTAB, this );

    connect(service, SIGNAL( treeItemIsSelected( TreeWidgetType, TreeItem * )),
               this, SLOT( treeItemIsSelected( TreeWidgetType, TreeItem * ) ));

    return true; // initialisation is ok => plugin should be shown
}

SimpleExample::cubeClosed()
{
    delete widget_;
}
```

Each plugin has to set a version number. If several plugins with the same identifier (see function `name()`) exist, the one with the highest version number will be loaded.

```
void SimpleExample::version( int& major, int& minor, int& bugfix ) const
{
    major  = 1;
    minor  = 0;
    bugfix = 0;
}
```

This function returns the unique plugin name. Only one plugin with this name will be loaded.

```
QString SimpleExample::name() const
{
    return "Simple  Example Plugin";
}
```

The following function returns a text to describe the plugin. It will be used by help menu of the cube GUI.

```
QString SimpleExample::getHelpText() const
{
    return "Just a simple example.";
}
```

The following two functions contain the implementation of TabInterface.

The function `widget()` returns the QWidget that will be placed into the tab, which has been created with `service->addTab` in `initialize()`.

```
QWidget* SimpleExample::widget()
{
    return widget_;
}
```

The function `label()` returns the label of the new tab.

```
QString
SimpleExample::label() const
{
    return "Example Plugin Label";
}
```

This method is a slot, which is called if a tree item is selected. The first arguments shows whether the selected item is part of a metric tree, call tree, flat view or system tree. The second argument provides information about the selected item.

```
void
SimpleExample::treeItemIsSelected( TreeWidgetType type, TreeItem* item )
{
    QString txt = item->getName() + " " + QString::number( item->getValue() );
    qlabel_->setText( txt );
}
```

3 Step by step example for ContextFreePlugin

The following sections describe the steps that are required to create a plugin which derives from ContextFreePlugin. For simplicity, a separate project is created and the generated binary will to be copied to the plugin directory of the given cube installation.

3.1 Qt project file

To create a cube plugins, a makefile and source files have to be generated. The makefile can be generated automatically from a Qt project file

First we specify the path to the "cube-config" script of the cube installation. This script delivers correct flags for compiling and linking.

```
CUBE_CONFIG = /opt/cube/bin/cube-config

INCLUDEPATH    += $$system($$CUBE_CONFIG --gui-include-path) $$system($$CUBE_CONF
      IG --cube-include-path)
LIBS           += $$system($$CUBE_CONFIG --gui-ldflags) $$system($$CUBE_CONFIG --
      cube-ldflags)
DESTDIR         = $$system($$CUBE_CONFIG --cube-dir)/plugins

TEMPLATE        = lib
CONFIG         += plugin
HEADERS         = ContextFreePlugin.h
SOURCES         = ContextFreePlugin.cpp
TARGET          = $$qtLibraryTarget(ContextFreeExamplePlugin)
```

qmake && make will build the first plugin example libContextFreeExamplePlugin.so. The plugin will be copied to the plugin directory, e.g. /opt/cube/lib64/plugins.

3.2 ContextFreePluginExample.h

The example describes a minimal context free plugin. The plugin becames active, if Cube is started without an input file, or if the cube file is closed.

The complete source of the example can be found in `$CUBE_INSTALL_PREFIX/share/doc/cube/example/gui/context-free`.

A context free plugin has to derive from ContextFreePlugin. To use Qt's signal and slot mechanism it also has to derive from QObject.

```
class ContextFreePluginExample : public QObject, public ContextFreePlugin
```

The class header is followed by the following macro definitions:

- `Q_OBJECT` is required to handle signals and slots.
- `Q_INTERFACES( ContextFreePlugin )` tells Qt that the class implements the ContextFreePlugin interface and generates the method `qt_metacast(char*)` to cast the plugin object to ContextFreePlugin using the class name given as as character array.
- For Qt versions >= 5.0 the plugin has to be exported using the `Q_PLUGIN_METADATA()` macro. The unique plugin name "ContextFreePlugin" is assigned. For Qt versions < 5.0, `Q_EXPORT_PLUGIN2` has be be used (see Section'ContextFreeExample.cpp').

```
class ContextFreePluginExample : public QObject, public ContextFreePlugin
{
    Q_OBJECT
    Q_INTERFACES( ContextFreePlugin )
#if QT_VERSION >= 0x050000
    Q_PLUGIN_METADATA( IID "ContextFreePluginExample" )
#endif
```

The class ContextFreePluginExample has to implement all pure virtual methods from ContextFreePlugin.

```
public:
  // ContextFreePlugin interface
  virtual QString name() const;
  virtual void opened( ContextFreeServices* service );
  virtual void closed();
  virtual void version( int& major,int& minor,int& bugfix ) const;
  virtual QString getHelpText() const;


private slots:
    void startAction();
private:
    ContextFreeServices* service;
};
```

3.3 ContextFreeExample.cpp

For Qt versions < 5.0, `Q_EXPORT_PLUGIN2` is used to export the plugin. The first argument is a unique name for the plugin, the second the name of the plugin class.

```
#if QT_VERSION < 0x050000
Q_EXPORT_PLUGIN2( ContextFreePluginExample, ContextFreePluginExample )
#endif

#include "ContextFreePluginExample.h"
#include "ContextFreeServices.h"
#include "Cube.h"
```

The function `opened(ContextFreeServices* service)` is the starting point of our plugin. With `service->getWidget()` we get a widget on Cube's main screen, in which we can place the GUI elements of our plugin. In this example, only one button will be placed on the main screen. Activation of this button will call the slot function `startAction()`.

```
void
ContextFreePluginExample::opened( ContextFreeServices* service )
{
    this->service = service;
    qDebug() << "context free plugin opened";

    QWidget *widget = service->getWidget();
    QVBoxLayout* layout = new QVBoxLayout();
    widget->setLayout( layout );

    QPushButton *but = new QPushButton("Load cube file");
    layout->addWidget( but );

    connect(but, SIGNAL(clicked()), this, SLOT(startAction()) );
}
```

The function `closed()` is called if the plugin gets inactive because a cube file is loaded or the Cube GUI is closed. All resources which have been allocated in `opened()` have to be deleted here.

```
void
ContextFreePluginExample::closed()
{
    qDebug() << "context free plugin closed";
}
```

This function is called, if the user clicks on the Button "Load cube file". Usually, a context free plugin will create cube data. In this small example, it simply loads the cube file which is choosen from a file dialog.

```
void
ContextFreePluginExample::startAction()
{
    QString openFileName = QFileDialog::getOpenFileName( service->getWidget(),
                                                          tr( "Choose a file to op
      en" ),
                                                          "",
                                                          tr( "Cube3/4 files (*cub
      e *cube.gz *.cubex);;Cube4 files (*.cubex);;Cube3 files (*.cube.gz *.cube);;All f
      iles (*.*);;All files (*)" ) );
    cube::Cube *cube = new cube::Cube();
    cube->openCubeReport( openFileName.toStdString() );
    service->openCube(cube); // will be deleted automatically, if user closes cub
      e
}
```

Each plugin has to set a version number. If several plugins with the same identifier (see function `name()`) exist, the one with the highest version number will be loaded.

```
void ContextFreeExample::version( int& major, int& minor, int& bugfix ) const
{
    major = 1; minor = 0; bugfix = 0;
}
```

This function returns the unique plugin name. Only one plugin with this name will be loaded.

```
QString ContextFreePluginExample::name() const
{
    return "ContextFreePluginExample";
}
```

The following function returns a text to describe the plugin. It will be used by help menu of the cube GUI.

```
QString
ContextFreePluginExample::getHelpText() const
{
    return "context free plugin help text";
}
```

4 Developing plugins

4.1 Extensive example

The example in `$CUBE_INSTALL_PREFIX/share/doc/cube/example/gui/plugin-demo` uses all major functions of PluginServices. It contains functions to handle

- settings
- further tab functions
- selections
- menus, context menus and toolbars
- global values to communicate with other plugins
- global preferences e.g. number formats

See also:

PluginServices

4.2 Problems loading plugins

If the plugin doesn't load, the most likely reason is an undefined reference. To check this, the plugin has be be linked with the cube libraries.

1. `cd $CUBE_SOURCE_DIR/build-backend/`
2. `export CHECK_PLUGIN=$PREFIX/plugins/libSimpleExamplePlugin.so`
3. `make check-plugin`

The last command generates the folloing output:

```
/opt/cube/lib64/plugins/libSimpleExamplePlugin.so passed test for undefined refer
      ences
```

If we remove the definition of the method `name()` in SimpleExample.cpp, the result shows an undefines reference. The plugin cannot be loaded.

```
/opt/cube/lib64/plugins/libSimpleExamplePlugin.so: undefined reference to 'Simple
      Example::name() const'
```