Table of contents

Introduction
Reasons for using QWindow over QWidget
Inheritance Chain of QWidget
Example integration code
1. Definitions
2. Fair Use Rights
3. License Grant
4. Restrictions
5. Representations, Warranties and Disclaimer
6. Limitation on Liability.
7. Termination
8. Miscellaneous

Introduction

Original forum link: http://www.ogre3d.org/forums/viewtopic.php?p=515314#p515314

The following document is an up-to-date method of integrating Ogre3D into the latest version of Qt (5.4 as of this writing).

Previous methods of integrating Ogre3D with Qt focused on using a QWidget and overriding the paint methods for drawing.

The method proposed in this document uses a QWindow as opposed to a QWidget. There are several reasons for using a QWindow over a QWidget, outlined in the next section.

Reasons for using QWindow over QWidget

- **Smaller memory and processing footprint**. Simply put, QWidgets use a lot of resources in comparison to QWindows and thus adds a lot of unnecessary overhead. Ogre3D doesn't require much from Qt besides a surface to draw upon and the input/window events from the underlying OS and windowing system that a QWindow provides. As a related sidenote, QWidget actually inherits from QWindow as shown in the below inheritance chain.
- **Easier integration**. Because of the simplicity of QWindow it is actually much easier to integrate Ogre3D into Qt using a QWindow versus using a QWidget. QWidget by default creates a surface to draw on which Ogre3D doesn't need to draw upon. Thus, as shown in earlier Qt-Ogre3D implementation documents you need to override Qt methods to prevent the default QWidget methods from "drawing over" Ogre3D. Using a QWindow this is unnecessary as with a QWindow no "surface" is automatically created; you can simply supply QWindow's native "Window ID" (whether it is a HWND on Windows or the equivalent representations on Linux/Mac OS) and Ogre3D takes care of the rest.
- **All standard Qt functionality can still be implemented**. If you still need to have a QWidget (for integration into a QMainWindow/QTabWidget/etc.) this can still be achieved. You can easily create a QWidget "container" for your QWindow and get all of the standard benefits of a QWidget. A standard use case for this would be needing to place an Ogre3D-integrated QWindow inside of a standard QMainWindow as a "central widget".

Inheritance Chain of QWidget

Example integration code

Note that the below implementation depends on you following the guides on setting up include/library/runtime binary directories. Since the setup for this is compiler and operating system dependent such details is beyond the scope of this document.

In summary, **consult a guide like this** for setting up directory paths, includes, etc. Note that you will need to do this for Qt as well so consult equivalent documentation for Qt Creator **like this**.

With the proper setup of the environment done, we need to create a header file to include the necessary Qt components as well as including Ogre3D:

```
#ifndef QTOGREWINDOW_H
#define QTOGREWINDOW_H

/*
Qt headers
*/
#include <QtWidgets/QApplication>
#include <QtGui/QKeyEvent>
#include <QtGui/QWindow>

/*
Ogre3D header
*/
#include <Ogre.h>

/*
Changed SdkCameraMan implementation to work with QKeyEvent, QMouseEvent, QWheelEvent
*/
#include "SdkQtCameraMan.h"

/*
With the headers included we now need to inherit from QWindow.
*/
class QTOgreWindow : public QWindow, public Ogre::FrameListener
{
        /*
        A QWindow still inherits from QObject and can have signals/slots; we need to
add the appropriate
        Q_OBJECT keyword so that Qt's intermediate compiler can do the necessary
wireup between our class
        and the rest of Qt.
        */
        Q_OBJECT

public:
        explicit QTOgreWindow(QWindow *parent = NULL);
        ~QTOgreWindow();

        /*
        We declare these methods virtual to allow for further inheritance.
```

```
        */
        virtual void render(QPainter *painter);
        virtual void render();
        virtual void initialize();
        virtual void createScene();
#if OGRE_VERSION >= ((2 << 16) | (0 << 8) | 0)
        virtual void createCompositor();
#endif

        void setAnimating(bool animating);

public slots:

        virtual void renderLater();
        virtual void renderNow();

        /*
        We use an event filter to be able to capture keyboard/mouse events. More on
this later.
        */
        virtual bool eventFilter(QObject *target, QEvent *event);

signals:
        /*
        Event for clicking on an entity.
        */
        void entitySelected(Ogre::Entity* entity);

protected:
        /*
        Ogre3D pointers added here. Useful to have the pointers here for use by the
window later.
        */
        Ogre::Root* m_ogreRoot;
        Ogre::RenderWindow* m_ogreWindow;
        Ogre::SceneManager* m_ogreSceneMgr;
        Ogre::Camera* m_ogreCamera;
        Ogre::ColourValue m_ogreBackground;
        OgreQtBites::SdkQtCameraMan* m_cameraMan;

        bool m_update_pending;
        bool m_animating;

        /*
        The below methods are what is actually fired when they keys on the keyboard
are hit.
        Similar events are fired when the mouse is pressed or other events occur.
        */
        virtual void keyPressEvent(QKeyEvent * ev);
```

```
        virtual void keyReleaseEvent(QKeyEvent * ev);
        virtual void mouseMoveEvent(QMouseEvent* e);
        virtual void wheelEvent(QWheelEvent* e);
        virtual void mousePressEvent(QMouseEvent* e);
        virtual void mouseReleaseEvent(QMouseEvent* e);
        virtual void exposeEvent(QExposeEvent *event);
        virtual bool event(QEvent *event);

        /*
        FrameListener method
        */
        virtual bool frameRenderingQueued(const Ogre::FrameEvent& evt);

        /*
        Write log messages to Ogre log
        */
        void log(Ogre::String msg);
        void log(QString msg);
};

#endif // QTOGREWINDOW_H
```

With the header prepared, it's now time to implement the class:

```
#include "QTOgreWindow.h"
#if OGRE_VERSION >= ((2 << 16) | (0 << 8) | 0)
#include <Compositor/OgreCompositorManager2.h>
#endif

/*
Note that we pass any supplied QWindow parent to the base QWindow class. This is
necessary should we
need to use our class within a container.
*/
QTOgreWindow::QTOgreWindow(QWindow *parent)
        : QWindow(parent)
        , m_update_pending(false)
        , m_animating(false)
        , m_ogreRoot(NULL)
        , m_ogreWindow(NULL)
        , m_ogreCamera(NULL)
        , m_cameraMan(NULL)
{
        setAnimating(true);
        installEventFilter(this);
        m_ogreBackground = Ogre::ColourValue(0.0f, 0.5f, 1.0f);
}
```

```
/*
Upon destruction of the QWindow object we destroy the Ogre3D scene.
*/
QTOgreWindow::~QTOgreWindow()
{
        if (m_cameraMan) delete m_cameraMan;
        delete m_ogreRoot;
}

/*
In case any drawing surface backing stores (QRasterWindow or QOpenGLWindow) of Qt
are supplied to this
class in any way we inform Qt that they will be unused.
*/
void QTOgreWindow::render(QPainter *painter)
{
        Q_UNUSED(painter);
}

/*
Our initialization function. Called by our renderNow() function once when the window
is first exposed.
*/
void QTOgreWindow::initialize()
{
        /*
        As shown Ogre3D is initialized normally; just like in other documentation.
        */
#ifdef _MSC_VER
        m_ogreRoot = new Ogre::Root(Ogre::String("plugins" OGRE_BUILD_SUFFIX
".cfg"));
#else
        m_ogreRoot = new Ogre::Root(Ogre::String("plugins.cfg"));
#endif
        Ogre::ConfigFile ogreConfig;

        /*

        Commended out for simplicity but should you need to initialize resources you
can do so normally.

        ogreConfig.load("resources/resource_configs/resources.cfg");

        Ogre::ConfigFile::SectionIterator seci = ogreConfig.getSectionIterator();
        Ogre::String secName, typeName, archName;
        while (seci.hasMoreElements())
        {
                secName = seci.peekNextKey();
                Ogre::ConfigFile::SettingsMultiMap *settings = seci.getNext();
```

```
                Ogre::ConfigFile::SettingsMultiMap::iterator i;
                for (i = settings->begin(); i != settings->end(); ++i)
                {
                        typeName = i->first;
                        archName = i->second;

Ogre::ResourceGroupManager::getSingleton().addResourceLocation(
                        archName, typeName, secName);
                }
        }

        */

        const Ogre::RenderSystemList& rsList = m_ogreRoot->getAvailableRenderers();
        Ogre::RenderSystem* rs = rsList[0];

        /*
        This list setup the search order for used render system.
        */
        Ogre::StringVector renderOrder;
#if defined(Q_OS_WIN)
        renderOrder.push_back("Direct3D9");
        renderOrder.push_back("Direct3D11");
#endif
        renderOrder.push_back("OpenGL");
        renderOrder.push_back("OpenGL 3+");
        for (Ogre::StringVector::iterator iter = renderOrder.begin(); iter !=
renderOrder.end(); iter++)
        {
                for (Ogre::RenderSystemList::const_iterator it = rsList.begin(); it
!= rsList.end(); it++)
                {
                        if ((*it)->getName().find(*iter) != Ogre::String::npos)
                        {
                                rs = *it;
                                break;
                        }
                }
                if (rs != NULL) break;
        }
        if (rs == NULL)
        {
                if (!m_ogreRoot->restoreConfig())
                {
                        if (!m_ogreRoot->showConfigDialog())
                                OGRE_EXCEPT(Ogre::Exception::ERR_INVALIDPARAMS,
                                        "Abort render system configuration",
                                        "QTOgreWindow::initialize");
                }
```

```
        }

        /*
        Setting size and VSync on windows will solve a lot of problems
        */
        QString dimensions = QString("%1 x
%2").arg(this->width()).arg(this->height());
        rs->setConfigOption("Video Mode", dimensions.toStdString());
        rs->setConfigOption("Full Screen", "No");
        rs->setConfigOption("VSync", "Yes");
        m_ogreRoot->setRenderSystem(rs);
        m_ogreRoot->initialise(false);

        Ogre::NameValuePairList parameters;
        /*
        Flag within the parameters set so that Ogre3D initializes an OpenGL context
on it's own.
        */
        if (rs->getName().find("GL") <= rs->getName().size())
                parameters["currentGLContext"] = Ogre::String("false");

        /*
        We need to supply the low level OS window handle to this QWindow so that
Ogre3D knows where to draw
        the scene. Below is a cross-platform method on how to do this.
        If you set both options (externalWindowHandle and parentWindowHandle) this
code will work with OpenGL
        and DirectX.
        */
#if defined(Q_OS_MAC) || defined(Q_OS_WIN)
        parameters["externalWindowHandle"] =
Ogre::StringConverter::toString((size_t)(this->winId()));
        parameters["parentWindowHandle"] = Ogre::StringConverter::toString((size_t)
(this->winId()));
#else
        parameters["externalWindowHandle"] =
Ogre::StringConverter::toString((unsigned long)(this->winId()));
        parameters["parentWindowHandle"] = Ogre::StringConverter::toString((unsigned
long)(this->winId()));
#endif

#if defined(Q_OS_MAC)
        parameters["macAPI"] = "cocoa";
        parameters["macAPICocoaUseNSView"] = "true";
#endif

        /*
        Note below that we supply the creation function for the Ogre3D window the
width and height
```

```
        from the current QWindow object using the "this" pointer.
        */
        m_ogreWindow = m_ogreRoot->createRenderWindow("QT Window",
                this->width(),
                this->height(),
                false,
                &parameters);
        m_ogreWindow->setVisible(true);

        /*
        The rest of the code in the initialization function is standard Ogre3D scene
code. Consult other
        tutorials for specifics.
        */
#if OGRE_VERSION >= ((2 << 16) | (0 << 8) | 0)
        const size_t numThreads = std::max<int>(1,
Ogre::PlatformInformation::getNumLogicalCores());
        Ogre::InstancingThreadedCullingMethod threadedCullingMethod =
Ogre::INSTANCING_CULLING_SINGLETHREAD;
        if (numThreads > 1)threadedCullingMethod =
Ogre::INSTANCING_CULLING_THREADED;
        m_ogreSceneMgr = m_ogreRoot->createSceneManager(Ogre::ST_GENERIC,
numThreads, threadedCullingMethod);
#else
        m_ogreSceneMgr = m_ogreRoot->createSceneManager(Ogre::ST_GENERIC);
#endif

        m_ogreCamera = m_ogreSceneMgr->createCamera("MainCamera");
        m_ogreCamera->setPosition(Ogre::Vector3(0.0f, 0.0f, 10.0f));
        m_ogreCamera->lookAt(Ogre::Vector3(0.0f, 0.0f, -300.0f));
        m_ogreCamera->setNearClipDistance(0.1f);
        m_ogreCamera->setFarClipDistance(200.0f);
        m_cameraMan = new OgreQtBites::SdkQtCameraMan(m_ogreCamera);   // create a
default camera controller

#if OGRE_VERSION >= ((2 << 16) | (0 << 8) | 0)
        createCompositor();
#else
        Ogre::Viewport* pViewPort = m_ogreWindow->addViewport(m_ogreCamera);
        pViewPort->setBackgroundColour(m_ogreBackground);
#endif

        m_ogreCamera->setAspectRatio(
                Ogre::Real(m_ogreWindow->getWidth()) /
Ogre::Real(m_ogreWindow->getHeight()));
        m_ogreCamera->setAutoAspectRatio(true);

        Ogre::TextureManager::getSingleton().setDefaultNumMipmaps(5);
        Ogre::ResourceGroupManager::getSingleton().initialiseAllResourceGroups();
```

```
        createScene();

        m_ogreRoot->addFrameListener(this);
}

void QTOgreWindow::createScene()
{
        /*
        Example scene
        Derive this class for your own purpose and overwite this function to have a
working Ogre widget with
        your own content.
        */
        m_ogreSceneMgr->setAmbientLight(Ogre::ColourValue(0.5f, 0.5f, 0.5f));

#if OGRE_VERSION >= ((2 << 16) | (0 << 8) | 0)
        Ogre::Entity* sphereMesh =
m_ogreSceneMgr->createEntity(Ogre::SceneManager::PT_SPHERE);
#else
        Ogre::Entity* sphereMesh = m_ogreSceneMgr->createEntity("mySphere",
Ogre::SceneManager::PT_SPHERE);
#endif

        Ogre::SceneNode* childSceneNode =
m_ogreSceneMgr->getRootSceneNode()->createChildSceneNode();

        childSceneNode->attachObject(sphereMesh);

        Ogre::MaterialPtr sphereMaterial =
Ogre::MaterialManager::getSingleton().create("SphereMaterial",
                Ogre::ResourceGroupManager::DEFAULT_RESOURCE_GROUP_NAME, true);

        sphereMaterial->getTechnique(0)->getPass(0)->setAmbient(0.1f, 0.1f, 0.1f);
        sphereMaterial->getTechnique(0)->getPass(0)->setDiffuse(0.2f, 0.2f, 0.2f,
1.0f);
        sphereMaterial->getTechnique(0)->getPass(0)->setSpecular(0.9f, 0.9f, 0.9f,
1.0f);
        //sphereMaterial->setAmbient(0.2f, 0.2f, 0.5f);
        //sphereMaterial->setSelfIllumination(0.2f, 0.2f, 0.1f);

        sphereMesh->setMaterialName("SphereMaterial");
        childSceneNode->setPosition(Ogre::Vector3(0.0f, 0.0f, 0.0f));
        childSceneNode->setScale(Ogre::Vector3(0.01f, 0.01f, 0.01f)); // Radius, in
theory.

#if OGRE_VERSION >= ((2 << 16) | (0 << 8) | 0)
        Ogre::SceneNode* pLightNode =
m_ogreSceneMgr->getRootSceneNode()->createChildSceneNode();
```

```
        Ogre::Light* light = m_ogreSceneMgr->createLight();
        pLightNode->attachObject(light);
        pLightNode->setPosition(20.0f, 80.0f, 50.0f);
#else
        Ogre::Light* light = m_ogreSceneMgr->createLight("MainLight");
        light->setPosition(20.0f, 80.0f, 50.0f);
#endif
}

#if OGRE_VERSION >= ((2 << 16) | (0 << 8) | 0)
void QTOgreWindow::createCompositor()
{
        /*
        Example compositor
        Derive this class for your own purpose and overwite this function to have a
working Ogre
        widget with your own compositor.
        */
        Ogre::CompositorManager2* compMan = m_ogreRoot->getCompositorManager2();
        const Ogre::String workspaceName = "default scene workspace";
        const Ogre::IdString workspaceNameHash = workspaceName;
        compMan->createBasicWorkspaceDef(workspaceName, m_ogreBackground);
        compMan->addWorkspace(m_ogreSceneMgr, m_ogreWindow, m_ogreCamera,
workspaceNameHash, true);
}
#endif

void QTOgreWindow::render()
{
        /*
        How we tied in the render function for OGre3D with QWindow's render
function. This is what gets call
        repeatedly. Note that we don't call this function directly; rather we use
the renderNow() function
        to call this method as we don't want to render the Ogre3D scene unless
everything is set up first.
        That is what renderNow() does.

        Theoretically you can have one function that does this check but from my
experience it seems better
        to keep things separate and keep the render function as simple as possible.
        */
        Ogre::WindowEventUtilities::messagePump();
        m_ogreRoot->renderOneFrame();
}

void QTOgreWindow::renderLater()
{
        /*
```

```
        This function forces QWindow to keep rendering. Omitting this causes the
renderNow() function to
        only get called when the window is resized, moved, etc. as opposed to all of
the time; which is
        generally what we need.
        */
        if (!m_update_pending)
        {
                m_update_pending = true;
                QApplication::postEvent(this, new QEvent(QEvent::UpdateRequest));
        }
}

bool QTOgreWindow::event(QEvent *event)
{
        /*
        QWindow's "message pump". The base method that handles all QWindow events.
As you will see there
        are other methods that actually process the keyboard/other events of Qt and
the underlying OS.

        Note that we call the renderNow() function which checks to see if everything
is initialized, etc.
        before calling the render() function.
        */

        switch (event->type())
        {
        case QEvent::UpdateRequest:
                m_update_pending = false;
                renderNow();
                return true;

        default:
                return QWindow::event(event);
        }
}

/*
Called after the QWindow is reopened or when the QWindow is first opened.
*/
void QTOgreWindow::exposeEvent(QExposeEvent *event)
{
        Q_UNUSED(event);

        if (isExposed())
                renderNow();
}
```

```
/*
The renderNow() function calls the initialize() function when needed and if the
QWindow is already
initialized and prepped calls the render() method.
*/
void QTOgreWindow::renderNow()
{
        if (!isExposed())
                return;

        if (m_ogreRoot == NULL)
        {
                initialize();
        }

        render();

        if (m_animating)
        renderLater();
}

/*
Our event filter; handles the resizing of the QWindow. When the size of the QWindow
changes note the
call to the Ogre3D window and camera. This keeps the Ogre3D scene looking correct.
*/
bool QTOgreWindow::eventFilter(QObject *target, QEvent *event)
{
        if (target == this)
        {
                if (event->type() == QEvent::Resize)
                {
                        if (isExposed() && m_ogreWindow != NULL)
                        {
                                m_ogreWindow->resize(this->width(), this->height());
                        }
                }
        }

        return false;
}

/*
How we handle keyboard and mouse events.
*/
void QTOgreWindow::keyPressEvent(QKeyEvent * ev)
{
        if(m_cameraMan)
                m_cameraMan->injectKeyDown(*ev);
```

```
}

void QTOgreWindow::keyReleaseEvent(QKeyEvent * ev)
{
        if(m_cameraMan)
                m_cameraMan->injectKeyUp(*ev);
}

void QTOgreWindow::mouseMoveEvent( QMouseEvent* e )
{
        static int lastX = e->x();
        static int lastY = e->y();
        int relX = e->x() - lastX;
        int relY = e->y() - lastY;
        lastX = e->x();
        lastY = e->y();

        if(m_cameraMan && (e->buttons() & Qt::LeftButton))
                m_cameraMan->injectMouseMove(relX, relY);
}

void QTOgreWindow::wheelEvent(QWheelEvent *e)
{
        if(m_cameraMan)
                m_cameraMan->injectWheelMove(*e);
}

void QTOgreWindow::mousePressEvent( QMouseEvent* e )
{
        if(m_cameraMan)
                m_cameraMan->injectMouseDown(*e);
}

void QTOgreWindow::mouseReleaseEvent( QMouseEvent* e )
{
        if(m_cameraMan)
                m_cameraMan->injectMouseUp(*e);

        QPoint pos = e->pos();
        Ogre::Ray mouseRay = m_ogreCamera->getCameraToViewportRay(
                (Ogre::Real)pos.x() / m_ogreWindow->getWidth(),
                (Ogre::Real)pos.y() / m_ogreWindow->getHeight());
        Ogre::RaySceneQuery* pSceneQuery = m_ogreSceneMgr->createRayQuery(mouseRay);
        pSceneQuery->setSortByDistance(true);
        Ogre::RaySceneQueryResult vResult = pSceneQuery->execute();
        for (size_t ui = 0; ui < vResult.size(); ui++)
        {
                if (vResult[ui].movable)
                {
```

```
                        if (vResult[ui].movable->getMovableType().compare("Entity")
== 0)
                        {
                                emit
entitySelected((Ogre::Entity*)vResult[ui].movable);
                        }
                }
        }
        m_ogreSceneMgr->destroyQuery(pSceneQuery);
}

/*
Function to keep track of when we should and shouldn't redraw the window; we
wouldn't want to do
rendering when the QWindow is minimized. This takes care of those scenarios.
*/
void QTOgreWindow::setAnimating(bool animating)
{
        m_animating = animating;

        if (animating)
                renderLater();
}

bool QTOgreWindow::frameRenderingQueued(const Ogre::FrameEvent& evt)
{
        m_cameraMan->frameRenderingQueued(evt);
        return true;
}

void QTOgreWindow::log(Ogre::String msg)
{
        if(Ogre::LogManager::getSingletonPtr() != NULL)
Ogre::LogManager::getSingletonPtr()->logMessage(msg);
}

void QTOgreWindow::log(QString msg)
{
        log(Ogre::String(msg.toStdString().c_str()));
}
```

The camera manager is a modified copy of the SDK version.

```
/*
 -----------------------------------------------------------------------------
 This source file is part of OGRE
 (Object-oriented Graphics Rendering Engine)
 For the latest info, see http://www.ogre3d.org/
```

```
 Copyright (c) 2000-2014 Torus Knot Software Ltd

 Permission is hereby granted, free of charge, to any person obtaining a copy
 of this software and associated documentation files (the "Software"), to deal
 in the Software without restriction, including without limitation the rights
 to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
 copies of the Software, and to permit persons to whom the Software is
 furnished to do so, subject to the following conditions:

 The above copyright notice and this permission notice shall be included in
 all copies or substantial portions of the Software.

 THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
 FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
 AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
 LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
 OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN
 THE SOFTWARE.
 -----------------------------------------------------------------------------
 */
// File modified to change OIS to Qt KeyEvents
#ifndef __SdkQtCameraMan_H__
#define __SdkQtCameraMan_H__

#include "OgreCamera.h"
#include "OgreSceneNode.h"
#include "OgreFrameListener.h"
#include <QKeyEvent>
#include <QMouseEvent>

// enum CameraStyle should be in other namespace than OgreBites::CameraStyle
namespace OgreQtBites
{
    enum CameraStyle   // enumerator values for different styles of camera movement
    {
        CS_FREELOOK,
        CS_ORBIT,
        CS_MANUAL
    };

    /*=============================================================================
    | Utility class for controlling the camera in samples.
    =============================================================================*/
    class SdkQtCameraMan
    {
    public:
        SdkQtCameraMan(Ogre::Camera* cam)
```

```
        : mCamera(0)
        , mTarget(0)
        , mOrbiting(false)
        , mZooming(false)
        , mTopSpeed(150)
        , mVelocity(Ogre::Vector3::ZERO)
        , mGoingForward(false)
        , mGoingBack(false)
        , mGoingLeft(false)
        , mGoingRight(false)
        , mGoingUp(false)
        , mGoingDown(false)
        , mFastMove(false)
        {

            setCamera(cam);
            setStyle(CS_FREELOOK);
        }

        virtual ~SdkQtCameraMan() {}


/*-----------------------------------------------------------------------------
        | Swaps the camera on our camera man for another camera.

-----------------------------------------------------------------------------*/
        virtual void setCamera(Ogre::Camera* cam)
        {
            mCamera = cam;
        }

        virtual Ogre::Camera* getCamera()
        {
            return mCamera;
        }


/*-----------------------------------------------------------------------------
        | Sets the target we will revolve around. Only applies for orbit style.

-----------------------------------------------------------------------------*/
        virtual void setTarget(Ogre::SceneNode* target)
        {
            if (target != mTarget)
            {
                mTarget = target;
                if(target)
                {
                    setYawPitchDist(Ogre::Degree(0), Ogre::Degree(15), 150);
```

```
                    mCamera->setAutoTracking(true, mTarget);
                }
                else
                {
                    mCamera->setAutoTracking(false);
                }

            }


        }

        virtual Ogre::SceneNode* getTarget()
        {
            return mTarget;
        }


/*-----------------------------------------------------------------------------
        | Sets the spatial offset from the target. Only applies for orbit style.

-----------------------------------------------------------------------------*/
        virtual void setYawPitchDist(Ogre::Radian yaw, Ogre::Radian pitch,
Ogre::Real dist)
        {
            mCamera->setPosition(mTarget->_getDerivedPosition());
            mCamera->setOrientation(mTarget->_getDerivedOrientation());
            mCamera->yaw(yaw);
            mCamera->pitch(-pitch);
            mCamera->moveRelative(Ogre::Vector3(0, 0, dist));
        }


/*-----------------------------------------------------------------------------
        | Sets the camera's top speed. Only applies for free-look style.

-----------------------------------------------------------------------------*/
        virtual void setTopSpeed(Ogre::Real topSpeed)
        {
            mTopSpeed = topSpeed;
        }

        virtual Ogre::Real getTopSpeed()
        {
            return mTopSpeed;
        }


/*-----------------------------------------------------------------------------
```

```
        | Sets the movement style of our camera man.

-------------------------------------------------------------------------------*/
        virtual void setStyle(CameraStyle style)
        {
            if (mStyle != CS_ORBIT && style == CS_ORBIT)
            {
                setTarget(mTarget ? mTarget :
mCamera->getSceneManager()->getRootSceneNode());
                mCamera->setFixedYawAxis(true);
                manualStop();
                setYawPitchDist(Ogre::Degree(0), Ogre::Degree(15), 150);
            }
            else if (mStyle != CS_FREELOOK && style == CS_FREELOOK)
            {
                mCamera->setAutoTracking(false);
                mCamera->setFixedYawAxis(true);
            }
            else if (mStyle != CS_MANUAL && style == CS_MANUAL)
            {
                mCamera->setAutoTracking(false);
                manualStop();
            }
            mStyle = style;

        }

        virtual CameraStyle getStyle()
        {
            return mStyle;
        }


/*-------------------------------------------------------------------------------
        | Manually stops the camera when in free-look mode.

-------------------------------------------------------------------------------*/
        virtual void manualStop()
        {
            if (mStyle == CS_FREELOOK)
            {
                mGoingForward = false;
                mGoingBack = false;
                mGoingLeft = false;
                mGoingRight = false;
                mGoingUp = false;
                mGoingDown = false;
                mVelocity = Ogre::Vector3::ZERO;
            }
```

```
        }

        virtual bool frameRenderingQueued(const Ogre::FrameEvent& evt)
        {
            if (mStyle == CS_FREELOOK)
            {
                // build our acceleration vector based on keyboard input composite
                Ogre::Vector3 accel = Ogre::Vector3::ZERO;
                if (mGoingForward) accel += mCamera->getDirection();
                if (mGoingBack) accel -= mCamera->getDirection();
                if (mGoingRight) accel += mCamera->getRight();
                if (mGoingLeft) accel -= mCamera->getRight();
                if (mGoingUp) accel += mCamera->getUp();
                if (mGoingDown) accel -= mCamera->getUp();

                // if accelerating, try to reach top speed in a certain time
                Ogre::Real topSpeed = mFastMove ? mTopSpeed * 20 : mTopSpeed;
                if (accel.squaredLength() != 0)
                {
                    accel.normalise();
                    mVelocity += accel * topSpeed * evt.timeSinceLastFrame * 10;
                }
                // if not accelerating, try to stop in a certain time
                else mVelocity -= mVelocity * evt.timeSinceLastFrame * 10;

                Ogre::Real tooSmall = std::numeric_limits<Ogre::Real>::epsilon();

                // keep camera velocity below top speed and above epsilon
                if (mVelocity.squaredLength() > topSpeed * topSpeed)
                {
                    mVelocity.normalise();
                    mVelocity *= topSpeed;
                }
                else if (mVelocity.squaredLength() < tooSmall * tooSmall)
                    mVelocity = Ogre::Vector3::ZERO;

                if (mVelocity != Ogre::Vector3::ZERO) mCamera->move(mVelocity *
evt.timeSinceLastFrame);
            }

            return true;
        }


/*-----------------------------------------------------------------------------
        | Processes key presses for free-look style movement.

-----------------------------------------------------------------------------*/
        virtual void injectKeyDown(const QKeyEvent& evt)
```

```
        {
            if (mStyle == CS_FREELOOK)
            {
                if (evt.key() == Qt::Key_W || evt.key() == Qt::Key_Up) mGoingForward
= true;
                else if (evt.key() == Qt::Key_S || evt.key() == Qt::Key_Down)
mGoingBack = true;
                else if (evt.key() == Qt::Key_A || evt.key() == Qt::Key_Left)
mGoingLeft = true;
                else if (evt.key() == Qt::Key_D || evt.key() == Qt::Key_Right)
mGoingRight = true;
                else if (evt.key() == Qt::Key_PageUp) mGoingUp = true;
                else if (evt.key() == Qt::Key_PageDown) mGoingDown = true;
                else if (evt.key() == Qt::Key_Shift) mFastMove = true;
            }
        }


/*-----------------------------------------------------------------------------
        | Processes key releases for free-look style movement.

-----------------------------------------------------------------------------*/
        virtual void injectKeyUp(const QKeyEvent& evt)
        {
            if (mStyle == CS_FREELOOK)
            {
                if (evt.key() == Qt::Key_W || evt.key() == Qt::Key_Up) mGoingForward
= false;
                else if (evt.key() == Qt::Key_S || evt.key() == Qt::Key_Down)
mGoingBack = false;
                else if (evt.key() == Qt::Key_A || evt.key() == Qt::Key_Left)
mGoingLeft = false;
                else if (evt.key() == Qt::Key_D || evt.key() == Qt::Key_Right)
mGoingRight = false;
                else if (evt.key() == Qt::Key_PageUp) mGoingUp = false;
                else if (evt.key() == Qt::Key_PageDown) mGoingDown = false;
                else if (evt.key() == Qt::Key_Shift) mFastMove = false;
            }
        }


/*-----------------------------------------------------------------------------
        | Processes mouse movement differently for each style.

-----------------------------------------------------------------------------*/
        virtual void injectMouseMove(int relX, int relY)
        {
//            static int lastX = evt.x();
//            static int lastY = evt.y();
```

```
//              int relX = evt.x() - lastX;
//              int relY = evt.y() - lastY;
//              lastX = evt.x();
//              lastY = evt.y();
            if (mStyle == CS_ORBIT)
            {
                Ogre::Real dist = (mCamera->getPosition() -
mTarget->_getDerivedPosition()).length();

                if (mOrbiting)   // yaw around the target, and pitch locally
                {
                    mCamera->setPosition(mTarget->_getDerivedPosition());

                    mCamera->yaw(Ogre::Degree(-relX * 0.025f));
                    mCamera->pitch(Ogre::Degree(-relY * 0.025f));

                    mCamera->moveRelative(Ogre::Vector3(0, 0, dist));

                    // don't let the camera go over the top or around the bottom of
the target
                }
                else if (mZooming)  // move the camera toward or away from the
target
                {
                    // the further the camera is, the faster it moves
                    mCamera->moveRelative(Ogre::Vector3(0, 0, relY * 0.004f *
dist));
                }
            }
            else if (mStyle == CS_FREELOOK)
            {
                mCamera->yaw(Ogre::Degree(-relX * 0.15f));
                mCamera->pitch(Ogre::Degree(-relY * 0.15f));
            }
        }


/*-----------------------------------------------------------------------------
        | Processes mouse movement differently for each style.

-----------------------------------------------------------------------------*/
        virtual void injectWheelMove(const QWheelEvent& evt)
        {
            int relZ = evt.delta();
            if (mStyle == CS_ORBIT)
            {
                Ogre::Real dist = (mCamera->getPosition() -
mTarget->_getDerivedPosition()).length();
```

```
                if (relZ != 0)  // move the camera toward or away from the target
                {
                    // the further the camera is, the faster it moves
                    mCamera->moveRelative(Ogre::Vector3(0, 0, -relZ * 0.0008f *
dist));
                }
            }
        }


/*-----------------------------------------------------------------------------
        | Processes mouse presses. Only applies for orbit style.
        | Left button is for orbiting, and right button is for zooming.

-----------------------------------------------------------------------------*/
        virtual void injectMouseDown(const QMouseEvent& evt)
        {
            if (mStyle == CS_ORBIT)
            {
                if (evt.buttons() & Qt::LeftButton) mOrbiting = true;
                else if (evt.buttons() & Qt::RightButton) mZooming = true;
            }
        }


/*-----------------------------------------------------------------------------
        | Processes mouse releases. Only applies for orbit style.
        | Left button is for orbiting, and right button is for zooming.

-----------------------------------------------------------------------------*/
        virtual void injectMouseUp(const QMouseEvent& evt)
        {
            if (mStyle == CS_ORBIT)
            {
                if (evt.buttons() & Qt::LeftButton) mOrbiting = false;
                else if (evt.buttons() & Qt::RightButton) mZooming = false;
            }
        }

    protected:

        Ogre::Camera* mCamera;
        CameraStyle mStyle;
        Ogre::SceneNode* mTarget;
        bool mOrbiting;
        bool mZooming;
        Ogre::Real mTopSpeed;
        Ogre::Vector3 mVelocity;
        bool mGoingForward;
```

```
        bool mGoingBack;
        bool mGoingLeft;
        bool mGoingRight;
        bool mGoingUp;
        bool mGoingDown;
        bool mFastMove;
    };
}

#endif
```

With our class set up, here is how we would use it as a standalone window:

```
/*

Place the include along with your other includes where you will be creating and/or
calling the above QTOgreWindow:

*/


#include "QTOgreWindow.h"

/*

In the method you are creating/calling a QTOgreWindow:

*/

QTOgreWindow* ogreWindow = new QTOgreWindow();
ogreWindow->show();
```

If you wish to integrate this QWindow inside of a QWidget for use with another section/QWidget of Qt, you do this:

```
/*

Place the include along with your other includes where you will be creating and/or
calling the above QTOgreWindow:

*/


#include "QTOgreWindow.h"

/*

In the method you are creating/calling a QTOgreWindow:
```

```
*/

QTOgreWindow* ogreWindow = new QTOgreWindow();
QWidget* renderingContainer = QWidget::createWindowContainer(ogreWindow);

/*

As an example, the below method places the QTOgreWindow we just created inside of a
QTabWidget as a tab.

*/

mainTabs->addTab(renderingContainer, tr("New Ogre Window"));
```

And that's it!

Comments and edits welcome.

1. Definitions

- "Collective Work" means a work, such as a periodical issue, anthology or encyclopedia, in which the Work in its entirety in unmodified form, along with a number of other contributions, constituting separate and independent works in themselves, are assembled into a collective whole. A work that constitutes a Collective Work will not be considered a Derivative Work (as defined below) for the purposes of this License.
- "Derivative Work" means a work based upon the Work or upon the Work and other pre-existing works, such as a translation, musical arrangement, dramatization, fictionalization, motion picture version, sound recording, art reproduction, abridgment, condensation, or any other form in which the Work may be recast, transformed, or adapted, except that a work that constitutes a Collective Work will not be considered a Derivative Work for the purpose of this License. For the avoidance of doubt, where the Work is a musical composition or sound recording, the synchronization of the Work in timed-relation with a moving image ("synching") will be considered a Derivative Work for the purpose of this License.
- "Licensor" means the individual or entity that offers the Work under the terms of this License.
- "Original Author" means the individual or entity who created the Work.
- "Work" means the copyrightable work of authorship offered under the terms of this License.
- "You" means an individual or entity exercising rights under this License who has not previously violated the terms of this License with respect to the Work, or who has received express permission from the Licensor to exercise rights under this License despite a previous violation.
- "License Elements" means the following high-level license attributes as selected by Licensor and indicated in the title

of this License: Attribution, ShareAlike.

2. Fair Use Rights

Nothing in this license is intended to reduce, limit, or restrict any rights arising from fair use, first sale or other limitations on the exclusive rights of the copyright owner under copyright law or other applicable laws.

3. License Grant

Subject to the terms and conditions of this License, Licensor hereby grants You a worldwide, royalty-free, non-exclusive, perpetual (for the duration of the applicable copyright) license to exercise the rights in the Work as stated below:

- to reproduce the Work, to incorporate the Work into one or more Collective Works, and to reproduce the Work as incorporated in the Collective Works;
- to create and reproduce Derivative Works;
- to distribute copies or phonorecords of, display publicly, perform publicly, and perform publicly by means of a digital audio transmission the Work including as incorporated in Collective Works;
- to distribute copies or phonorecords of, display publicly, perform publicly, and perform publicly by means of a digital audio transmission Derivative Works.
- For the avoidance of doubt, where the work is a musical composition:
    - Performance Royalties Under Blanket Licenses. Licensor waives the exclusive right to collect, whether individually or via a performance rights society (e.g. ASCAP, BMI, SESAC), royalties for the public performance or public digital performance (e.g. webcast) of the Work.
    - Mechanical Rights and Statutory Royalties. Licensor waives the exclusive right to collect, whether individually or via a music rights society or designated agent (e.g. Harry Fox Agency), royalties for any phonorecord You create from the Work ("cover version") and distribute, subject to the compulsory license created by 17 USC Section 115 of the US Copyright Act (or the equivalent in other jurisdictions).
    - Webcasting Rights and Statutory Royalties. For the avoidance of doubt, where the Work is a sound recording, Licensor waives the exclusive right to collect, whether individually or via a performance-rights society (e.g. SoundExchange), royalties for the public digital performance (e.g. webcast) of the Work, subject to the compulsory license created by 17 USC Section 114 of the US Copyright Act (or the equivalent in other jurisdictions).

The above rights may be exercised in all media and formats whether now known or hereafter devised. The above rights include the right to make such modifications as are technically necessary to exercise the rights in other media and formats. All rights not expressly granted by Licensor are hereby reserved.

4. Restrictions

The license granted in Section 3 above is expressly made subject to and limited by the following restrictions:

- You may distribute, publicly display, publicly perform, or publicly digitally perform the Work only under the terms of this License, and You must include a copy of, or the Uniform Resource Identifier for, this License with every copy or phonorecord of the Work You distribute, publicly display, publicly perform, or publicly digitally perform. You may not offer or impose any terms on the Work that alter or restrict the terms of this License or the recipients' exercise of the rights granted hereunder. You may not sublicense the Work. You must keep intact all notices that refer to this License and to the disclaimer of warranties. You may not distribute, publicly display, publicly perform, or publicly digitally perform the Work with any technological measures that control access or use of the Work in a manner inconsistent with the terms of this License Agreement. The above applies to the Work as incorporated in a Collective Work, but this does not require the Collective Work apart from the Work itself to be made subject to the terms of this License. If

You create a Collective Work, upon notice from any Licensor You must, to the extent practicable, remove from the Collective Work any credit as required by clause 4(c), as requested. If You create a Derivative Work, upon notice from any Licensor You must, to the extent practicable, remove from the Derivative Work any credit as required by clause 4(c), as requested.

- You may distribute, publicly display, publicly perform, or publicly digitally perform a Derivative Work only under the terms of this License, a later version of this License with the same License Elements as this License, or a Creative Commons iCommons license that contains the same License Elements as this License (e.g. Attribution-ShareAlike 2.5 Japan). You must include a copy of, or the Uniform Resource Identifier for, this License or other license specified in the previous sentence with every copy or phonorecord of each Derivative Work You distribute, publicly display, publicly perform, or publicly digitally perform. You may not offer or impose any terms on the Derivative Works that alter or restrict the terms of this License or the recipients' exercise of the rights granted hereunder, and You must keep intact all notices that refer to this License and to the disclaimer of warranties. You may not distribute, publicly display, publicly perform, or publicly digitally perform the Derivative Work with any technological measures that control access or use of the Work in a manner inconsistent with the terms of this License Agreement. The above applies to the Derivative Work as incorporated in a Collective Work, but this does not require the Collective Work apart from the Derivative Work itself to be made subject to the terms of this License.
- If you distribute, publicly display, publicly perform, or publicly digitally perform the Work or any Derivative Works or Collective Works, You must keep intact all copyright notices for the Work and provide, reasonable to the medium or means You are utilizing: (i) the name of the Original Author (or pseudonym, if applicable) if supplied, and/or (ii) if the Original Author and/or Licensor designate another party or parties (e.g. a sponsor institute, publishing entity, journal) for attribution in Licensor's copyright notice, terms of service or by other reasonable means, the name of such party or parties; the title of the Work if supplied; to the extent reasonably practicable, the Uniform Resource Identifier, if any, that Licensor specifies to be associated with the Work, unless such URI does not refer to the copyright notice or licensing information for the Work; and in the case of a Derivative Work, a credit identifying the use of the Work in the Derivative Work (e.g., "French translation of the Work by Original Author," or "Screenplay based on original Work by Original Author"). Such credit may be implemented in any reasonable manner; provided, however, that in the case of a Derivative Work or Collective Work, at a minimum such credit will appear where any other comparable authorship credit appears and in a manner at least as prominent as such other comparable authorship credit.

5. Representations, Warranties and Disclaimer

UNLESS OTHERWISE AGREED TO BY THE PARTIES IN WRITING, LICENSOR OFFERS THE WORK AS-IS AND MAKES NO REPRESENTATIONS OR WARRANTIES OF ANY KIND CONCERNING THE MATERIALS, EXPRESS, IMPLIED, STATUTORY OR OTHERWISE, INCLUDING, WITHOUT LIMITATION, WARRANTIES OF TITLE, MERCHANTIBILITY, FITNESS FOR A PARTICULAR PURPOSE, NONINFRINGEMENT, OR THE ABSENCE OF LATENT OR OTHER DEFECTS, ACCURACY, OR THE PRESENCE OF ABSENCE OF ERRORS, WHETHER OR NOT DISCOVERABLE. SOME JURISDICTIONS DO NOT ALLOW THE EXCLUSION OF IMPLIED WARRANTIES, SO SUCH EXCLUSION MAY NOT APPLY TO YOU.

6. Limitation on Liability.

EXCEPT TO THE EXTENT REQUIRED BY APPLICABLE LAW, IN NO EVENT WILL LICENSOR BE LIABLE TO YOU ON ANY LEGAL THEORY FOR ANY SPECIAL, INCIDENTAL, CONSEQUENTIAL, PUNITIVE OR EXEMPLARY DAMAGES ARISING OUT OF THIS LICENSE OR THE USE OF THE WORK, EVEN IF LICENSOR HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

7. Termination

- This License and the rights granted hereunder will terminate automatically upon any breach by You of the terms of this License. Individuals or entities who have received Derivative Works or Collective Works from You under this

License, however, will not have their licenses terminated provided such individuals or entities remain in full compliance with those licenses. Sections 1, 2, 5, 6, 7, and 8 will survive any termination of this License.

- Subject to the above terms and conditions, the license granted here is perpetual (for the duration of the applicable copyright in the Work). Notwithstanding the above, Licensor reserves the right to release the Work under different license terms or to stop distributing the Work at any time; provided, however that any such election will not serve to withdraw this License (or any other license that has been, or is required to be, granted under the terms of this License), and this License will continue in full force and effect unless terminated as stated above.

8. Miscellaneous

- Each time You distribute or publicly digitally perform the Work or a Collective Work, the Licensor offers to the recipient a license to the Work on the same terms and conditions as the license granted to You under this License.
- Each time You distribute or publicly digitally perform a Derivative Work, Licensor offers to the recipient a license to the original Work on the same terms and conditions as the license granted to You under this License.
- If any provision of this License is invalid or unenforceable under applicable law, it shall not affect the validity or enforceability of the remainder of the terms of this License, and without further action by the parties to this agreement, such provision shall be reformed to the minimum extent necessary to make such provision valid and enforceable.
- No term or provision of this License shall be deemed waived and no breach consented to unless such waiver or consent shall be in writing and signed by the party to be charged with such waiver or consent.
- This License constitutes the entire agreement between the parties with respect to the Work licensed here. There are no understandings, agreements or representations with respect to the Work not specified here. Licensor shall not be bound by any additional provisions that may appear in any communication from You. This License may not be modified without the mutual written agreement of the Licensor and You.

The original document is available at **http://www.ogre3d.org/tikiwiki/tiki-index.php?page=Integrating+Ogre+into+QT5**