```
#include "qtgstreamerdriver.h"

QtGStreamerDriver::QtGStreamerDriver(QWidget *parent): QGst::Ui::VideoWidget(parent)
{
    connect(&mPositionTimer,SIGNAL(timeout()),this,SIGNAL(positionChanged()));
    setVolume(3);
}

QtGStreamerDriver::~QtGStreamerDriver()
{
    if(mPipelinePtr)
    {
        mPipelinePtr->setState(QGst::StateNull);

        //http://gstreamer.freedesktop.org/data/doc/gstreamer/head/qt-gstreamer/html/
classQGst_1_1Ui_1_1VideoWidget.html
        //void QGst::Ui::VideoWidget::stopPipelineWatch    ()
        //Stops watching a pipeline and also detaches the sink that was discovered in the
pipeline, if any.
        stopPipelineWatch();
    }
}

//Private Member Function / Methods

void QtGStreamerDriver::onBusMessage(QGst::MessagePtr vMessage)
{
    switch (vMessage->type())
    {
    case (QGst::MessageEos):
    {//End of stream. We reached the end of the file.
        stop();
        break;
    }
    case (QGst::MessageError): //Some error occurred.
    {
        qCritical() << vMessage.staticCast<QGst::ErrorMessage>()->error();
        stop();
        break;
    }
    case (QGst::MessageStateChanged): //The element in message->source() has changed
state
    {
        if (vMessage->source() == mPipelinePtr)
        {
            handlePipelineStateChange(vMessage.staticCast<QGst::StateChangedMessage>());
        }
        break;
    }
    default:
        break;
    }
}

void QtGStreamerDriver::handlePipelineStateChange(QGst::StateChangedMessagePtr
vStateChangedMessage)
{
    switch (vStateChangedMessage->newState())
    {
    case (QGst::StatePlaying):
    {
        //start the timer when the pipeline starts playing
        mPositionTimer.start(100);
        break;
    }
    case (QGst::StatePaused):
    {
        //stop the timer when the pipeline pauses
        if(vStateChangedMessage->oldState() == QGst::StatePlaying)
```

```
        {
            mPositionTimer.stop();
        }
        break;
    }
    default:
        break;
    }

    emit(stateChanged());
}

// Public Member Function / methods

//Mutators

void QtGStreamerDriver::setPath(QString vPath)
{
        QString realUri = vPath;

        //if uri is not a real uri, assume it is a file path
        if (realUri.indexOf("://") < 0) {
            realUri = QUrl::fromLocalFile(realUri).toEncoded();
        }

    if (!mPipelinePtr)
    {
        mPipelinePtr =
QGst::ElementFactory::make("playbin2").dynamicCast<QGst::Pipeline>();

        if (mPipelinePtr) {
            //void QGst::Ui::VideoWidget::watchPipeline    (   const PipelinePtr &
pipeline    )
            //let the video widget watch the pipeline for new video sinks
            watchPipeline(mPipelinePtr);

            //watch the bus for messages
            QGst::BusPtr vBus = mPipelinePtr->bus();
            vBus->addSignalWatch();
            QGlib::connect(vBus, "message", this, &QtGStreamerDriver::onBusMessage);
        }
        else
        {
            qCritical() << "Failed to create the pipeline";
        }
    }

    if (mPipelinePtr) {
        mPipelinePtr->setProperty("uri", realUri);
    }
}


void QtGStreamerDriver::setPosition(QTime vPosition)
{
    QGst::SeekEventPtr vEvent = QGst::SeekEvent::create(
                1.0, QGst::FormatTime, QGst::SeekFlagFlush,
                QGst::SeekTypeSet, QGst::ClockTime::fromTime(vPosition),
                QGst::SeekTypeNone, QGst::ClockTime::None
                );

    mPipelinePtr->sendEvent(vEvent);
}
// Accessors
QTime QtGStreamerDriver::getPosition()
{
    if (mPipelinePtr) {
        //here we query the pipeline about its position
        //and we request that the result is returned in time format
```

```
        QGst::PositionQueryPtr vQuery = QGst::PositionQuery::create(QGst::FormatTime);
        mPipelinePtr->query(vQuery);
        return QGst::ClockTime(vQuery->position()).toTime();
    } else {
        return QTime();
    }
}

int QtGStreamerDriver::getVolume()
{
    if (mPipelinePtr) {
        QGst::StreamVolumePtr vStreamVolumePtr =
mPipelinePtr.dynamicCast<QGst::StreamVolume>();

        if (vStreamVolumePtr) {
            return vStreamVolumePtr->volume(QGst::StreamVolumeFormatCubic) * 10;
        }
    }

    return 0;
}



QTime QtGStreamerDriver::getDuration()
{
    if (mPipelinePtr) {
        //here we query the pipeline about the content's duration
        //and we request that the result is returned in time format
        QGst::DurationQueryPtr vQuery = QGst::DurationQuery::create(QGst::FormatTime);
        mPipelinePtr->query(vQuery);
        return QGst::ClockTime(vQuery->duration()).toTime();
    } else {
        return QTime();
    }
}


QGst::State QtGStreamerDriver::getState()
{
    if(mPipelinePtr)
    {
        return mPipelinePtr->currentState();
    }
    else
    {
        return QGst::StateNull;
    }
}

//SLOTS

void QtGStreamerDriver::play()
{
    if (mPipelinePtr) {
        mPipelinePtr->setState(QGst::StatePlaying);
    }
}


void QtGStreamerDriver::pause()
{
    if (mPipelinePtr) {
        mPipelinePtr->setState(QGst::StatePaused);
    }
}


void QtGStreamerDriver::stop()
```

```
{
    if (mPipelinePtr) {
        mPipelinePtr->setState(QGst::StateNull);

        //once the pipeline stops, the bus is flushed so we will
        //not receive any StateChangedMessage about this.
        //so, to inform the ui, we have to emit this signal manually.
        emit(stateChanged());
    }
}


void QtGStreamerDriver::setVolume(int vVolume)
{
    if (mPipelinePtr) {
        QGst::StreamVolumePtr vStreamVolumePtr =
mPipelinePtr.dynamicCast<QGst::StreamVolume>();
        if(vStreamVolumePtr) {
            vStreamVolumePtr->setVolume((double)vVolume / 10,
QGst::StreamVolumeFormatCubic);
        }
    }
}
```