```
#-------------------------------------------------
#
# Project created by QtCreator 2013-11-14T02:19:20
#
#-------------------------------------------------

QT       += core gui

greaterThan(QT_MAJOR_VERSION, 4): QT += widgets

TARGET = MediaTake
TEMPLATE = app


SOURCES += main.cpp\
        mainwindow.cpp \
    selectfiledialog.cpp \
    databaseoperations.cpp \
    qtgstreamerdriver.cpp \
    videosink.cpp \
    databaseoperationsaudio.cpp \
    databaseoperationsvideo.cpp \
    filefinderaudio.cpp \
    filefindervideo.cpp \
    librarymanageraudio.cpp \
    librarymanagervideo.cpp

HEADERS  += mainwindow.h \
    selectfiledialog.h \
    databaseoperations.h \
    qtgstreamerdriver.h \
    videosink.h \
    databaseoperationsaudio.h \
    databaseoperationsvideo.h \
    filefinderaudio.h \
    filefindervideo.h \
    librarymanageraudio.h \
    librarymanagervideo.h

FORMS    += mainwindow.ui \
    selectfiledialog.ui \
    videosink.ui


unix{
LIBS += -lmongoclient\
        -lboost_thread-mt\
        -lboost_filesystem\
        -lboost_program_options\
        -lboost_system\


QMAKE_CXXFLAGS += -std=c++11 -pthread -Wno-deprecated #-fpermissive -pedantic
LIBS += -pthread

LIBS += -ltag\
#        -lQt5GStreamer-0.10 -lQt5GLib-2.0 -lQt5Core -lQt5GStreamerUi-0.10 -
lQt5GStreamerUtils-0.10\
#        -L/usr/local/Qt5.0.1/5.0.1/gcc_64/lib\
        -L/usr/local/lib

INCLUDEPATH+= /usr/local/include

#CONFIG += link_pkgconfig
#PKGCONFIG += gstreamer-1.0

#CONFIG += link_pkgconfig
#PKGCONFIG += \
#Qt5GLib-2.0\  #- the libraries needed to use QtGLib
#Qt5GStreamer-0.10\ #- the libraries needed to use QtGStreamer
#Qt5GStreamerUi-0.10\   #- the libraries needed to use QtGStreamerUi
#Qt5GStreamerUtils-0.10\  #-the libraries needed to use QtGStreamerUtils

CONFIG += link_pkgconfig
PKGCONFIG += \
QtGLib-2.0\  #- the libraries needed to use QtGLib
QtGStreamer-0.10\ #- the libraries needed to use QtGStreamer
QtGStreamerUi-0.10\   #- the libraries needed to use QtGStreamerUi
QtGStreamerUtils-0.10\  #-the libraries needed to use QtGStreamerUtils

}

RESOURCES += \
    UiResources.qrc
```

```
#ifndef DATABASEOPERATIONS_H
#define DATABASEOPERATIONS_H

#include <cstdio>
#include <cstdlib>
#include <vector>
#include <QtCore>
#include <QMessageBox>

#include <taglib/tag.h>
#include <taglib/fileref.h>
#include <taglib/tpropertymap.h>

//#include <taglib/tbytevector.h>
//#include <taglib/mpegfile.h>

//#include <taglib/id3v2tag.h>
//#include <taglib/id3v2frame.h>
//#include <taglib/id3v2header.h>

//#include <taglib/id3v1tag.h>
//#include <taglib/apetag.h>

#include <mongo/client/dbclient.h>

using std::vector;
class DatabaseOperations : public QThread
{
    Q_OBJECT

public:

    void getSetting();
    virtual void getSource()=0;

protected:
    mongo::DBClientConnection mDBConnection;
    mongo::auto_ptr<mongo::DBClientCursor> mCursor;


    mongo::BSONObjBuilder *mBSONObjBuilder;
    mongo::BSONObj mBSONObj;

    TagLib::FileRef *mFile;
    TagLib::Tag *mTag;
    TagLib::AudioProperties *mAudioProp;
    //    TagLib::ID3v2::Tag mID3Tag;

    vector < vector < QString > > mFileProp;

    QMutex mMutex;

    explicit DatabaseOperations(QObject *parent = 0);
    virtual ~DatabaseOperations();
    virtual void removeFromDB(QString)=0;
    virtual void updateDB(QString,QFileInfo,QFileInfoList)=0;
    virtual void updateTreeView()=0;
    virtual void updateSource(QString)=0;

    virtual void initiator()=0;
    virtual void destroyer()=0;
    virtual void manager()=0;
    void setSetting();

signals:

    virtual void updateTreeWidgetLibraryDisplay(vector < vector < QString> >)=0;
    virtual void updatePath(QString)=0;
public slots:


};

#endif // DATABASEOPERATIONS_H
```

```
#ifndef DATABASEOPERATIONSAUDIO_H
#define DATABASEOPERATIONSAUDIO_H

#include "databaseoperations.h"
class DatabaseOperationsAudio : public DatabaseOperations
{
    Q_OBJECT
public:

    void getSource();
protected:
    explicit DatabaseOperationsAudio(QObject *parent = 0);

    void removeFromDB(QString);
    void updateDB(QString,QFileInfo,QFileInfoList);
    void updateTreeView();
    void updateSource(QString);
    virtual void initiator()=0;
    virtual void destroyer()=0;
    virtual void manager()=0;

signals:
    void updateTreeWidgetLibraryDisplay(vector < vector < QString> >);
    void updatePath(QString);


public slots:

};

#endif // DATABASEOPERATIONSAUDIO_H
```

```cpp
#ifndef DATABASEOPERATIONSVIDEO_H
#define DATABASEOPERATIONSVIDEO_H

#include "databaseoperations.h"

class DatabaseOperationsVideo : public DatabaseOperations
{
    Q_OBJECT

public:

    void getSource();
protected:
    explicit DatabaseOperationsVideo(QObject *parent = 0);

    void removeFromDB(QString);
    void updateDB(QString,QFileInfo,QFileInfoList);
    void updateTreeView();
    void updateSource(QString);
    void setSetting();
    virtual void initiator()=0;
    virtual void destroyer()=0;
    virtual void manager()=0;
signals:
    void updateTreeWidgetLibraryDisplay(vector < vector < QString> >);
    void updatePath(QString);

public slots:

};

#endif // DATABASEOPERATIONSVIDEO_H
```

```
#ifndef FILEFINDERAUDIO_H
#define FILEFINDERAUDIO_H

#include <QtCore>
#include <QtGui>
#include <queue>
using std::queue;

#include"databaseoperationsaudio.h"

class FileFinderAudio : public DatabaseOperationsAudio
{
    Q_OBJECT

    QDir *mDir;
    queue<QString> mFileQueue;

    queue<QString> mPathDestructor;

protected:
    explicit FileFinderAudio(QObject *parent = 0);

    queue<QString> mPath;
    void setPath(QString);
    void getDirTree();
    void initiator();
    virtual void destroyer()=0;
    virtual void manager()=0;
signals:

public slots:

};

#endif // FILEFINDERAUDIO_H
```

```
#ifndef FILEFINDERVIDEO_H
#define FILEFINDERVIDEO_H

#include <QtCore>
#include <QtGui>
#include <queue>
#include "databaseoperationsvideo.h"
using std::queue;

class FileFinderVideo : public DatabaseOperationsVideo
{
    Q_OBJECT

    QDir *mDir;
    queue<QString> mFileQueue;


protected:
    explicit FileFinderVideo(QObject *parent = 0);

    queue<QString> mPath;
    void setPath(QString);
    void getDirTree();
    void initiator();
    virtual void destroyer()=0;
    virtual void manager()=0;
signals:

public slots:

};

#endif // FILEFINDERVIDEO_H
```

```
#ifndef LIBRARYMANAGERAUDIO_H
#define LIBRARYMANAGERAUDIO_H

#include <QtCore>
#include <queue>

#include"filefinderaudio.h"

using std::queue;

class LibraryManagerAudio : public FileFinderAudio
{
    Q_OBJECT

    queue<bool> isInitiatorOnline;
    queue<bool> isDestroyerOnline;
    queue<bool> isManagerOnline;
    queue<QString> mPathDestroyer;
    void run();
    QFile *mFile;

protected:

    void destroyer();
    void manager();

public:
    explicit LibraryManagerAudio(QObject *parent = 0);

    void setInitiatorPath(QString);
    void setDestroyerPath(QString);
    void setManagerOnline();

signals:

public slots:

};

#endif // LIBRARYMANAGERAUDIO_H
```

```
#ifndef LIBRARYMANAGERVIDEO_H
#define LIBRARYMANAGERVIDEO_H
#include <QtCore>
#include <queue>

#include "filefindervideo.h"

using std::queue;

class LibraryManagerVideo : public FileFinderVideo
{
    Q_OBJECT
    queue<bool> isInitiatorOnline;
    queue<bool> isDestroyerOnline;
    queue<bool> isManagerOnline;
    queue<QString> mPathDestroyer;
    void run();
    QFile *mFile;

protected:

    void destroyer();
    void manager();

public:
    explicit LibraryManagerVideo(QObject *parent = 0);

    void setInitiatorPath(QString);
    void setDestroyerPath(QString);
    void setManagerOnline();

signals:

public slots:

};

#endif // LIBRARYMANAGERVIDEO_H
```

```cpp
#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include <QMainWindow>
#include <QtCore>
#include <vector>
#include <QSlider>
#include <QTreeWidgetItem>
#include "selectfiledialog.h"
#include "qtgstreamerdriver.h"
#include "videosink.h"

#include "librarymanageraudio.h"
#include "librarymanagervideo.h"

using std::vector;

namespace Ui {
class MainWindow;
}

class MainWindow : public QMainWindow
{
    Q_OBJECT

    LibraryManagerAudio *mDBAudio;
    LibraryManagerVideo *mDBVideo;

    enum treeWidgetSortStatesAudio
    {
        Folder,
        Artist,
        Album,
        Genre,
        Year
    };

    enum treeWidgetSortStatesVideo
    {
        FolderVideo
    };

    int mSortStateAudio=Folder;
    int mSortStateVideo=FolderVideo;
    QTreeWidgetItem *mNowPlaying = NULL;


    vector < vector < QString > > mTreeViewDataAudio;
    vector < vector < QString > > mTreeViewDataVideo;



    QtGStreamerDriver *mGstDriver;

    bool isPlaying=false;
    bool isParentAudio=true;

    QIcon mPlay ;
    QIcon mPause;
    QIcon mVolume;
    QIcon mVolumeMuted;

    QTime mPlayBacklength;
    QTime mPlayBackcurpos;
    QTimer mManagerTimer;
    unsigned long long mManagerTimerValue = 180000;
    int mManagerTimerCounter=1;
    long mTempTime;
    VideoSink *mVideoWidget;

    bool isVideoModeON=false;

    int mCurrentVolume;

public:

    explicit MainWindow(QWidget *parent = 0);

    ~MainWindow();

    int getSortStateAudio();
    void setSortStateAudio(treeWidgetSortStatesAudio vState);

    int getSortStateVideo();
    void setSortStateVideo(treeWidgetSortStatesVideo vState);

signals:
    void setPlayState();
    void setPauseState();
    void setStopState();
    void setVideoWidgetToAudio(bool);
    void goFullScreen();
    void setVolumeAtVideo(int);
    void setNowPlayingVideo(QString);

private slots:

    void getSelectedAudioPath(QString);
    void getSelectedVideoPath(QString);

    void pushButtonAddAudio_clicked();

    void pushButtonRemoveAudio_clicked();

    void pushButtonAddVideo_clicked();

    void pushButtonRemoveVideo_clicked();

    void updateTreeViewAudio(vector< vector<QString> >);

    void updateTreeViewVideo(vector< vector<QString> >);

    void onStateChanged();

    void onPositionChanged();

    void setPlayPause_clicked();
    void setNext_clicked();
    void setPrevious_clicked();


    void positionSliderMoved(int);
    void setVolume(int);

    void treeLibraryDisplay_doubleClicked();
    void treeLibraryDisplay_Addto_Queue();
    void treeLibraryDisplay_itemClicked(QTreeWidgetItem *, int);

    void treeWidgetQueue_onDoubleClicked(QTreeWidgetItem*);

    void treeCategoryChosser_doubleClicked();

    void toggleMute();
    void toFullScreen();
    void toVideoMode();
    void setVideoMode(bool);

    void shutdown();

    void pushButtonUpdate_clicked();

    void runManager();

    void toolButtonClearQueue_Clicked();

    void treeQueue_RemoveFromQueue();
private:
    Ui::MainWindow *ui;
    SelectFileDialog *mDialog;

    void sortTreeViewAudio();
    void sortTreeViewVideo();
    int setSliderOnClick(QSlider *, int);
//    bool caseInsensitiveLessThan(QTreeWidgetItem * s1 , QTreeWidgetItem * s2);
    //    void closeEvent(QCloseEvent *);
};

#endif // MAINWINDOW_H
```

```
#ifndef QTGSTREAMERDRIVER_H
#define QTGSTREAMERDRIVER_H

#include <QtCore>
#include <Qt5GStreamer/QGst/Pipeline>
#include <Qt5GStreamer/QGst/Ui/VideoWidget>
#include <Qt5GStreamer/QGlib/Connect>
#include <Qt5GStreamer/QGlib/Error>
#include <Qt5GStreamer/QGst/Pipeline>
#include <Qt5GStreamer/QGst/ElementFactory>
#include <Qt5GStreamer/QGst/Bus>
#include <Qt5GStreamer/QGst/Message>
#include <Qt5GStreamer/QGst/Query>
#include <Qt5GStreamer/QGst/ClockTime>
#include <Qt5GStreamer/QGst/Event>
#include <Qt5GStreamer/QGst/StreamVolume>

class QtGStreamerDriver: public QGst::Ui::VideoWidget
{
    Q_OBJECT

    void onBusMessage(QGst::MessagePtr);
    void handlePipelineStateChange(QGst::StateChangedMessagePtr);

    QGst::PipelinePtr mPipelinePtr;
    QTimer mPositionTimer;

public:
    QtGStreamerDriver(QWidget *parent = 0);
    ~QtGStreamerDriver();

    //Accessors
    int getVolume();
    QTime getPosition();
    QTime getDuration();
    QGst::State getState();

    //Mutators
    void setPath(QString);
    void setPosition(QTime );


public slots:
    void play();
    void pause();
    void stop();
    void setVolume(int volume);

signals:
    void positionChanged();
    void stateChanged();


};

#endif // QTGSTREAMERDRIVER_H
```

```
#ifndef SELECTFILEDIALOG_H
#define SELECTFILEDIALOG_H

#include <QDialog>
#include <QtCore>
#include <QtGui>
#include <QFileSystemModel>

namespace Ui {
class SelectFileDialog;
}

class SelectFileDialog : public QDialog
{
    Q_OBJECT

public:
    explicit SelectFileDialog(QWidget *parent = 0);
    ~SelectFileDialog();

signals:
    void selectedPath(QString);
private slots:
    void on_treeViewFolderExplorer_pressed(const QModelIndex &index);

    void on_pushButtonSelect_clicked();

    void on_pushButtonClose_clicked();

private:
    Ui::SelectFileDialog *ui;
    QFileSystemModel *mDirModel;
};

#endif // SELECTFILEDIALOG_H
```

```
#ifndef VIDEOSINK_H
#define VIDEOSINK_H

#include <QMainWindow>
#include <QCloseEvent>
#include <QSlider>
#include <QMouseEvent>
#include "qtgstreamerdriver.h"

namespace Ui {
class VideoSink;
}

class VideoSink : public QMainWindow
{
    Q_OBJECT

    QtGStreamerDriver *mGstDriver;
    QIcon mPlay ;
    QIcon mPause;
    QIcon mVolume;
    QIcon mVolumeMuted;
    long long mTempTime;

    QTime mPlayBacklength,mPlayBackcurpos;

    bool isPlaying;
    int mCurrentVolume;

    QTimer mShowControlsTimer;

public:
    explicit VideoSink(QWidget *parent = 0);
    explicit VideoSink(QWidget *,QtGStreamerDriver *);
    bool eventFilter(QObject *obj, QEvent *event);
    ~VideoSink();

signals:
    void closeMain();
    void nextClicked();
    void prevClicked();
    void setPauseState();
    void setPlayState();
    void setVolumeAtMain(int);
    void setVideoMode(bool);

private slots:

    void onStateChanged();
    void onPositionChanged();
    void setPlayPause_clicked();
    void setNext_clicked();
    void setPrevious_clicked();
    void positionSliderMoved(int);
    void setVolume(int);
    void toggleMute();
    void toFullScreen();
    void toLibraryMode();
    void setVolumeSlider(int);
    void setNowPlaying(QString);
    void hideControls();


private:
    Ui::VideoSink *ui;
    void closeEvent(QCloseEvent *);
    int setSliderOnClick(QSlider * , int );
    void showControls();
protected:
    //    void mouseMoveEvent(QMouseEvent *event);
};

#endif // VIDEOSINK_H
```

```cpp
#include "databaseoperations.h"


DatabaseOperations::DatabaseOperations(QObject *parent) :
    QThread(parent)
{

    try {
        mDBConnection.connect("localhost");
    } catch( const mongo::DBException &e )
    {
        QMessageBox::information(NULL,"Database Connection Error",QString(e.what()));

    }
}
DatabaseOperations::~DatabaseOperations()
{
    mBSONObjBuilder = new mongo::BSONObjBuilder;
    mBSONObjBuilder->append("shutdown","1");
    mBSONObj=mBSONObjBuilder->obj();
    mongo::BSONObj vTemp;
    try
    {
    mDBConnection.runCommand("admin",mBSONObj,vTemp);
    }
    catch( const mongo::DBException &e )
    {

    }
}
```

```
#include "databaseoperationsaudio.h"

DatabaseOperationsAudio::DatabaseOperationsAudio(QObject *parent) :
    DatabaseOperations(parent)
{
}

void DatabaseOperationsAudio::getSource()
{
    mCursor=mDBConnection.query("MediaTake.SourceAudio",mongo::Query());

    while(mCursor->more())
    {
        mBSONObj = mCursor->next();

        emit(updatePath(mBSONObj.getStringField("SourcePath")));
    }
    mDBConnection.killCursor(mCursor->getCursorId());
}

void DatabaseOperationsAudio::updateDB(QString vSource,QFileInfo vParent, QFileInfoList
vChildren)
{
    //    mBSONObjBuilder = new mongo::BSONObjBuilder;
    //    mBSONObjBuilder->append("Parent",vParent.absoluteFilePath().toStdString());
    //    mBSONObj = mBSONObjBuilder->obj();

    //    delete mBSONObjBuilder;

    //    mBSONObjBuilder = new mongo::BSONObjBuilder;
    //    mBSONObjBuilder->append("Parent","1");


    //    mCursor = mDBConnection.query("MediaTake.FileListAudio",mongo::Query(mBSONObj),
0,0,&(mongo::BSONObj()=mBSONObjBuilder->obj()));

    //    delete mBSONObjBuilder;

    //    if(!(mCursor->more()))
    //    {

    int vSeconds;
    int vMinutes;
    char vSecondString[3];



    for(QFileInfoList::size_type i; i < vChildren.size(); ++i)
    {

        mBSONObjBuilder = new mongo::BSONObjBuilder;
        mBSONObjBuilder->append("FilePath",vChildren[i].filePath().toStdString());

        mBSONObj = mBSONObjBuilder->obj();

        delete mBSONObjBuilder;

        mBSONObjBuilder = new mongo::BSONObjBuilder;
        mBSONObjBuilder->append("FilePath","1");

        mDBConnection.killCursor(mCursor->getCursorId());


        mCursor = mDBConnection.query("MediaTake.FileListAudio",mongo::Query(mBSONObj),
0,0,&(mongo::BSONObj()=mBSONObjBuilder->obj()));

        delete mBSONObjBuilder;
        if(!(mCursor->more()))
        {
            mFile = new
TagLib::FileRef(vChildren[i].absoluteFilePath().toStdString().c_str());

            if(!mFile->isNull())
            {

                mTag = mFile->tag();
                mAudioProp = mFile->audioProperties();


                vSeconds = mAudioProp->length() % 60;
                vMinutes = (mAudioProp->length() - vSeconds) / 60;
                std::sprintf(vSecondString, "%02i", vSeconds);

                mBSONObjBuilder = new mongo::BSONObjBuilder;
                mBSONObjBuilder->append("Source",vSource.toStdString());
                mBSONObjBuilder-
>append("Parent",vParent.absoluteFilePath().toStdString());
                mBSONObjBuilder-
>append("FileName",vChildren[i].fileName().toStdString());
                mBSONObjBuilder-
>append("FilePath",vChildren[i].absoluteFilePath().toStdString());
                mBSONObjBuilder->append("Album",mTag->album().toCString());
                mBSONObjBuilder->append("Track",std::to_string(mTag->track()));
                mBSONObjBuilder->append("Title",mTag->title().toCString());
                //                    mBSONObjBuilder->append("AlbumArtist",mFile-
>file()->properties()["ALBUMARTIST"].toString().toCString());
                mBSONObjBuilder->append("Artist",mTag->artist().toCString());
                mBSONObjBuilder-
>append("Length",std::string().append(std::to_string(vMinutes)).append(":").append(vSecon
dString));
                mBSONObjBuilder->append("Bitrate",std::to_string(mAudioProp->bitrate()));
                mBSONObjBuilder->append("Composer",mFile->file()->properties()
["COMPOSER"].toString().toCString());
                mBSONObjBuilder->append("Genre",mTag->genre().toCString());
                mBSONObjBuilder->append("Year",std::to_string(mTag->year()));

                mBSONObj = mBSONObjBuilder->obj();

                try {
                    mDBConnection.insert("MediaTake.FileListAudio",mBSONObj);
                } catch( const mongo::DBException &e ) {
                    QMessageBox::information(NULL,"Database Insertion
Error",QString(e.what()));
                }

            }
            else
            {
                mBSONObjBuilder = new mongo::BSONObjBuilder;
                mBSONObjBuilder->append("Source",vSource.toStdString());
                mBSONObjBuilder-
>append("Parent",vParent.absoluteFilePath().toStdString());
                mBSONObjBuilder-
>append("FileName",vChildren[i].fileName().toStdString());
                mBSONObjBuilder-
>append("FilePath",vChildren[i].absoluteFilePath().toStdString());
                mBSONObjBuilder->append("Album","");
                mBSONObjBuilder->append("Track","");
                mBSONObjBuilder->append("Title","");
                //                    mBSONObjBuilder->append("AlbumArtist",mFile-
>file()->properties()["ALBUMARTIST"].toString().toCString());
                mBSONObjBuilder->append("Artist","");
                mBSONObjBuilder->append("Length","");
                mBSONObjBuilder->append("Bitrate","");
                mBSONObjBuilder->append("Composer","");
                mBSONObjBuilder->append("Genre","");
                mBSONObjBuilder->append("Year","");
                mBSONObj = mBSONObjBuilder->obj();

                try {
                    mDBConnection.insert("MediaTake.FileListAudio",mBSONObj);
                } catch( const mongo::DBException &e ) {
                    QMessageBox::information(NULL,"Database Insertion
Error",QString(e.what()));
                }

            }
            delete mFile;
        }

        //mFile->~FileRef();


    }

    //    }

    mDBConnection.killCursor(mCursor->getCursorId());
    //    mCursor.release();
}

void DatabaseOperationsAudio::updateTreeView()
{
    mCursor = mDBConnection.query("MediaTake.FileListAudio", mongo::BSONObj());

    vector <QString> vTrackProp;

    mFileProp.clear();
    while(mCursor->more())
    {
        mBSONObj = mCursor->next();

        vTrackProp.push_back(mBSONObj.getStringField("Parent"));
        vTrackProp.push_back(mBSONObj.getStringField("FileName"));
        vTrackProp.push_back(mBSONObj.getStringField("FilePath"));
        vTrackProp.push_back(mBSONObj.getStringField("Album"));
        vTrackProp.push_back(mBSONObj.getStringField("Track"));
        vTrackProp.push_back(mBSONObj.getStringField("Title"));
        vTrackProp.push_back(mBSONObj.getStringField("Artist"));
        vTrackProp.push_back(mBSONObj.getStringField("Length"));
        vTrackProp.push_back(mBSONObj.getStringField("Bitrate"));
        vTrackProp.push_back(mBSONObj.getStringField("Composer"));
        vTrackProp.push_back(mBSONObj.getStringField("Genre"));
        vTrackProp.push_back(mBSONObj.getStringField("Year"));


        mFileProp.push_back(vTrackProp);
        vTrackProp.clear();
    }

    emit(updateTreeWidgetLibraryDisplay(mFileProp));

    mDBConnection.killCursor(mCursor->getCursorId());
    //    mCursor.release();
}

void DatabaseOperationsAudio::updateSource(QString vPath)
{

    mBSONObjBuilder = new mongo::BSONObjBuilder;
    mBSONObjBuilder->append("SourcePath",vPath.toStdString());

    mBSONObj = mBSONObjBuilder->obj();
    delete mBSONObjBuilder;

    mCursor=mDBConnection.query("MediaTake.SourceAudio",mongo::Query(mBSONObj));
    if(!(mCursor->more()))
    {
        try {
            mDBConnection.insert("MediaTake.SourceAudio",mBSONObj);
        } catch( const mongo::DBException &e ) {
            QMessageBox::information(NULL,"Database Insertion Error",QString(e.what()));
        }
    }

}
void DatabaseOperationsAudio::removeFromDB(QString vPath)
{
    mBSONObjBuilder = new mongo::BSONObjBuilder;
    mBSONObjBuilder->append("SourcePath",vPath.toStdString());
    mBSONObj = mBSONObjBuilder->obj();

    mDBConnection.remove("MediaTake.SourceAudio",mongo::Query(mBSONObj));

    delete mBSONObjBuilder;

    mBSONObjBuilder = new mongo::BSONObjBuilder;
    mBSONObjBuilder->append("Source",vPath.toStdString());
    mBSONObj = mBSONObjBuilder->obj();
    mDBConnection.remove("MediaTake.FileListAudio",mongo::Query(mBSONObj));

}
```

```
#include "databaseoperationsvideo.h"

DatabaseOperationsVideo::DatabaseOperationsVideo(QObject *parent) :
    DatabaseOperations(parent)
{
}

void DatabaseOperationsVideo::updateDB(QString vSource,QFileInfo vParent, QFileInfoList
vChildren)
{
    //    mBSONObjBuilder = new mongo::BSONObjBuilder;
    //    mBSONObjBuilder->append("Parent",vParent.absoluteFilePath().toStdString());
    //    mBSONObj = mBSONObjBuilder->obj();

    //    delete mBSONObjBuilder;

    //    mBSONObjBuilder = new mongo::BSONObjBuilder;
    //    mBSONObjBuilder->append("Parent","1");


    //    mCursor = mDBConnection.query("MediaTake.FileListVideo",mongo::Query(mBSONObj),
0,0,&(mongo::BSONObj()=mBSONObjBuilder->obj()));

    //    delete mBSONObjBuilder;

    //    if(!(mCursor->more()))
    //    {

    //        int vSeconds;
    //        int vMinutes;
    //        char vSecondString[3];


    for(QFileInfoList::size_type i; i < vChildren.size(); ++i)
    {

        mBSONObjBuilder = new mongo::BSONObjBuilder;
        mBSONObjBuilder->append("FilePath",vChildren[i].filePath().toStdString());

        mBSONObj = mBSONObjBuilder->obj();

        delete mBSONObjBuilder;

        mBSONObjBuilder = new mongo::BSONObjBuilder;
        mBSONObjBuilder->append("FilePath","1");

        mCursor = mDBConnection.query("MediaTake.FileListVideo",mongo::Query(mBSONObj),
0,0,&(mongo::BSONObj()=mBSONObjBuilder->obj()));

        delete mBSONObjBuilder;
        if(!(mCursor->more()))
        {
            //                mFile = new
TagLib::FileRef(vChildren[i].absoluteFilePath().toStdString().c_str());

            //               mFile->
            //                if(!mFile->isNull())
            //                {

            //                    mTag = mFile->tag();
            //                    mAudioProp = mFile->audioProperties();


            //                    vSeconds = mVideoProp->length() % 60;
            //                    vMinutes = (mVideoProp->length() - vSeconds) / 60;
            //                    std::sprintf(vSecondString, "%02i", vSeconds);

            mBSONObjBuilder = new mongo::BSONObjBuilder;
            mBSONObjBuilder->append("Source",vSource.toStdString());
            mBSONObjBuilder->append("Parent",vParent.absoluteFilePath().toStdString());
            mBSONObjBuilder->append("FileName",vChildren[i].fileName().toStdString());
            mBSONObjBuilder-
>append("FilePath",vChildren[i].absoluteFilePath().toStdString());
            //                      mBSONObjBuilder->append("Album",mTag-
>album().toCString());
            //                      mBSONObjBuilder->append("Track",std::to_string(mTag-
>track()));
            //                      mBSONObjBuilder->append("Title",mTag-
>title().toCString());
            ////                      mBSONObjBuilder->append("AlbumArtist",mFile->file()-
>properties()["ALBUMARTIST"].toString().toCString());
            //                      mBSONObjBuilder->append("Artist",mTag-
>artist().toCString());
            //                      mBSONObjBuilder-
>append("Length",std::string().append(std::to_string(vMinutes)).append(":").append(vSecon
dString));
            //                      mBSONObjBuilder-
>append("Bitrate",std::to_string(mAudioProp->bitrate()));
            //                      mBSONObjBuilder->append("Composer",mFile->file()-
>properties()["COMPOSER"].toString().toCString());
            //                      mBSONObjBuilder->append("Genre",mTag-
>genre().toCString());
            //                      mBSONObjBuilder->append("Year",std::to_string(mTag-
>year()));

            mBSONObj = mBSONObjBuilder->obj();

            try {
                mDBConnection.insert("MediaTake.FileListVideo",mBSONObj);
            } catch( const mongo::DBException &e ) {
                QMessageBox::information(NULL,"Database Insertion
Error",QString(e.what()));
            }

            //                }
            //mFile->~FileRef();
//            delete mFile;
        }

        //        }

    }
    mDBConnection.killCursor(mCursor->getCursorId());

}

void DatabaseOperationsVideo::updateTreeView()
{
    mCursor = mDBConnection.query("MediaTake.FileListVideo", mongo::BSONObj());

    vector <QString> vTrackProp;
    mFileProp.clear();
    while(mCursor->more())
    {
        mBSONObj = mCursor->next();

        vTrackProp.push_back(mBSONObj.getStringField("Parent"));
        vTrackProp.push_back(mBSONObj.getStringField("FileName"));
        vTrackProp.push_back(mBSONObj.getStringField("FilePath"));
        //        vTrackProp.push_back(mBSONObj.getStringField("Album"));
        //        vTrackProp.push_back(mBSONObj.getStringField("Track"));
        //        vTrackProp.push_back(mBSONObj.getStringField("Title"));
        //        vTrackProp.push_back(mBSONObj.getStringField("Artist"));
        //        vTrackProp.push_back(mBSONObj.getStringField("Length"));
        //        vTrackProp.push_back(mBSONObj.getStringField("Bitrate"));
        //        vTrackProp.push_back(mBSONObj.getStringField("Composer"));
        //        vTrackProp.push_back(mBSONObj.getStringField("Genre"));
        //        vTrackProp.push_back(mBSONObj.getStringField("Year"));
        mFileProp.push_back(vTrackProp);
        vTrackProp.clear();
    }

    emit(updateTreeWidgetLibraryDisplay(mFileProp));
    mDBConnection.killCursor(mCursor->getCursorId());

}

void DatabaseOperationsVideo::updateSource(QString vPath)
{
    mBSONObjBuilder = new mongo::BSONObjBuilder;
    mBSONObjBuilder->append("SourcePath",vPath.toStdString());

    mBSONObj = mBSONObjBuilder->obj();
    delete mBSONObjBuilder;

    mCursor=mDBConnection.query("MediaTake.SourceVideo",mongo::Query(mBSONObj));

    if(!(mCursor->more()))
    {
        try {
            mDBConnection.insert("MediaTake.SourceVideo",mBSONObj);
        } catch( const mongo::DBException &e ) {
            QMessageBox::information(NULL,"Database Insertion Error",QString(e.what()));
        }
    }
    mDBConnection.killCursor(mCursor->getCursorId());
    //    mCursor.release();

}

void DatabaseOperationsVideo::getSource()
{
    //    mMutex.lock();
    //    mongo::auto_ptr<mongo::DBClientCursor> vCursorVideo;
    //    mongo::BSONObj vBSONObjVideo;

    mCursor=mDBConnection.query("MediaTake.SourceVideo",mongo::Query());

    while(mCursor->more())
    {
        mBSONObj = mCursor->next();

        emit(updatePath(mBSONObj.getStringField("SourcePath")));
    }
    mDBConnection.killCursor(mCursor->getCursorId());
    //    vCursorVideo.release();
    //    mMutex.unlock();
}

void DatabaseOperationsVideo::removeFromDB(QString vPath)
{
    mBSONObjBuilder = new mongo::BSONObjBuilder;
    mBSONObjBuilder->append("SourcePath",vPath.toStdString());
    mBSONObj = mBSONObjBuilder->obj();

    mDBConnection.remove("MediaTake.SourceVideo",mongo::Query(mBSONObj));

    delete mBSONObjBuilder;

    mBSONObjBuilder = new mongo::BSONObjBuilder;
    mBSONObjBuilder->append("Source",vPath.toStdString());
    mBSONObj = mBSONObjBuilder->obj();
    mDBConnection.remove("MediaTake.FileListVideo",mongo::Query(mBSONObj));
}
```

```cpp
#include "filefinderaudio.h"

FileFinderAudio::FileFinderAudio(QObject *parent) :
    DatabaseOperationsAudio(parent)
{
}

void FileFinderAudio::getDirTree()
{
    if(mPath.front()==NULL)
    {
        return;
    }
    mDir= new QDir(mPath.front());

    //Assuming the directory mentioned in the mPath exist

    if(mDir->exists()==true)
    {
        mFileQueue.push(mDir->absolutePath());
    }
    else
    {
//          QMessageBox::information(NULL,"Error","Directory :: "+mPath.front()+" doesn't
exist");
        return;
    }

    QFileInfoList vFileList;

    QStringList vNameFilter,vDefaultNameFilter=mDir->nameFilters();
    vNameFilter<<"*.mp3"<<"*.ogg"<<"*.flac"<<"*.midi"<<"*.wav"<<"*.wma";

    mDir->setSorting(QDir::Name);

    while(mFileQueue.empty()==false)
    {
        mDir->setPath(mFileQueue.front());

        mDir->setNameFilters(vNameFilter);
        mDir->setFilter(QDir::NoDotAndDotDot|QDir::Readable|QDir::Files);


        //          vFileList=mDir->entryInfoList();
        //          emiting the current directory as mDir -> absolutePath()
        //          and the files in the directory as mDir-> entryInfoList()


        updateDB(mPath.front(),QFileInfo(mDir->absolutePath()),mDir->entryInfoList());
        //          emit(updateDirTreeView(QFileInfo(mDir->absolutePath()),mDir-
>entryInfoList()));

        mDir->setNameFilters(vDefaultNameFilter);
        mDir->setFilter(QDir::NoDotAndDotDot|QDir::Readable|QDir::Dirs);
        vFileList=mDir->entryInfoList();


        for(QFileInfoList::size_type i=0; i <vFileList.count(); ++i)
        {
            mFileQueue.push(vFileList[i].absoluteFilePath());
        }


        mFileQueue.pop();

    }

    delete mDir;
}

void FileFinderAudio::setPath(QString vPath)
{
    mPath.push(vPath);
}

void FileFinderAudio::initiator()
{
        if(mPath.front()!=NULL)
    {
        updateSource(mPath.front());
        getDirTree();
        updateTreeView();
        mPath.pop();
    }

}
```

```
#include "filefindervideo.h"

FileFinderVideo::FileFinderVideo(QObject *parent) :
    DatabaseOperationsVideo(parent)
{
}


void FileFinderVideo::getDirTree()
{
    if(mPath.front()==NULL)
    {
        return;
    }
    mDir= new QDir(mPath.front());

    //Assuming the directory mentioned in the mPath exist
    if(mDir->exists()==true)
    {
        mFileQueue.push(mDir->absolutePath());
    }
    else
    {
//          QMessageBox::StandardButton vReply;
//          vReply=QMessageBox::warning(NULL,"Error","Directory :: "+mPath.front()+"doesn't
exist",QMessageBox::Ok);

////        QMessageBox::information(NULL,"Error","Directory :: /*+mPath.front()+*/
doesn't exist");
//          if(vReply == QMessageBox::Ok)
//          {
         return;
//          }
    }


    QFileInfoList vFileList;

    QStringList vNameFilter,vDefaultNameFilter=mDir->nameFilters();
    vNameFilter<<"*.mp4"<<"*.mkv"<<"*.avi"<<"*.vorbis"<<"*.wmv"<<"*.mpeg"<<"*.
3gp"<<"*.flv"<<"*.FLV";

    mDir->setSorting(QDir::Name);

    while(mFileQueue.empty()==false)
    {
        mDir->setPath(mFileQueue.front());

        mDir->setNameFilters(vNameFilter);
        mDir->setFilter(QDir::NoDotAndDotDot|QDir::Readable|QDir::Files);


        //        vFileList=mDir->entryInfoList();
        //        emiting the current directory as mDir -> absolutePath()
        //        and the files in the directory as mDir-> entryInfoList()


        updateDB(mPath.front(),QFileInfo(mDir->absolutePath()),mDir->entryInfoList());
        //        emit(updateDirTreeView(QFileInfo(mDir->absolutePath()),mDir-
>entryInfoList()));

        mDir->setNameFilters(vDefaultNameFilter);
        mDir->setFilter(QDir::NoDotAndDotDot|QDir::Readable|QDir::Dirs);
        vFileList=mDir->entryInfoList();


        for(QFileInfoList::size_type i=0; i <vFileList.count(); ++i)
        {
            mFileQueue.push(vFileList[i].absoluteFilePath());
        }

        mFileQueue.pop();

    }
    delete mDir;

}

void FileFinderVideo::setPath(QString vPath)
{
    mPath.push(vPath);
}

void FileFinderVideo::initiator()
{

    if(mPath.front()!=NULL)
    {
      updateSource(mPath.front());
      getDirTree();
      updateTreeView();
      mPath.pop();
    }

}
```

```cpp
#include "librarymanageraudio.h"

LibraryManagerAudio::LibraryManagerAudio(QObject *parent) :
    FileFinderAudio(parent)
{
}

void LibraryManagerAudio::setInitiatorPath(QString vPath)
{
    FileFinderAudio::setPath(vPath);
    isInitiatorOnline.push(true);
}

void LibraryManagerAudio::setDestroyerPath(QString vPath)
{
    mPathDestroyer.push(vPath);
    isDestroyerOnline.push(true);
}

void LibraryManagerAudio::run()
{
    if(isInitiatorOnline.empty()==false && isInitiatorOnline.front()==true)
    {
        isInitiatorOnline.pop();
        FileFinderAudio::initiator();
    }
    else if(isDestroyerOnline.empty()==false && isDestroyerOnline.front()==true)
    {
        isDestroyerOnline.pop();
        destroyer();
    }
    else if(isManagerOnline.empty()==false && isManagerOnline.front()==true)
    {
        isManagerOnline.pop();
        manager();
    }
}

void LibraryManagerAudio::destroyer()
{
    removeFromDB(mPathDestroyer.front());
    mPathDestroyer.pop();
    updateTreeView();
}

void LibraryManagerAudio::setManagerOnline()
{
    isManagerOnline.push(true);
}

void LibraryManagerAudio::manager()
{
    mCursor = mDBConnection.query("MediaTake.FileListAudio", mongo::BSONObj());

    while(mCursor->more())
    {
        mBSONObj = mCursor->next();

        mFile = new QFile(QString::fromStdString(mBSONObj.getStringField("FilePath")));

        if(mFile->exists()==false)
        {
            mBSONObjBuilder = new mongo::BSONObjBuilder;
            mBSONObjBuilder->append("FilePath",mBSONObj.getStringField("FilePath"));
//            mBSONObj = mBSONObjBuilder->obj();
            mDBConnection.remove("MediaTake.FileListAudio",mongo::Query(mongo::BSONObj()=
mBSONObjBuilder->obj()));
        }
    }
    mDBConnection.killCursor(mCursor->getCursorId());

    mCursor=mDBConnection.query("MediaTake.SourceAudio",mongo::Query());

    while(mCursor->more())
    {
        mBSONObj = mCursor->next();
        FileFinderAudio::setPath(mBSONObj.getStringField("SourcePath"));
        getDirTree();
        this->updateTreeView();
        mPath.pop();
    }
    mDBConnection.killCursor(mCursor->getCursorId());
}
```

```cpp
#include "librarymanagervideo.h"

LibraryManagerVideo::LibraryManagerVideo(QObject *parent) :
    FileFinderVideo(parent)
{
}

void LibraryManagerVideo::setInitiatorPath(QString vPath)
{
    FileFinderVideo::setPath(vPath);
    isInitiatorOnline.push(true);
}

void LibraryManagerVideo::setDestroyerPath(QString vPath)
{
    mPathDestroyer.push(vPath);
    isDestroyerOnline.push(true);
}

void LibraryManagerVideo::run()
{
    if(isInitiatorOnline.empty()==false && isInitiatorOnline.front()==true)
    {
        isInitiatorOnline.pop();
        FileFinderVideo::initiator();
    }
    else if(isDestroyerOnline.empty()==false && isDestroyerOnline.front()==true)
    {
        isDestroyerOnline.pop();
        destroyer();
    }
    else if(isManagerOnline.empty()==false && isManagerOnline.front()==true)
    {
        isManagerOnline.pop();
        manager();
    }
}

void LibraryManagerVideo::destroyer()
{
    removeFromDB(mPathDestroyer.front());
    mPathDestroyer.pop();
    updateTreeView();
}

void LibraryManagerVideo::setManagerOnline()
{
    isManagerOnline.push(true);
}

void LibraryManagerVideo::manager()
{
    mCursor = mDBConnection.query("MediaTake.FileListVideo", mongo::BSONObj());

    while(mCursor->more())
    {
        mBSONObj = mCursor->next();

        mFile = new QFile(QString::fromStdString(mBSONObj.getStringField("FilePath")));

        if(mFile->exists()==false)
        {
            mBSONObjBuilder = new mongo::BSONObjBuilder;
            mBSONObjBuilder->append("FilePath",mBSONObj.getStringField("FilePath"));
//            mBSONObj = mBSONObjBuilder->obj();
            mDBConnection.remove("MediaTake.FileListVideo",mongo::Query(mongo::BSONObj()=
mBSONObjBuilder->obj()));
        }
    }
    mDBConnection.killCursor(mCursor->getCursorId());

    mCursor=mDBConnection.query("MediaTake.SourceVideo",mongo::Query());

    while(mCursor->more())
    {
        mBSONObj = mCursor->next();
        FileFinderVideo::setPath(mBSONObj.getStringField("SourcePath"));
        getDirTree();
        this->updateTreeView();
        mPath.pop();
    }
    mDBConnection.killCursor(mCursor->getCursorId());
}
```

```cpp
#include "mainwindow.h"
#include <QApplication>
#include <Qt5GStreamer/QGst/Init>
//#include <QtGStreamer/QGst/Init>
#include <cstdlib>
int main(int argc, char *argv[])
{
    std::system("mongod --config /etc/mongodb.conf");
    QApplication a(argc, argv);
    QGst::init(&argc, &argv);
    MainWindow w;
    w.show();

    return a.exec();
}
```

```
#include "mainwindow.h"
#include "ui_mainwindow.h"
#include <QDebug>

MainWindow::MainWindow(QWidget *parent) :
    QMainWindow(parent),
    ui(new Ui::MainWindow)
{
    ui->setupUi(this);
    connect(ui-
>pushButtonAddAudio,SIGNAL(clicked()),this,SLOT(pushButtonAddAudio_clicked()));
    connect(ui-
>pushButtonAddVideo,SIGNAL(clicked()),this,SLOT(pushButtonAddVideo_clicked()));
    connect(ui-
>pushButtonRemoveAudio,SIGNAL(clicked()),this,SLOT(pushButtonRemoveAudio_clicked()));
    connect(ui-
>pushButtonRemoveVideo,SIGNAL(clicked()),this,SLOT(pushButtonRemoveVideo_clicked()));
    connect(ui-
>pushButtonUpdateLibrary,SIGNAL(clicked()),this,SLOT(pushButtonUpdate_clicked()));

    connect(ui-
>treeWidgetLibraryDisplay,SIGNAL(doubleClicked(QModelIndex)),this,SLOT(treeLibraryDisplay
_doubleClicked()));
    connect(ui-
>treeWidgetLibraryDisplay,SIGNAL(itemClicked(QTreeWidgetItem*,int)),this,SLOT(treeLibrary
Display_itemClicked(QTreeWidgetItem*,int)));


    connect(ui->toolButtonFullScreen,SIGNAL(clicked()),this,SLOT(toFullScreen()));
    connect(ui->toolButtonVolume,SIGNAL(clicked()),this,SLOT(toggleMute()));
    connect(ui->toolButtonToScreen,SIGNAL(clicked()),this,SLOT(toVideoMode()));

    connect(ui->toolButtonNext,SIGNAL(clicked()),this,SLOT(setNext_clicked()));
    connect(ui->toolButtonPrevious,SIGNAL(clicked()),this,SLOT(setPrevious_clicked()));


    ui->treeWidgetLibraryDisplay->addAction(ui->actionAddtoQueue);
    connect(ui-
>actionAddtoQueue,SIGNAL(triggered()),this,SLOT(treeLibraryDisplay_Addto_Queue()));

    ui->treeWidgetQueue->addAction(ui->actionRemoveFromQueue);
    connect(ui-
>actionRemoveFromQueue,SIGNAL(triggered()),this,SLOT(treeQueue_RemoveFromQueue()));

    connect(ui-
>treeWidgetCatergoryChooser,SIGNAL(doubleClicked(QModelIndex)),this,SLOT(treeCategoryChos
ser_doubleClicked()));

    connect(ui-
>treeWidgetQueue,SIGNAL(itemDoubleClicked(QTreeWidgetItem*,int)),this,SLOT(treeWidgetQueu
e_onDoubleClicked(QTreeWidgetItem*)));

    connect(ui-
>toolButtonClearQueue,SIGNAL(clicked()),this,SLOT(toolButtonClearQueue_Clicked()));


    mPlay.addFile(":/ui/toolButton/icon/Resources/ui/toolButton/icon/play.png");
    mPause.addFile(":/ui/toolButton/icon/Resources/ui/toolButton/icon/pause.png");
    mVolume.addFile(":/ui/toolButton/icon/Resources/ui/toolButton/icon/volume.png");
    mVolumeMuted.addFile(":/ui/toolButton/icon/Resources/ui/toolButton/icon/
volumeMuted.png");

    ui->toolButtonPlayPause->setIcon(mPlay);

    ui->treeWidgetCatergoryChooser->expandAll();
    ui->toolButtonFullScreen->setIcon(ui->widgetPlayControls->style()-
>standardIcon(QStyle::SP_TitleBarMaxButton));
    ui->toolButtonToScreen->setIcon(ui->widgetPlayControls->style()-
>standardIcon(QStyle::SP_TitleBarNormalButton));
```

```
    mManagerTimer.setSingleShot(true);
    connect(&mManagerTimer,SIGNAL(timeout()),this,SLOT(runManager()));
    mManagerTimer.start(mManagerTimerValue);

    ui->treeWidgetQueue->hideColumn(1);

    //Database Intializations and connections
    mDBAudio = new LibraryManagerAudio(this);
    mDBVideo = new LibraryManagerVideo(this);

    connect(mDBAudio,SIGNAL(updateTreeWidgetLibraryDisplay(vector<vector<QString>
>)),this,SLOT(updateTreeViewAudio(vector<vector<QString> >)));
    connect(mDBAudio,SIGNAL(updatePath(QString)),this,SLOT(getSelectedAudioPath(QString))
);

    connect(mDBVideo,SIGNAL(updateTreeWidgetLibraryDisplay(vector<vector<QString>
>)),this,SLOT(updateTreeViewVideo(vector<vector<QString> >)));
    connect(mDBVideo,SIGNAL(updatePath(QString)),this,SLOT(getSelectedVideoPath(QString))
);

    isParentAudio=true;

    mDBAudio->getSource();
    mDBVideo->getSource();

    ui->treeWidgetLibraryDisplay->hideColumn(10);

    //Initialization and Connection with Gstreamer
    mGstDriver = new QtGStreamerDriver(this);

    connect(mGstDriver, SIGNAL(positionChanged()), this, SLOT(onPositionChanged()));
    connect(mGstDriver, SIGNAL(stateChanged()), this, SLOT(onStateChanged()));

    onStateChanged();


    connect(ui->toolButtonPlayPause,SIGNAL(clicked()),this,SLOT(setPlayPause_clicked()));
    connect(this,SIGNAL(setPlayState()),mGstDriver,SLOT(play()));
    connect(this,SIGNAL(setPauseState()),mGstDriver,SLOT(pause()));
    connect(this,SIGNAL(setStopState()),mGstDriver,SLOT(stop()));

    ui->horizontalSliderMediaPosition->setTracking(false);
    connect(ui-
>horizontalSliderMediaPosition,SIGNAL(valueChanged(int)),this,SLOT(positionSliderMoved(in
t)));

    ui->horizontalSliderVolume->setMaximum(10);

    connect(ui-
>horizontalSliderVolume,SIGNAL(valueChanged(int)),this,SLOT(setVolume(int)));
    ui->horizontalSliderVolume->setValue(05);
    ui->horizontalSliderVolume->setSliderPosition(5);
    ui->horizontalSliderVolume->setEnabled(true);

    //    Linking Video Widget

    mVideoWidget = new VideoSink(this,mGstDriver);
    mVideoWidget->show();
    mVideoWidget->setVisible(false);


    connect(mVideoWidget,SIGNAL(closeMain()),this,SLOT(shutdown()));


    connect(mVideoWidget,SIGNAL(setVideoMode(bool)),this,SLOT(setVideoMode(bool)));
    connect(mVideoWidget,SIGNAL(setVolumeAtMain(int)),this,SLOT(setVolume(int)));
    connect(mVideoWidget,SIGNAL(nextClicked()),this,SLOT(setNext_clicked()));
    connect(mVideoWidget,SIGNAL(prevClicked()),this,SLOT(setPrevious_clicked()));
    connect(this,SIGNAL(setVolumeAtVideo(int)),mVideoWidget,SLOT(setVolumeSlider(int)));
    connect(this,SIGNAL(setNowPlayingVideo(QString)),mVideoWidget,SLOT(setNowPlaying(QStr
ing)));
    connect(this,SIGNAL(goFullScreen()),mVideoWidget,SLOT(toFullScreen()));

}

MainWindow::~MainWindow()
{
    delete ui;
}


// member functions/ methods

bool caseInsensitiveLessThan(QTreeWidgetItem * s1 , QTreeWidgetItem * s2)
{
    return s1->text(0).toLower() < s2->text(0).toLower();
}

void MainWindow::sortTreeViewAudio()
{
    if(isParentAudio==false)
        return;
    if(mTreeViewDataAudio.empty()==true)
    {
        ui->treeWidgetLibraryDisplay->clear();
        return;
    }

    ui->treeWidgetLibraryDisplay->clear();


    QTreeWidgetItem *vRootItem , *vChildItem;

    QString vCurrentRoot;

    int vIndex;

    vector<QTreeWidgetItem *> vRootList;

    switch(mSortStateAudio)
    {
    case(Folder):
    {
        vIndex=0;
        break;
    }
    case(Artist):
    {
        vIndex=6;
        break;
    }
    case(Album):
    {
        vIndex=3;
        break;
    }
    case(Year):
    {
        vIndex=11;
        break;
    }
    case(Genre):
    {
        vIndex=10;
        break;
```

```cpp
        }
    }

    QFont vTempFont;
    vTempFont.setBold(true);

    vCurrentRoot = mTreeViewDataAudio[0][vIndex];
    if(mSortStateAudio==Folder)
    {
        vRootItem = new QTreeWidgetItem(ui->treeWidgetLibraryDisplay);
        //    vRootItem->setText(0,QFileInfo(vCurrentRoot).fileName()+" :
"+QFileInfo(vCurrentRoot).filePath());
        vRootItem->setText(0,QFileInfo(vCurrentRoot).fileName());
        vRootItem->setFont(0,vTempFont);
        vRootItem->setText(10,QFileInfo(vCurrentRoot).filePath());
        vRootList.push_back(vRootItem);
    }
    else
    {
        vRootItem = new QTreeWidgetItem(ui->treeWidgetLibraryDisplay);
        vRootItem->setText(0,vCurrentRoot);
        vRootItem->setFont(0,vTempFont);
        vRootList.push_back(vRootItem);
    }
    bool foundRoot=false;
    int vRootPos;

    for(unsigned long long i =0 ; i < mTreeViewDataAudio.size(); ++i)
    {
        foundRoot=false;
        vCurrentRoot=mTreeViewDataAudio[i][vIndex];
        for(unsigned long long j=0; j < vRootList.size(); ++j)
        {
            if(mSortStateAudio!=Folder)
            {
                if( vRootList[j]->text(0) == mTreeViewDataAudio[i][vIndex] )
                {
                    vRootPos=j;
                    foundRoot=true;
                    break;
                }
            }
            else
            {
                if( vRootList[j]->text(10) == mTreeViewDataAudio[i][vIndex] )
                {
                    vRootPos=j;
                    foundRoot=true;
                    break;
                }
            }
        }

        if(foundRoot==false)
        {
            if(mSortStateAudio==Folder)
            {
                vRootItem = new QTreeWidgetItem();
                vRootItem->setText(0,QFileInfo(vCurrentRoot).fileName());
                vRootItem->setFont(0,vTempFont);
                vRootItem->setText(10,QFileInfo(vCurrentRoot).filePath());
                vRootList.push_back(vRootItem);
                vRootPos=vRootList.size()-1;
            }
            else
            {
                vRootItem = new QTreeWidgetItem();
                vRootItem->setText(0,vCurrentRoot);
                vRootItem->setFont(0,vTempFont);
                vRootList.push_back(vRootItem);
                vRootPos=vRootList.size()-1;
            }
        }

        //        if (vCurrentRoot != mTreeViewData[i][vIndex])
        //        {
        //            vCurrentRoot = mTreeViewData[i][vIndex];
        //            vRootItem = new QTreeWidgetItem(ui->treeWidgetLibraryDisplay);
        //            //            vRootItem-
>setText(0,QFileInfo(vCurrentRoot).fileName()+": "+QFileInfo(vCurrentRoot).filePath());
        //            vRootItem->setText(0,QFileInfo(vCurrentRoot).fileName());
        //            vRootItem->setFont(0,vTempFont);
        //            vRootItem->setText(10,QFileInfo(vCurrentRoot).filePath());
        //        }


        vChildItem = new QTreeWidgetItem();
        vChildItem->setText(0,mTreeViewDataAudio[i][1]);
        vChildItem->setText(1,mTreeViewDataAudio[i][4]);
        vChildItem->setText(2,mTreeViewDataAudio[i][3]);
        vChildItem->setText(3,mTreeViewDataAudio[i][5]);
        vChildItem->setText(4,mTreeViewDataAudio[i][7]);
        vChildItem->setText(5,mTreeViewDataAudio[i][6]);
        vChildItem->setText(6,mTreeViewDataAudio[i][10]);
        vChildItem->setText(7,mTreeViewDataAudio[i][11]);
        vChildItem->setText(8,mTreeViewDataAudio[i][8]);
        vChildItem->setText(9,mTreeViewDataAudio[i][9]);
        vChildItem->setText(10,mTreeViewDataAudio[i][2]);
        vRootList[vRootPos]->addChild(vChildItem);
        //        for(unsigned long long i=0; i < mTreeViewData.size();++i)

    }


    //    if(mSortStateAudio!=Folder)
    //    {
    //        std::sort(vRootList.begin(),vRootList.end());
    qSort(vRootList.begin(),vRootList.end(),caseInsensitiveLessThan);
    for(unsigned long long j=0; j < vRootList.size(); ++j)
    {
        vRootItem = new QTreeWidgetItem(ui->treeWidgetLibraryDisplay);
        vRootItem->setText(0,vRootList[j]->text(0));
        vRootItem->setFont(0,vTempFont);
        vRootItem->addChildren(vRootList[j]->takeChildren());
    }
    qDeleteAll(vRootList);
    //    }

    ui->treeWidgetLibraryDisplay->resizeColumnToContents(0);

}

void MainWindow::sortTreeViewVideo()
{
    if(isParentAudio==true)
        return;

    if(mTreeViewDataVideo.empty()==true)
    {    ui->treeWidgetLibraryDisplay->clear();
        return;
    }

    ui->treeWidgetLibraryDisplay->clear();


    QTreeWidgetItem *vRootItem , *vChildItem;

    QString vCurrentRoot;
    vector<QTreeWidgetItem *> vRootList;

    QFont vTempFont;
    vTempFont.setBold(true);

    vCurrentRoot = mTreeViewDataVideo[0][0];
    //    if(mSortStateVideo==FolderVideo)
    //    {
    vRootItem = new QTreeWidgetItem(ui->treeWidgetLibraryDisplay);
    //    vRootItem->setText(0,QFileInfo(vCurrentRoot).fileName()+" :
"+QFileInfo(vCurrentRoot).filePath());
    vRootItem->setText(0,QFileInfo(vCurrentRoot).fileName());
    vRootItem->setFont(0,vTempFont);
    vRootItem->setText(10,QFileInfo(vCurrentRoot).filePath());
    vRootList.push_back(vRootItem);
    //    }
    bool foundRoot=false;
    int vRootPos;

    for(unsigned long long i =0 ; i < mTreeViewDataVideo.size(); ++i)
    {
        foundRoot=false;
        vCurrentRoot=mTreeViewDataVideo[i][0];
        for(unsigned long long j=0; j < vRootList.size(); ++j)
        {
            if( vRootList[j]->text(10) == mTreeViewDataVideo[i][0] )
            {
                vRootPos=j;
                foundRoot=true;
                break;
            }

        }

        if(foundRoot==false)
        {
            //            if(mSortStateAudio==Folder)
            //            {
            vRootItem = new QTreeWidgetItem();
            vRootItem->setText(0,QFileInfo(vCurrentRoot).fileName());
            vRootItem->setFont(0,vTempFont);
            vRootItem->setText(10,QFileInfo(vCurrentRoot).filePath());
            vRootList.push_back(vRootItem);
            vRootPos=vRootList.size()-1;

        }

        //        if (vCurrentRoot != mTreeViewData[i][vIndex])
        //        {
        //            vCurrentRoot = mTreeViewData[i][vIndex];
        //            vRootItem = new QTreeWidgetItem(ui->treeWidgetLibraryDisplay);
        //            //            vRootItem-
>setText(0,QFileInfo(vCurrentRoot).fileName()+": "+QFileInfo(vCurrentRoot).filePath());
        //            vRootItem->setText(0,QFileInfo(vCurrentRoot).fileName());
        //            vRootItem->setFont(0,vTempFont);
        //            vRootItem->setText(10,QFileInfo(vCurrentRoot).filePath());
        //        }


        vChildItem = new QTreeWidgetItem();
        vChildItem->setText(0,mTreeViewDataVideo[i][1]);
        //        vChildItem->setText(1,mTreeViewDataVideo[i][4]);
        //        vChildItem->setText(2,mTreeViewDataVideo[i][3]);
        //        vChildItem->setText(3,mTreeViewDataVideo[i][5]);
        //        vChildItem->setText(4,mTreeViewDataVideo[i][7]);
        //        vChildItem->setText(5,mTreeViewDataVideo[i][6]);
        //        vChildItem->setText(6,mTreeViewDataVideo[i][10]);
        //        vChildItem->setText(7,mTreeViewDataVideo[i][11]);
```

```cpp
        //        vChildItem->setText(8,mTreeViewDataVideo[i][8]);
        //        vChildItem->setText(9,mTreeViewDataVideo[i][9]);
        vChildItem->setText(10,mTreeViewDataVideo[i][2]);
        vRootList[vRootPos]->addChild(vChildItem);
        //        for(unsigned long long i=0; i < mTreeViewData.size();++i)

    }

    qSort(vRootList.begin(),vRootList.end(),caseInsensitiveLessThan);
    for(unsigned long long j=0; j < vRootList.size(); ++j)
    {
        vRootItem = new QTreeWidgetItem(ui->treeWidgetLibraryDisplay);
        vRootItem->setText(0,vRootList[j]->text(0));
        vRootItem->setFont(0,vTempFont);
        vRootItem->addChildren(vRootList[j]->takeChildren());
    }
    qDeleteAll(vRootList);

    ui->treeWidgetLibraryDisplay->resizeColumnToContents(0);
}

int MainWindow::setSliderOnClick(QSlider * vQSlider , int vClickedPosition)
{
    Qt::MouseButtons vMouseButton = QApplication::mouseButtons();
    QPoint vMousePos = vQSlider->mapFromGlobal(QCursor::pos());
    bool isClicked = (vMouseButton &Qt::LeftButton) &&
            (vMousePos.x() >=0 && vMousePos.y() >=0 &&
             vMousePos.x() < vQSlider->size().width() &&
             vMousePos.y() < vQSlider->size().height());

    if(isClicked == true)
    {
        float vPosRatio = vMousePos.x() / (float)vQSlider->size().width();
        int vSliderRange = vQSlider->maximum()-vQSlider->minimum();
        int vSliderPositionUnderMouse = vQSlider->minimum() + vSliderRange *vPosRatio;

        if(vSliderPositionUnderMouse != vClickedPosition)
        {
            vQSlider->setValue(vSliderPositionUnderMouse);
            return vSliderPositionUnderMouse;
        }
    }
    return vClickedPosition;
}



// Getter Methods

int MainWindow::getSortStateAudio()
{
    return mSortStateAudio;
}

// Setter Methods

void MainWindow::setSortStateAudio(treeWidgetSortStatesAudio vState)
{
    switch(vState)
    {
    case(Folder):
    {
        mSortStateAudio=Folder;
        break;
    }
    case(Artist):
    {
        mSortStateAudio=Artist;
        break;
    }
    case(Album):
    {
        mSortStateAudio=Album;
        break;
    }
    case(Year):
    {
        mSortStateAudio=Year;
        break;
    }
    case(Genre):
    {
        mSortStateAudio=Genre;
        break;
    }
    default:
    {
        throw std::string("Audio State Cannot be Matched");

    }
    }


}

//SLOTS
void MainWindow::getSelectedAudioPath(QString vPath)
{

    bool doPathExist=false;

    for(int i= 0; i < ui->listWidgetAudioLibrary->count();++i)
    {
        if(ui->listWidgetAudioLibrary->item(i)->text()==vPath)
        {
            doPathExist=true;
        }

    }
    if(doPathExist==false)
    {
        ui->listWidgetAudioLibrary->addItem(vPath);
        mDBAudio->setInitiatorPath(vPath);
        qRegisterMetaType<QFileInfoList>("QFileInfoList");
        qRegisterMetaType<vector<vector<QString> > >("vector<vector<QString> >");
        mDBAudio->start(QThread::HighPriority);
    }
    else
    {
        QMessageBox::information(NULL,"Warning","Path : "+vPath+"Already Exist");
    }

}

void MainWindow::getSelectedVideoPath(QString vPath)
{
    bool doPathExist=false;

    for(int i= 0; i < ui->listWidgetVideoLibrary->count();++i)
    {
        if(ui->listWidgetVideoLibrary->item(i)->text()==vPath)
        {
            doPathExist=true;
        }

    }
    if(doPathExist==false)
    {
        ui->listWidgetVideoLibrary->addItem(vPath);

        mDBVideo->setInitiatorPath(vPath);
        qRegisterMetaType<QFileInfoList>("QFileInfoList");
        qRegisterMetaType<vector<vector<QString> > >("vector<vector<QString> >");
        mDBVideo->start(QThread::HighestPriority);

    }
    else
    {
        QMessageBox::information(NULL,"Warning","Path : "+vPath+"Already Exist");
    }

    //    ui->listWidgetVideoLibrary->addItem(vPath);
}

void MainWindow::pushButtonAddAudio_clicked()
{
    mDialog = new SelectFileDialog(this);
    connect
(mDialog,SIGNAL(selectedPath(QString)),this,SLOT(getSelectedAudioPath(QString)));
    mDialog->exec();
}

void MainWindow::pushButtonRemoveAudio_clicked()
{
    mDBAudio->setDestroyerPath(ui->listWidgetAudioLibrary->currentItem()->text());
    mDBAudio->start(QThread::HighestPriority);

    qDeleteAll(ui->listWidgetAudioLibrary->selectedItems());
}

void MainWindow::pushButtonAddVideo_clicked()
{
    mDialog = new SelectFileDialog(this);
    connect
(mDialog,SIGNAL(selectedPath(QString)),this,SLOT(getSelectedVideoPath(QString)));
    mDialog->exec();
}

void MainWindow::pushButtonRemoveVideo_clicked()
{
    mDBVideo->setDestroyerPath(ui->listWidgetVideoLibrary->currentItem()->text());
    mDBVideo->start(QThread::HighestPriority);

    qDeleteAll(ui->listWidgetVideoLibrary->selectedItems());
}

void MainWindow::updateTreeViewAudio(vector<vector<QString> > vTreeViewData)
{
    mTreeViewDataAudio.clear();
    mTreeViewDataAudio=vTreeViewData;
    sortTreeViewAudio();
}

void MainWindow::updateTreeViewVideo(vector<vector<QString> > vTreeViewData)
{
    mTreeViewDataVideo.clear();
    mTreeViewDataVideo=vTreeViewData;
    sortTreeViewVideo();
}


void MainWindow::onStateChanged()
{

    QGst::State vNewState = mGstDriver->getState();

    if(vNewState == QGst::StateNull || vNewState == QGst::StatePaused)
    {
        ui->toolButtonPlayPause->setIcon(mPlay);
        isPlaying=false;
    }
    else if(vNewState == QGst::StatePlaying)
    {
        ui->toolButtonPlayPause->setIcon(mPause);
        mTempTime = 0;
        mTempTime = mTempTime + ( mGstDriver->getDuration().hour() * 60);
        mTempTime = ( mTempTime +  mGstDriver->getDuration().minute() ) * 60;
        mTempTime = mTempTime + (mGstDriver->getDuration().second());
        ui->horizontalSliderMediaPosition->setMaximum(mTempTime);
        isPlaying=true;
    }



    ui->toolButtonNext->setEnabled(vNewState != QGst::StateNull);
    ui->toolButtonPrevious->setEnabled(vNewState != QGst::StateNull);
    ui->horizontalSliderMediaPosition->setEnabled(vNewState != QGst::StateNull);

    if(vNewState == QGst::StatePaused || vNewState == QGst::StatePlaying)
    {
        ui->labelTime->show();
        ui->labelLength->show();
        ui->labelTime->setEnabled(true);
        ui->horizontalSliderMediaPosition->setEnabled(true);
    }
    else
    {
        ui->labelTime->hide();
        ui->labelLength->hide();
    }


    //if we are in Null state, call onPositionChanged() to restore
    //the position of the slider and the text on the label


    if (vNewState == QGst::StateNull) {
        onPositionChanged();

        ui->labelNowPlaying->setText("");
        emit(setNowPlayingVideo(""));

    }
}

void MainWindow::onPositionChanged()
{

    if (mGstDriver->getState() != QGst::StateReady && mGstDriver->getState() !=
QGst::StateNull)
    {
        mPlayBacklength = mGstDriver->getDuration();
        if(mPlayBacklength.hour()==0)
        {
            ui->labelLength->setText(mPlayBacklength.toString("mm:ss"));
        }
        else
        {
            ui->labelLength->setText(mPlayBacklength.toString("HH:mm:ss"));
        }
        mPlayBackcurpos = mGstDriver->getPosition();


        if(mPlayBackcurpos.toString() == mPlayBacklength.toString())
        {
            if(mNowPlaying != NULL && ui->treeWidgetQueue->itemBelow(mNowPlaying))
            {
                treeWidgetQueue_onDoubleClicked(ui->treeWidgetQueue-
>itemBelow(mNowPlaying));
            }
        }
    }

    if(mPlayBackcurpos.hour() ==0)
    {
        ui->labelTime->setText(mPlayBackcurpos.toString("mm:ss"));
    }
    else
    {
        ui->labelTime->setText(mPlayBackcurpos.toString("HH:mm:ss"));
    }

    if(mGstDriver->getState()!= QGst::StateNull)
    {
        mTempTime = 0;
        mTempTime = mTempTime + ( mPlayBackcurpos.hour() * 60);
        mTempTime = ( mTempTime +  mPlayBackcurpos.minute() ) * 60;
        mTempTime = mTempTime + (mPlayBackcurpos.second());
        if(ui->horizontalSliderMediaPosition->isSliderDown()==false)
        {
            ui->horizontalSliderMediaPosition->setSliderPosition(mTempTime);
        }
    }
    else
    {
        ui->horizontalSliderMediaPosition->setValue(0);
        ui->horizontalSliderMediaPosition->setSliderPosition(0);
        //        QMessageBox::information(this,"State Changed","Playing");


    }
}


void MainWindow::setPlayPause_clicked()
{
    if(isPlaying==true)
    {
        isPlaying=false;
        emit(setPauseState());
    }
    else
    {
        isPlaying=true;
        emit(setPlayState());
    }

}

void MainWindow::setNext_clicked()
{
    if(mNowPlaying != NULL && ui->treeWidgetQueue->itemBelow(mNowPlaying))
    {
        treeWidgetQueue_onDoubleClicked(ui->treeWidgetQueue->itemBelow(mNowPlaying));
    }
}
void MainWindow::setPrevious_clicked()
{
    if(mNowPlaying != NULL && ui->treeWidgetQueue->itemAbove(mNowPlaying))
    {
        treeWidgetQueue_onDoubleClicked(ui->treeWidgetQueue->itemAbove(mNowPlaying));
    }

}
```

```
void MainWindow::treeLibraryDisplay_doubleClicked()
{
    QTreeWidgetItem *vItem;
    QTreeWidgetItemIterator vItemIterator(ui->treeWidgetQueue);

    bool doItemExist=false;

    while(*vItemIterator)
    {
        if( (*vItemIterator)->text(1) != ui->treeWidgetLibraryDisplay->currentItem()-
>text(10) )
        {
            (*vItemIterator)->setBackgroundColor(0,Qt::white);
            (*vItemIterator)->setSelected(false);
        }
        else
        {
            doItemExist=true;
            (*vItemIterator)->setBackgroundColor(0,Qt::green);
            (*vItemIterator)->setSelected(true);
            mNowPlaying=(*vItemIterator);
        }
        vItemIterator++;
    }

    if(ui->treeWidgetLibraryDisplay->currentItem()->parent() != NULL)
    {

        emit(setStopState());

        if(isParentAudio==false)
        {

            if(doItemExist==false)
            {
                vItem = new QTreeWidgetItem(ui->treeWidgetQueue);
                vItem->setText(0,ui->treeWidgetLibraryDisplay->currentItem()->text(0));
                vItem->setText(1,ui->treeWidgetLibraryDisplay->currentItem()->text(10));
                vItem->setBackgroundColor(0,Qt::green);

                mNowPlaying=vItem;
            }
            ui->labelNowPlaying->setText(ui->treeWidgetLibraryDisplay->currentItem()-
>text(0));
            emit(setNowPlayingVideo(ui->treeWidgetLibraryDisplay->currentItem()-
>text(0)));
            toVideoMode();
        }
        else
        {


            if(ui->treeWidgetLibraryDisplay->currentItem()->text(3) != "")
            {
                if(doItemExist==false)
                {
                    vItem = new QTreeWidgetItem(ui->treeWidgetQueue);
                    vItem->setText(0,ui->treeWidgetLibraryDisplay->currentItem()-
>text(3));

                    vItem->setText(1,ui->treeWidgetLibraryDisplay->currentItem()-
>text(10));

                    vItem->setBackgroundColor(0,Qt::green);
                    vItem->setSelected(true);
                    mNowPlaying=vItem;
                }
                ui->labelNowPlaying->setText(ui->treeWidgetLibraryDisplay->currentItem()-
>text(3));
                emit(setNowPlayingVideo(ui->treeWidgetLibraryDisplay->currentItem()-
>text(3)));
```

```
            }
            else
            {
                if(doItemExist==false)
                {
                    vItem = new QTreeWidgetItem(ui->treeWidgetQueue);
                    vItem->setText(0,ui->treeWidgetLibraryDisplay->currentItem()-
>text(0));
                    vItem->setText(1,ui->treeWidgetLibraryDisplay->currentItem()-
>text(10));
                    vItem->setBackgroundColor(0,Qt::green);
                    vItem->setSelected(true);
                    mNowPlaying=vItem;
                }
                ui->labelNowPlaying->setText(ui->treeWidgetLibraryDisplay->currentItem()-
>text(0));
                emit(setNowPlayingVideo(ui->treeWidgetLibraryDisplay->currentItem()-
>text(0)));
            }
        }


        mGstDriver->setPath(ui->treeWidgetLibraryDisplay->currentItem()->text(10));
        //        ui->toolButtonPlayPause->click();
        emit(setPlayState());
    }

    //    QMessageBox::information(this,"Testing",vCurrentItem->text(10));
}

void MainWindow::treeLibraryDisplay_itemClicked(QTreeWidgetItem * vItem,int vColumn)
{
    ui->treeWidgetLibraryDisplay->resizeColumnToContents(vColumn);
}

void MainWindow::positionSliderMoved(int vSliderValue)
{

    vSliderValue=setSliderOnClick(ui->horizontalSliderMediaPosition,vSliderValue);
    mTempTime = vSliderValue;
    int vSeconds = mTempTime % 60;
    mTempTime = mTempTime/60 ;
    int vMinutes = mTempTime % 60;
    int vHours = mTempTime/60;
    QTime vPosition(vHours,vMinutes,vSeconds);
    mGstDriver->setPosition(vPosition);
}

void MainWindow::setVolume(int vSliderValue)
{
    vSliderValue = setSliderOnClick(ui->horizontalSliderVolume,vSliderValue);
    mGstDriver->setVolume(vSliderValue);
    ui->horizontalSliderVolume->setSliderPosition(vSliderValue);
    emit(setVolumeAtVideo(vSliderValue));
}

void MainWindow::treeCategoryChosser_doubleClicked()
{
    if(ui->treeWidgetCatergoryChooser->currentItem()->parent() != NULL)
    {
        if(ui->treeWidgetCatergoryChooser->currentItem()->parent()->text(0)!="Video")
        {
            ui->treeWidgetLibraryDisplay->showColumn(1);
            ui->treeWidgetLibraryDisplay->showColumn(2);
            ui->treeWidgetLibraryDisplay->showColumn(3);
            ui->treeWidgetLibraryDisplay->showColumn(5);
            ui->treeWidgetLibraryDisplay->showColumn(6);
            ui->treeWidgetLibraryDisplay->showColumn(7);
            ui->treeWidgetLibraryDisplay->showColumn(8);
            ui->treeWidgetLibraryDisplay->showColumn(9);
            ui->treeWidgetLibraryDisplay->hideColumn(10);

            isParentAudio=true;
            switch(ui->treeWidgetCatergoryChooser->currentIndex().row())
            {
            case(0):
            {
                mSortStateAudio=Folder;
                break;
            }
            case(1):
            {
                mSortStateAudio=Artist;
                break;
            }
            case(2):
            {
                mSortStateAudio=Album;
                break;
            }
            case(3):
            {
                mSortStateAudio=Genre;
                break;
            }
            case(4):
            {
                mSortStateAudio=Year;
                break;
            }
            }
            sortTreeViewAudio();

        }
        else
        {
            isParentAudio=false;
            sortTreeViewVideo();
            ui->treeWidgetLibraryDisplay->showColumn(10);
            ui->treeWidgetLibraryDisplay->hideColumn(1);
            ui->treeWidgetLibraryDisplay->hideColumn(2);
            ui->treeWidgetLibraryDisplay->hideColumn(3);
            ui->treeWidgetLibraryDisplay->hideColumn(5);
            ui->treeWidgetLibraryDisplay->hideColumn(6);
            ui->treeWidgetLibraryDisplay->hideColumn(7);
            ui->treeWidgetLibraryDisplay->hideColumn(8);
            ui->treeWidgetLibraryDisplay->hideColumn(9);
        }

    }
    else
    {
        if(ui->treeWidgetCatergoryChooser->currentItem()->text(0)=="Video")
        {
            isParentAudio=false;

        }
        else
        {
            isParentAudio=true;
        }
    }

}

void MainWindow::treeLibraryDisplay_Addto_Queue()
{
    QTreeWidgetItem *vItem,*vTemp;
    QTreeWidgetItemIterator vItemIterator(ui->treeWidgetQueue);
    bool doItemExist=false;

    for(QList<QListWidget*>::size_type i = 0; i<ui->treeWidgetLibraryDisplay-
>selectedItems().size(); ++i)
    {
        vTemp=ui->treeWidgetLibraryDisplay->selectedItems().at(i);
        vItemIterator=QTreeWidgetItemIterator(ui->treeWidgetQueue);
        if(vTemp->parent()==NULL)
        {
            long j=0;
            if(ui->treeWidgetQueue->topLevelItemCount()==0)
            {
                vItem = new QTreeWidgetItem(ui->treeWidgetQueue);
                vItem->setText(0,vTemp->child(0)->text(0));
                vItem->setText(1,vTemp->child(0)->text(10));
                treeWidgetQueue_onDoubleClicked(vItem);
                j=1;
            }

            for(;j<vTemp->childCount();++j)
            {
                while(*vItemIterator)
                {
                    if( (*vItemIterator)->text(1) == vTemp->child(j)->text(10) )
                    {
                        doItemExist=true;
                        break;
                    }
                    vItemIterator++;
                }
                if(doItemExist==false)
                {
                    vItem = new QTreeWidgetItem(ui->treeWidgetQueue);
                    vItem->setText(0,vTemp->child(j)->text(0));
                    vItem->setText(1,vTemp->child(j)->text(10));
                }
                doItemExist=false;
            }

        }
        else
        {
            if(ui->treeWidgetQueue->topLevelItemCount()==0)
            {
                vItem = new QTreeWidgetItem(ui->treeWidgetQueue);
                vItem->setText(0,vTemp->text(0));
                vItem->setText(1,vTemp->text(10));
                treeWidgetQueue_onDoubleClicked(vItem);

            }
            else
            {
                while(*vItemIterator)
                {
                    if( (*vItemIterator)->text(1) == vTemp->text(10) )
                    {
                        doItemExist=true;
                        break;
                    }
                    vItemIterator++;
                }
                if(doItemExist==false)
                {
                    vItem = new QTreeWidgetItem(ui->treeWidgetQueue);
                    vItem->setText(0,vTemp->text(0));
                    vItem->setText(1,vTemp->text(10));
                }
            }
        }
    }
}

void MainWindow::shutdown()
{
    MainWindow::close();
}

void MainWindow::toggleMute()
{
    if(ui->toolButtonVolume->isChecked()==true)
    {
        mCurrentVolume=ui->horizontalSliderVolume->value();
        setVolume(0);
        ui->toolButtonVolume->setIcon(mVolumeMuted);
    }
    else
    {
        setVolume(mCurrentVolume);
        ui->toolButtonVolume->setIcon(mVolume);
    }
}

void MainWindow::toFullScreen()
{
    isVideoModeON=true;
    //emit(setVideoWidget(mGstDriver));
    emit(goFullScreen());
    mVideoWidget->show();
    MainWindow::hide();
}

void MainWindow::toVideoMode()
{
    isVideoModeON=true;
    //    emit(setVolume(ui->horizontalSliderVolume->value()));

    MainWindow::hide();
    mVideoWidget->show();
}

void MainWindow::setVideoMode(bool vValue)
{
    isVideoModeON=vValue;
    MainWindow::show();
}

void MainWindow::pushButtonUpdate_clicked()
{
    mDBAudio->setManagerOnline();
    mDBAudio->start(QThread::HighestPriority);
    mDBVideo->setManagerOnline();
    mDBVideo->start(QThread::HighestPriority);
    mManagerTimerCounter=1;
}

void MainWindow::runManager()
{
    if(mManagerTimerCounter < 7)
    {
        mDBAudio->setManagerOnline();
        mDBAudio->start(QThread::HighestPriority);
        mDBVideo->setManagerOnline();
        mDBVideo->start(QThread::HighestPriority);
        mManagerTimerCounter++;
        mManagerTimer.start(mManagerTimerValue*mManagerTimerCounter);
    }
}
```

```
void MainWindow::treeWidgetQueue_onDoubleClicked(QTreeWidgetItem * vItem)
{
    mNowPlaying = vItem;

    QTreeWidgetItemIterator vItemIterator(ui->treeWidgetQueue);

    while(*vItemIterator)
    {
        if( (*vItemIterator)->text(1) != vItem->text(1) )
        {
            (*vItemIterator)->setBackgroundColor(0,Qt::white);
            (*vItemIterator)->setSelected(false);
        }
        else
        {
            (*vItemIterator)->setBackgroundColor(0,Qt::green);
            (*vItemIterator)->setSelected(true);
            mNowPlaying=(*vItemIterator);
        }
        vItemIterator++;
    }
    emit(setStopState());
    mGstDriver->setPath(vItem->text(1));
    emit(setPlayState());
    if(isParentAudio==false)
    {

        emit(setNowPlayingVideo(vItem->text(0)));
        toVideoMode();
    }
    else
    {
        ui->labelNowPlaying->setText(vItem->text(0));
        emit(setNowPlayingVideo(vItem->text(0)));
    }


}

void MainWindow::toolButtonClearQueue_Clicked()
{
    ui->treeWidgetQueue->clear();
    emit(setStopState());
    mGstDriver->setPath("");
}

void MainWindow::treeQueue_RemoveFromQueue()
{
    qDebug()<<ui->treeWidgetQueue->selectedItems().at(0)->text(1);
    if(ui->treeWidgetQueue->selectedItems().at(0)->text(1)==mNowPlaying->text(1))
    {
        emit(setStopState());
        mNowPlaying=ui->treeWidgetQueue->itemBelow(ui->treeWidgetQueue-
>selectedItems().at(0));
        mGstDriver->setPath("");
        emit(setPlayState());
    }
    qDeleteAll(ui->treeWidgetQueue->selectedItems());
}
```

```
#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);
        }
    }
}
```

```cpp
#include "selectfiledialog.h"
#include "ui_selectfiledialog.h"

SelectFileDialog::SelectFileDialog(QWidget *parent) :
    QDialog(parent),
    ui(new Ui::SelectFileDialog)
{
    ui->setupUi(this);

    mDirModel=new QFileSystemModel(this);
    mDirModel->setRootPath("/");
    mDirModel->setFilter(QDir::NoDotAndDotDot | QDir::AllDirs);

    QModelIndex vIndex = mDirModel->index("/home");
    ui->lineEditFilePath->setText("/home");

    ui->treeViewFolderExplorer->setModel(mDirModel);
    ui->treeViewFolderExplorer->expand(vIndex);
    ui->treeViewFolderExplorer->scrollTo(vIndex);
    ui->treeViewFolderExplorer->setCurrentIndex(vIndex);
    ui->treeViewFolderExplorer->setColumnWidth(0,200);
//    ui->treeViewFolderExplorer->resizeColumnToContents(0);

}

SelectFileDialog::~SelectFileDialog()
{
    delete ui;
}

void SelectFileDialog::on_treeViewFolderExplorer_pressed(const QModelIndex &index)
{

    ui->lineEditFilePath->setText(mDirModel->fileInfo(index).absoluteFilePath());

}

void SelectFileDialog::on_pushButtonSelect_clicked()
{
    emit selectedPath(ui->lineEditFilePath->text());
    SelectFileDialog::close();
}

void SelectFileDialog::on_pushButtonClose_clicked()
{
    SelectFileDialog::close();
}
```

```
#include "videosink.h"
#include "ui_videosink.h"


VideoSink::VideoSink(QWidget *parent) :
    QMainWindow(parent),
    ui(new Ui::VideoSink)
{
    ui->setupUi(this);
}

VideoSink::VideoSink(QWidget *parent,QtGStreamerDriver * vGstDriver):QMainWindow(parent),
    ui(new Ui::VideoSink)
{
    ui->setupUi(this);

    ui->verticalLayoutVideoLayout->addWidget(vGstDriver);

    mGstDriver=vGstDriver;

    //    ui->toolButtonNext->setIcon(ui->centralwidget->style()-
>standardIcon(QStyle::SP_MediaSkipForward));
    //    ui->toolButtonPrevious->setIcon(ui->centralwidget->style()-
>standardIcon(QStyle::SP_MediaSkipBackward));
    ui->toolButtonFullScreen->setIcon(ui->centralwidget->style()-
>standardIcon(QStyle::SP_TitleBarMaxButton));
    //    ui->toolButtonVolume->setIcon(ui->centralwidget->style()-
>standardIcon(QStyle::SP_MediaVolume));
    ui->toolButtonToMain->setIcon(ui->centralwidget->style()-
>standardIcon(QStyle::SP_TitleBarMinButton));


    connect(mGstDriver, SIGNAL(positionChanged()), this, SLOT(onPositionChanged()));
    connect(mGstDriver, SIGNAL(stateChanged()), this, SLOT(onStateChanged()));

    onStateChanged();

    //    ui->toolButtonPlayPause->setIcon(ui->centralwidget->style()-
>standardIcon(QStyle::SP_MediaPlay));
    //    ui->toolButtonPlayPause->setIconSize(QSize(32,32));

    mPlay.addFile(":/ui/toolButton/icon/Resources/ui/toolButton/icon/play.png");
    mPause.addFile(":/ui/toolButton/icon/Resources/ui/toolButton/icon/pause.png");
    mVolume.addFile(":/ui/toolButton/icon/Resources/ui/toolButton/icon/volume.png");
    mVolumeMuted.addFile(":/ui/toolButton/icon/Resources/ui/toolButton/icon/
volumeMuted.png");
    ui->toolButtonPlayPause->setIcon(mPlay);

    connect(ui->toolButtonPlayPause,SIGNAL(clicked()),this,SLOT(setPlayPause_clicked()));
    connect(this,SIGNAL(setPlayState()),mGstDriver,SLOT(play()));
    connect(this,SIGNAL(setPauseState()),mGstDriver,SLOT(pause()));


    connect(ui->toolButtonNext,SIGNAL(clicked()),this,SLOT(setNext_clicked()));
    connect(ui->toolButtonPrevious,SIGNAL(clicked()),this,SLOT(setPrevious_clicked()));


    ui->horizontalSliderMediaPosition->setTracking(false);
    connect(ui-
>horizontalSliderMediaPosition,SIGNAL(valueChanged(int)),this,SLOT(positionSliderMoved(in
t)));

    ui->horizontalSliderVolume->setMaximum(10);
    connect(ui-
>horizontalSliderVolume,SIGNAL(valueChanged(int)),this,SLOT(setVolume(int)));
    ui->horizontalSliderVolume->setValue(5);
    ui->horizontalSliderVolume->setSliderPosition(5);

    connect(ui->toolButtonFullScreen,SIGNAL(clicked()),this,SLOT(toFullScreen()));
    connect(ui->toolButtonVolume,SIGNAL(clicked()),this,SLOT(toggleMute()));
    connect(ui->toolButtonToMain,SIGNAL(clicked()),this,SLOT(toLibraryMode()));


    //    setMouseTracking(true);
    //    mGstDriver->setMouseTracking(true);
    //    parent->setMouseTracking(true);
    mShowControlsTimer.setSingleShot(true);
    connect(&mShowControlsTimer, SIGNAL(timeout()), this, SLOT(hideControls()));
    this->installEventFilter(this);
    //    mShowControlsTimer.start(3000);
}

VideoSink::~VideoSink()
{
    delete ui;
}

void VideoSink::closeEvent(QCloseEvent * vEvent)
{
    emit(closeMain());
    vEvent->accept();
}


//slots


void VideoSink::onStateChanged()
{

    QGst::State vNewState = mGstDriver->getState();
    //    ui->toolButtonPlayPause->setEnabled(vNewState != QGst::StatePlaying);
    if(vNewState == QGst::StateNull || vNewState == QGst::StatePaused)
    {
        ui->toolButtonPlayPause->setIcon(mPlay);
        //        ui->toolButtonPlayPause->setIconSize(QSize(32,32));
        isPlaying=false;
    }
    else if(vNewState == QGst::StatePlaying)
    {
        mGstDriver->setAutoFillBackground(true);
        ui->toolButtonPlayPause->setIcon(mPause);
        //        ui->toolButtonPlayPause->setIconSize(QSize(32,32));
        mTempTime = 0;
        mTempTime = mTempTime + ( mGstDriver->getDuration().hour() * 60);
        mTempTime = ( mTempTime +  mGstDriver->getDuration().minute() ) * 60;
        mTempTime = mTempTime + (mGstDriver->getDuration().second());
        ui->horizontalSliderMediaPosition->setMaximum(mTempTime);
        isPlaying=true;
    }

    if(vNewState == QGst::StateNull)
    {
        ui->horizontalSliderMediaPosition->setValue(0);
        ui->horizontalSliderMediaPosition->setSliderPosition(0);
        //        QMessageBox::information(this,"State Changed","Playing");
    }

    ui->toolButtonNext->setEnabled(vNewState != QGst::StateNull);
    ui->toolButtonPrevious->setEnabled(vNewState != QGst::StateNull);
    ui->horizontalSliderMediaPosition->setEnabled(vNewState != QGst::StateNull);
    //    ui->labelVolume->setEnabled(vNewState != QGst::StateNull);

    if(vNewState == QGst::StatePaused || vNewState == QGst::StatePlaying)
    {
        ui->labelTime->show();
        ui->labelLength->show();
        ui->labelTime->setEnabled(true);
        ui->horizontalSliderMediaPosition->setEnabled(true);
    }
    else
    {
        ui->labelTime->hide();
        ui->labelLength->hide();
    }
    //if we are in Null state, call onPositionChanged() to restore
    //the position of the slider and the text on the label
    if (vNewState == QGst::StateNull) {
        onPositionChanged();
    }
}

void VideoSink::onPositionChanged()
{

    if (mGstDriver->getState() != QGst::StateReady && mGstDriver->getState() !=
QGst::StateNull)
    {
        mPlayBacklength = mGstDriver->getDuration();
        if(mPlayBacklength.hour()==0)
        {
            ui->labelLength->setText(mPlayBacklength.toString("mm:ss"));
        }
        else
        {
            ui->labelLength->setText(mPlayBacklength.toString("HH:mm:ss"));
        }
        mPlayBackcurpos = mGstDriver->getPosition();
    }

    if(mPlayBackcurpos.hour() ==0)
    {
        ui->labelTime->setText(mPlayBackcurpos.toString("mm:ss"));
    }
    else
    {
        ui->labelTime->setText(mPlayBackcurpos.toString("HH:mm:ss"));
    }
    if(mGstDriver->getState()!= QGst::StateNull)
    {
        mTempTime = 0;
        mTempTime = mTempTime + ( mPlayBackcurpos.hour() * 60);
        mTempTime = ( mTempTime +  mPlayBackcurpos.minute() ) * 60;
        mTempTime = mTempTime + (mPlayBackcurpos.second());
        if(ui->horizontalSliderMediaPosition->isSliderDown()==false)
        {
            ui->horizontalSliderMediaPosition->setSliderPosition(mTempTime);
        }
    }
    else
    {
        ui->horizontalSliderMediaPosition->setValue(0);
        ui->horizontalSliderMediaPosition->setSliderPosition(0);
        //        QMessageBox::information(this,"State Changed","Playing");


    }

}


void VideoSink::setPlayPause_clicked()
{
    if(isPlaying==true)
    {
        isPlaying=false;
        emit(setPauseState());
    }
    else
    {
        isPlaying=true;
        emit(setPlayState());
    }

}

void VideoSink::setNext_clicked()
{
    emit(nextClicked());
}

void VideoSink::setPrevious_clicked()
{
    emit(prevClicked());
}

void VideoSink::positionSliderMoved(int vSliderValue)
{

    vSliderValue=setSliderOnClick(ui->horizontalSliderMediaPosition,vSliderValue);
    mTempTime = vSliderValue;
    int vSeconds = mTempTime % 60;
    mTempTime = mTempTime/60 ;
    int vMinutes = mTempTime % 60;
    int vHours = mTempTime/60;
    QTime vPosition(vHours,vMinutes,vSeconds);
    mGstDriver->setPosition(vPosition);
}

void VideoSink::setVolume(int vSliderValue)
{
    vSliderValue = setSliderOnClick(ui->horizontalSliderVolume,vSliderValue);

    emit(setVolumeAtMain(vSliderValue));
    ui->horizontalSliderVolume->setSliderPosition(vSliderValue);
}

int VideoSink::setSliderOnClick(QSlider * vQSlider , int vClickedPosition)
{
    Qt::MouseButtons vMouseButton = QApplication::mouseButtons();
    QPoint vMousePos = vQSlider->mapFromGlobal(QCursor::pos());
    bool isClicked = (vMouseButton &Qt::LeftButton) &&
            (vMousePos.x() >=0 && vMousePos.y() >=0 &&
             vMousePos.x() < vQSlider->size().width() &&
             vMousePos.y() < vQSlider->size().height());

    if(isClicked == true)
    {
        float vPosRatio = vMousePos.x() / (float)vQSlider->size().width();
        int vSliderRange = vQSlider->maximum()-vQSlider->minimum();
        int vSliderPositionUnderMouse = vQSlider->minimum() + vSliderRange *vPosRatio;

        if(vSliderPositionUnderMouse != vClickedPosition)
        {
            vQSlider->setValue(vSliderPositionUnderMouse);
            return vSliderPositionUnderMouse;
        }
    }
    return vClickedPosition;
}

void VideoSink::toggleMute()
{
    if(ui->toolButtonVolume->isChecked()==true)
    {
        mCurrentVolume=ui->horizontalSliderVolume->value();
        setVolume(0);
        ui->toolButtonVolume->setIcon(mVolumeMuted);
    }
    else
    {
        setVolume(mCurrentVolume);
        ui->toolButtonVolume->setIcon(mVolume);
    }
}


void VideoSink::toFullScreen()
{
    if(VideoSink::isFullScreen())
    {
        VideoSink::showNormal();
    }
    else
    {
        VideoSink::showFullScreen();
    }
}

void VideoSink::toLibraryMode()
{
    emit(setVideoMode(false));
    VideoSink::hide();
}

void VideoSink::setVolumeSlider(int vPosition)
{
    ui->horizontalSliderVolume->setSliderPosition(vPosition);
}

void VideoSink::setNowPlaying(QString vNowPlaying)
{
    ui->labelNowPlaying->setText(vNowPlaying);
    VideoSink::setWindowTitle("MediaTake :: " + vNowPlaying);
}

void VideoSink::showControls()
{
    ui->horizontalSliderMediaPosition->show();
    ui->horizontalSliderVolume->show();
    ui->labelLength->show();
    ui->labelNowPlaying->show();
    ui->labelTime->show();
    ui->toolButtonFullScreen->show();
    ui->toolButtonNext->show();
    ui->toolButtonPlayPause->show();
    ui->toolButtonPrevious->show();
    ui->toolButtonToMain->show();
    ui->toolButtonVolume->show();
    ui->widget->show();
}

bool VideoSink::eventFilter(QObject *obj, QEvent *event)
{
    if (event->type() == QEvent::HoverMove)
    {
//        QMouseEvent *mouseEvent = static_cast<QMouseEvent*>(event);
        showControls();

        mShowControlsTimer.start(3000); //re-hide controls after 3s
    }
    return false;
}

void VideoSink::hideControls()
```

```
{

    ui->horizontalSliderMediaPosition->hide();
    ui->horizontalSliderVolume->hide();
    ui->labelLength->setVisible(false);
    ui->labelNowPlaying->setVisible(false);
    ui->labelTime->setVisible(false);
    ui->toolButtonFullScreen->hide();
    ui->toolButtonNext->hide();
    ui->toolButtonPlayPause->hide();
    ui->toolButtonPrevious->hide();
    ui->toolButtonToMain->hide();
    ui->toolButtonVolume->hide();
    ui->widget->hide();
}
```

```xml
<?xml version="1.0" encoding="UTF-8"?>
<ui version="4.0">
 <class>MainWindow</class>
 <widget class="QMainWindow" name="MainWindow">
  <property name="geometry">
   <rect>
    <x>0</x>
    <y>0</y>
    <width>826</width>
    <height>446</height>
   </rect>
  </property>
  <property name="sizePolicy">
   <sizepolicy hsizetype="Maximum" vsizetype="Maximum">
    <horstretch>0</horstretch>
    <verstretch>0</verstretch>
   </sizepolicy>
  </property>
  <property name="mouseTracking">
   <bool>false</bool>
  </property>
  <property name="windowTitle">
   <string>MediaTake</string>
  </property>
  <property name="autoFillBackground">
   <bool>true</bool>
  </property>
  <property name="tabShape">
   <enum>QTabWidget::Rounded</enum>
  </property>
  <widget class="QWidget" name="centralWidget">
   <property name="sizePolicy">
    <sizepolicy hsizetype="Expanding" vsizetype="Expanding">
     <horstretch>0</horstretch>
     <verstretch>0</verstretch>
    </sizepolicy>
   </property>
   <layout class="QGridLayout" name="gridLayout_4">
    <property name="leftMargin">
     <number>0</number>
    </property>
    <property name="topMargin">
     <number>0</number>
    </property>
    <property name="rightMargin">
     <number>0</number>
    </property>
    <property name="bottomMargin">
     <number>0</number>
    </property>
    <item row="0" column="0">
     <layout class="QGridLayout" name="gridLayout_3">
      <item row="0" column="0">
       <widget class="QSplitter" name="splitter">
        <property name="sizePolicy">
         <sizepolicy hsizetype="Expanding" vsizetype="Expanding">
          <horstretch>0</horstretch>
          <verstretch>0</verstretch>
         </sizepolicy>
        </property>
        <property name="frameShape">
         <enum>QFrame::NoFrame</enum>
        </property>
        <property name="orientation">
         <enum>Qt::Horizontal</enum>
        </property>
        <property name="opaqueResize">
         <bool>true</bool>
        </property>
```

```
<property name="handleWidth">
 <number>1</number>
</property>
<property name="childrenCollapsible">
 <bool>false</bool>
</property>
<widget class="QTabWidget" name="tabWidgetDisplay">
 <property name="sizePolicy">
  <sizepolicy hsizetype="Maximum" vsizetype="Maximum">
   <horstretch>6</horstretch>
   <verstretch>5</verstretch>
  </sizepolicy>
 </property>
 <property name="minimumSize">
  <size>
   <width>5</width>
   <height>5</height>
  </size>
 </property>
 <property name="baseSize">
  <size>
   <width>26</width>
   <height>16</height>
  </size>
 </property>
 <property name="currentIndex">
  <number>0</number>
 </property>
 <widget class="QWidget" name="tabLibrary">
  <attribute name="title">
   <string>Library</string>
  </attribute>
  <layout class="QGridLayout" name="gridLayout_2">
   <property name="leftMargin">
    <number>1</number>
   </property>
   <property name="topMargin">
    <number>1</number>
   </property>
   <property name="rightMargin">
    <number>1</number>
   </property>
   <property name="bottomMargin">
    <number>1</number>
   </property>
   <item row="0" column="0">
    <widget class="QSplitter" name="splitter_2">
     <property name="orientation">
      <enum>Qt::Horizontal</enum>
     </property>
     <property name="handleWidth">
      <number>1</number>
     </property>
     <property name="childrenCollapsible">
      <bool>false</bool>
     </property>
     <widget class="QTreeWidget" name="treeWidgetCatergoryChooser">
      <property name="sizePolicy">
       <sizepolicy hsizetype="Minimum" vsizetype="Expanding">
        <horstretch>0</horstretch>
        <verstretch>0</verstretch>
       </sizepolicy>
      </property>
      <property name="minimumSize">
       <size>
        <width>135</width>
        <height>0</height>
       </size>
      </property>
      <property name="maximumSize">
       <size>
        <width>150</width>
        <height>16777215</height>
       </size>
      </property>
      <property name="headerHidden">
       <bool>true</bool>
      </property>
      <column>
       <property name="text">
        <string/>
       </property>
      </column>
      <item>
       <property name="text">
        <string>Audio</string>
       </property>
       <item>
        <property name="text">
         <string>Folders</string>
        </property>
       </item>
       <item>
        <property name="text">
         <string>Artist</string>
        </property>
       </item>
       <item>
        <property name="text">
         <string>Album</string>
        </property>
       </item>
       <item>
        <property name="text">
         <string>Genre</string>
        </property>
       </item>
       <item>
        <property name="text">
         <string>Year</string>
        </property>
       </item>
      </item>
      <item>
       <property name="text">
        <string>Video</string>
       </property>
       <item>
        <property name="text">
         <string>Folders</string>
        </property>
       </item>
      </item>
      <item>
       <property name="text">
        <string>Playlist</string>
       </property>
      </item>
     </widget>
     <widget class="QTreeWidget" name="treeWidgetLibraryDisplay">
      <property name="sizePolicy">
       <sizepolicy hsizetype="Expanding" vsizetype="Expanding">
        <horstretch>0</horstretch>
        <verstretch>0</verstretch>
       </sizepolicy>
      </property>
      <property name="maximumSize">
       <size>
       <width>16777215</width>
       <height>16777215</height>
      </size>
     </property>
     <property name="contextMenuPolicy">
      <enum>Qt::ActionsContextMenu</enum>
     </property>
     <property name="selectionMode">
      <enum>QAbstractItemView::ExtendedSelection</enum>
     </property>
     <property name="selectionBehavior">
      <enum>QAbstractItemView::SelectRows</enum>
     </property>
     <column>
      <property name="text">
       <string/>
      </property>
     </column>
     <column>
      <property name="text">
       <string>#</string>
      </property>
     </column>
     <column>
      <property name="text">
       <string>Album</string>
      </property>
     </column>
     <column>
      <property name="text">
       <string>Title</string>
      </property>
     </column>
     <column>
      <property name="text">
       <string>Length</string>
      </property>
     </column>
     <column>
      <property name="text">
       <string>Artist</string>
      </property>
     </column>
     <column>
      <property name="text">
       <string>Genre</string>
      </property>
     </column>
     <column>
      <property name="text">
       <string>Year</string>
      </property>
     </column>
     <column>
      <property name="text">
       <string>BitRate</string>
      </property>
     </column>
     <column>
      <property name="text">
       <string>Composer</string>
      </property>
     </column>
     <column>
      <property name="text">
       <string>Path</string>
      </property>
     </column>
    </widget>
```

```
             </widget>
            </item>
           </layout>
          </widget>
          <widget class="QWidget" name="tabManager">
           <attribute name="title">
            <string>Manager</string>
           </attribute>
           <layout class="QGridLayout" name="gridLayout">
            <property name="leftMargin">
             <number>0</number>
            </property>
            <property name="topMargin">
             <number>0</number>
            </property>
            <property name="rightMargin">
             <number>0</number>
            </property>
            <property name="bottomMargin">
             <number>0</number>
            </property>
            <property name="spacing">
             <number>0</number>
            </property>
            <item row="0" column="0">
             <layout class="QHBoxLayout" name="horizontalLayout_8">
              <item>
               <spacer name="horizontalSpacer_3">
                <property name="orientation">
                 <enum>Qt::Horizontal</enum>
                </property>
                <property name="sizeHint" stdset="0">
                 <size>
                  <width>40</width>
                  <height>20</height>
                 </size>
                </property>
               </spacer>
              </item>
              <item>
               <widget class="QPushButton" name="pushButtonUpdateLibrary">
                <property name="text">
                 <string>UpdateLibrary</string>
                </property>
               </widget>
              </item>
             </layout>
            </item>
            <item row="1" column="0">
             <layout class="QVBoxLayout" name="verticalLayout_4">
              <item>
               <widget class="QLabel" name="labelAudioLibrary">
                <property name="text">
                 <string>AudioLibrary</string>
                </property>
               </widget>
              </item>
              <item>
               <layout class="QHBoxLayout" name="horizontalLayout_2">
                <item>
                 <widget class="QListWidget" name="listWidgetAudioLibrary"/>
                </item>
                <item>
                 <layout class="QVBoxLayout" name="verticalLayout_2">
                  <item>
                   <widget class="QPushButton" name="pushButtonAddAudio">
                    <property name="text">
                     <string>&lt;&lt; Add Folder</string>
                    </property>
                   </widget>
                  </item>
                  <item>
                   <widget class="QPushButton" name="pushButtonRemoveAudio">
                    <property name="text">
                     <string>&gt;&gt; Remove Folder</string>
                    </property>
                   </widget>
                  </item>
                  <item>
                   <spacer name="verticalSpacer">
                    <property name="orientation">
                     <enum>Qt::Vertical</enum>
                    </property>
                    <property name="sizeHint" stdset="0">
                     <size>
                      <width>20</width>
                      <height>40</height>
                     </size>
                    </property>
                   </spacer>
                  </item>
                 </layout>
                </item>
               </layout>
              </item>
             </layout>
            </item>
            <item row="2" column="0">
             <layout class="QVBoxLayout" name="verticalLayout_5">
              <item>
               <widget class="QLabel" name="labelVideoLibrary">
                <property name="text">
                 <string>Video Library</string>
                </property>
               </widget>
              </item>
              <item>
               <layout class="QHBoxLayout" name="horizontalLayout_3">
                <item>
                 <widget class="QListWidget" name="listWidgetVideoLibrary"/>
                </item>
                <item>
                 <layout class="QVBoxLayout" name="verticalLayout_3">
                  <item>
                   <widget class="QPushButton" name="pushButtonAddVideo">
                    <property name="text">
                     <string>&lt;&lt; Add Folder</string>
                    </property>
                   </widget>
                  </item>
                  <item>
                   <widget class="QPushButton" name="pushButtonRemoveVideo">
                    <property name="text">
                     <string>&gt;&gt; Remove Folder</string>
                    </property>
                   </widget>
                  </item>
                  <item>
                   <spacer name="verticalSpacer_2">
                    <property name="orientation">
                     <enum>Qt::Vertical</enum>
                    </property>
                    <property name="sizeHint" stdset="0">
                     <size>
                      <width>20</width>
                      <height>40</height>
                     </size>
                    </property>
```

```
          </spacer>
         </item>
        </layout>
       </item>
      </layout>
     </item>
    </layout>
   </item>
  </layout>
 </widget>
</widget>
<widget class="QWidget" name="layoutWidget">
 <layout class="QVBoxLayout" name="verticalLayout">
  <property name="leftMargin">
   <number>5</number>
  </property>
  <item>
   <layout class="QHBoxLayout" name="horizontalLayout_7">
    <item>
     <widget class="QLabel" name="labelPlayQueue">
      <property name="text">
       <string>Play Queue</string>
      </property>
     </widget>
    </item>
    <item>
     <widget class="QToolButton" name="toolButtonClearQueue">
      <property name="text">
       <string>ClearList</string>
      </property>
     </widget>
    </item>
   </layout>
  </item>
  <item>
   <widget class="QComboBox" name="comboBox">
    <item>
     <property name="text">
      <string>Now Playing</string>
     </property>
    </item>
   </widget>
  </item>
  <item>
   <widget class="QTreeWidget" name="treeWidgetQueue">
    <property name="contextMenuPolicy">
     <enum>Qt::ActionsContextMenu</enum>
    </property>
    <property name="rootIsDecorated">
     <bool>false</bool>
    </property>
    <property name="columnCount">
     <number>2</number>
    </property>
    <attribute name="headerVisible">
     <bool>false</bool>
    </attribute>
    <column>
     <property name="text">
      <string notr="true">1</string>
     </property>
    </column>
    <column>
     <property name="text">
      <string notr="true">2</string>
     </property>
    </column>
   </widget>
  </item>
```

```
          </layout>
         </widget>
        </widget>
       </item>
       <item row="1" column="0">
        <widget class="QWidget" name="widgetPlayControls" native="true">
         <layout class="QVBoxLayout" name="verticalLayout_8">
          <property name="spacing">
           <number>0</number>
          </property>
          <property name="leftMargin">
           <number>1</number>
          </property>
          <property name="topMargin">
           <number>1</number>
          </property>
          <property name="rightMargin">
           <number>1</number>
          </property>
          <property name="bottomMargin">
           <number>5</number>
          </property>
          <item>
           <layout class="QVBoxLayout" name="verticalLayout_6">
            <item>
             <widget class="QSlider" name="horizontalSliderMediaPosition">
              <property name="orientation">
               <enum>Qt::Horizontal</enum>
              </property>
             </widget>
            </item>
            <item>
             <layout class="QHBoxLayout" name="horizontalLayout_6">
              <property name="spacing">
               <number>0</number>
              </property>
              <item>
               <layout class="QHBoxLayout" name="horizontalLayout_5">
                <property name="spacing">
                 <number>8</number>
                </property>
                <property name="leftMargin">
                 <number>3</number>
                </property>
                <property name="rightMargin">
                 <number>0</number>
                </property>
                <item>
                 <layout class="QVBoxLayout" name="verticalLayout_9">
                  <property name="spacing">
                   <number>0</number>
                  </property>
                  <item>
                   <widget class="QToolButton" name="toolButtonFullScreen">
                    <property name="maximumSize">
                     <size>
                      <width>20</width>
                      <height>20</height>
                     </size>
                    </property>
                    <property name="toolTip">
                     <string>To Full Screen</string>
                    </property>
                    <property name="text">
                     <string>...</string>
                    </property>
                    <property name="iconSize">
                     <size>
                      <width>20</width>
                      <height>20</height>
                     </size>
                    </property>
                   </widget>
                  </item>
                  <item>
                   <widget class="QToolButton" name="toolButtonToScreen">
                    <property name="maximumSize">
                     <size>
                      <width>20</width>
                      <height>20</height>
                     </size>
                    </property>
                    <property name="toolTip">
                     <string>Switch To Video Mode</string>
                    </property>
                    <property name="text">
                     <string>...</string>
                    </property>
                    <property name="iconSize">
                     <size>
                      <width>20</width>
                      <height>20</height>
                     </size>
                    </property>
                   </widget>
                  </item>
                 </layout>
                </item>
                <item>
                 <widget class="QLabel" name="labelNowPlaying">
                  <property name="sizePolicy">
                   <sizepolicy hsizetype="Fixed" vsizetype="Fixed">
                    <horstretch>0</horstretch>
                    <verstretch>0</verstretch>
                   </sizepolicy>
                  </property>
                  <property name="minimumSize">
                   <size>
                    <width>150</width>
                    <height>37</height>
                   </size>
                  </property>
                  <property name="maximumSize">
                   <size>
                    <width>150</width>
                    <height>37</height>
                   </size>
                  </property>
                  <property name="text">
                   <string/>
                  </property>
                  <property name="wordWrap">
                   <bool>true</bool>
                  </property>
                 </widget>
                </item>
                <item>
                 <spacer name="horizontalSpacer">
                  <property name="orientation">
                   <enum>Qt::Horizontal</enum>
                  </property>
                  <property name="sizeType">
                   <enum>QSizePolicy::Expanding</enum>
                  </property>
                  <property name="sizeHint" stdset="0">
                   <size>
                    <width>120</width>
                    <height>20</height>
                  </size>
                 </property>
                </spacer>
               </item>
              </layout>
             </item>
             <item>
              <layout class="QHBoxLayout" name="horizontalLayout">
               <property name="spacing">
                <number>6</number>
               </property>
               <property name="sizeConstraint">
                <enum>QLayout::SetDefaultConstraint</enum>
               </property>
               <item>
                <widget class="QLabel" name="labelTime">
                 <property name="sizePolicy">
                  <sizepolicy hsizetype="Minimum" vsizetype="Minimum">
                   <horstretch>0</horstretch>
                   <verstretch>0</verstretch>
                  </sizepolicy>
                 </property>
                 <property name="minimumSize">
                  <size>
                   <width>0</width>
                   <height>0</height>
                  </size>
                 </property>
                 <property name="text">
                  <string/>
                 </property>
                </widget>
               </item>
               <item>
                <widget class="QToolButton" name="toolButtonPrevious">
                 <property name="minimumSize">
                  <size>
                   <width>32</width>
                   <height>32</height>
                  </size>
                 </property>
                 <property name="maximumSize">
                  <size>
                   <width>32</width>
                   <height>32</height>
                  </size>
                 </property>
                 <property name="text">
                  <string>...</string>
                 </property>
                 <property name="icon">
                  <iconset resource="UiResources.qrc">
                   <normaloff>:/ui/toolButton/icon/Resources/ui/toolButton/icon/
previous.png</normaloff>:/ui/toolButton/icon/Resources/ui/toolButton/icon/previous.png</
iconset>
                 </property>
                 <property name="iconSize">
                  <size>
                   <width>16</width>
                   <height>16</height>
                  </size>
                 </property>
                </widget>
               </item>
               <item>
                <widget class="QToolButton" name="toolButtonPlayPause">
                 <property name="sizePolicy">
                  <sizepolicy hsizetype="Fixed" vsizetype="Fixed">
                   <horstretch>0</horstretch>
```

```
                    <verstretch>0</verstretch>
                   </sizepolicy>
                  </property>
                  <property name="minimumSize">
                   <size>
                    <width>55</width>
                    <height>45</height>
                   </size>
                  </property>
                  <property name="text">
                   <string>...</string>
                  </property>
                  <property name="icon">
                   <iconset resource="UiResources.qrc">
                    <normaloff>:/ui/toolButton/icon/Resources/ui/toolButton/icon/
play.png</normaloff>:/ui/toolButton/icon/Resources/ui/toolButton/icon/play.png</iconset>
                  </property>
                  <property name="iconSize">
                   <size>
                    <width>20</width>
                    <height>20</height>
                   </size>
                  </property>
                  <property name="shortcut">
                   <string>Space</string>
                  </property>
                 </widget>
                </item>
                <item>
                 <widget class="QToolButton" name="toolButtonNext">
                  <property name="minimumSize">
                   <size>
                    <width>32</width>
                    <height>32</height>
                   </size>
                  </property>
                  <property name="maximumSize">
                   <size>
                    <width>32</width>
                    <height>32</height>
                   </size>
                  </property>
                  <property name="text">
                   <string>...</string>
                  </property>
                  <property name="icon">
                   <iconset resource="UiResources.qrc">
                    <normaloff>:/ui/toolButton/icon/Resources/ui/toolButton/icon/
next.png</normaloff>:/ui/toolButton/icon/Resources/ui/toolButton/icon/next.png</iconset>
                  </property>
                  <property name="iconSize">
                   <size>
                    <width>16</width>
                    <height>16</height>
                   </size>
                  </property>
                 </widget>
                </item>
                <item>
                 <widget class="QLabel" name="labelLength">
                  <property name="text">
                   <string/>
                  </property>
                 </widget>
                </item>
               </layout>
              </item>
              <item>
               <layout class="QHBoxLayout" name="horizontalLayout_4">
                <property name="spacing">
                 <number>0</number>
                </property>
                <property name="rightMargin">
                 <number>4</number>
                </property>
                <item>
                 <spacer name="horizontalSpacer_2">
                  <property name="orientation">
                   <enum>Qt::Horizontal</enum>
                  </property>
                  <property name="sizeType">
                   <enum>QSizePolicy::Expanding</enum>
                  </property>
                  <property name="sizeHint" stdset="0">
                   <size>
                    <width>158</width>
                    <height>20</height>
                   </size>
                  </property>
                 </spacer>
                </item>
                <item>
                 <widget class="QToolButton" name="toolButtonVolume">
                  <property name="minimumSize">
                   <size>
                    <width>30</width>
                    <height>30</height>
                   </size>
                  </property>
                  <property name="maximumSize">
                   <size>
                    <width>30</width>
                    <height>30</height>
                   </size>
                  </property>
                  <property name="text">
                   <string>...</string>
                  </property>
                  <property name="icon">
                   <iconset resource="UiResources.qrc">
                    <normaloff>:/ui/toolButton/icon/Resources/ui/toolButton/icon/
 volume.png</normaloff>:/ui/toolButton/icon/Resources/ui/toolButton/icon/volume.png</
 iconset>
                  </property>
                  <property name="iconSize">
                   <size>
                    <width>15</width>
                    <height>15</height>
                   </size>
                  </property>
                  <property name="checkable">
                   <bool>true</bool>
                  </property>
                 </widget>
                </item>
                <item>
                 <widget class="QSlider" name="horizontalSliderVolume">
                  <property name="sizePolicy">
                   <sizepolicy hsizetype="Fixed" vsizetype="Fixed">
                    <horstretch>0</horstretch>
                    <verstretch>0</verstretch>
                   </sizepolicy>
                  </property>
                  <property name="minimumSize">
                   <size>
                    <width>7</width>
                    <height>0</height>
                   </size>
                  </property>
                  <property name="orientation">
                   <enum>Qt::Horizontal</enum>
                  </property>
                 </widget>
                </item>
               </layout>
              </item>
             </layout>
            </item>
           </layout>
          </item>
         </layout>
        </widget>
       </item>
      </layout>
     </item>
    </layout>
   </widget>
   <action name="actionAddtoQueue">
    <property name="text">
     <string>AddtoQueue</string>
    </property>
   </action>
   <action name="actionRemoveFromQueue">
    <property name="text">
     <string>removeFromQueue</string>
    </property>
    <property name="shortcut">
     <string>Del</string>
    </property>
   </action>
  </widget>
  <layoutdefault spacing="6" margin="11"/>
  <tabstops>
   <tabstop>toolButtonPlayPause</tabstop>
   <tabstop>toolButtonNext</tabstop>
   <tabstop>toolButtonPrevious</tabstop>
   <tabstop>horizontalSliderMediaPosition</tabstop>
   <tabstop>toolButtonVolume</tabstop>
   <tabstop>horizontalSliderVolume</tabstop>
   <tabstop>toolButtonToScreen</tabstop>
   <tabstop>toolButtonFullScreen</tabstop>
   <tabstop>pushButtonUpdateLibrary</tabstop>
   <tabstop>listWidgetAudioLibrary</tabstop>
   <tabstop>pushButtonAddAudio</tabstop>
   <tabstop>pushButtonRemoveAudio</tabstop>
   <tabstop>listWidgetVideoLibrary</tabstop>
   <tabstop>pushButtonAddVideo</tabstop>
   <tabstop>pushButtonRemoveVideo</tabstop>
   <tabstop>toolButtonClearQueue</tabstop>
   <tabstop>comboBox</tabstop>
   <tabstop>treeWidgetQueue</tabstop>
   <tabstop>treeWidgetLibraryDisplay</tabstop>
   <tabstop>tabWidgetDisplay</tabstop>
   <tabstop>treeWidgetCatergoryChooser</tabstop>
  </tabstops>
  <resources>
   <include location="UiResources.qrc"/>
  </resources>
  <connections/>
 </ui>
```

```xml
<?xml version="1.0" encoding="UTF-8"?>
<ui version="4.0">
 <class>SelectFileDialog</class>
 <widget class="QDialog" name="SelectFileDialog">
  <property name="geometry">
   <rect>
    <x>0</x>
    <y>0</y>
    <width>617</width>
    <height>430</height>
   </rect>
  </property>
  <property name="sizePolicy">
   <sizepolicy hsizetype="Expanding" vsizetype="Expanding">
    <horstretch>0</horstretch>
    <verstretch>0</verstretch>
   </sizepolicy>
  </property>
  <property name="baseSize">
   <size>
    <width>480</width>
    <height>640</height>
   </size>
  </property>
  <property name="windowTitle">
   <string>Select File</string>
  </property>
  <layout class="QGridLayout" name="gridLayout">
   <item row="0" column="0">
    <layout class="QVBoxLayout" name="verticalLayout">
     <item>
      <widget class="QTreeView" name="treeViewFolderExplorer"/>
     </item>
     <item>
      <layout class="QHBoxLayout" name="horizontalLayout">
       <item>
        <widget class="QLineEdit" name="lineEditFilePath"/>
       </item>
       <item>
        <widget class="QPushButton" name="pushButtonSelect">
         <property name="text">
          <string>Select</string>
         </property>
        </widget>
       </item>
       <item>
        <widget class="QPushButton" name="pushButtonClose">
         <property name="text">
          <string>Close</string>
         </property>
        </widget>
       </item>
      </layout>
     </item>
    </layout>
   </item>
  </layout>
 </widget>
 <resources/>
 <connections/>
</ui>
```

```xml
<?xml version="1.0" encoding="UTF-8"?>
<ui version="4.0">
 <class>VideoSink</class>
 <widget class="QMainWindow" name="VideoSink">
  <property name="windowModality">
   <enum>Qt::NonModal</enum>
  </property>
  <property name="geometry">
   <rect>
    <x>0</x>
    <y>0</y>
    <width>640</width>
    <height>480</height>
   </rect>
  </property>
  <property name="sizePolicy">
   <sizepolicy hsizetype="Minimum" vsizetype="Minimum">
    <horstretch>0</horstretch>
    <verstretch>3</verstretch>
   </sizepolicy>
  </property>
  <property name="minimumSize">
   <size>
    <width>100</width>
    <height>100</height>
   </size>
  </property>
  <property name="mouseTracking">
   <bool>false</bool>
  </property>
  <property name="windowTitle">
   <string>MediaTake</string>
  </property>
  <widget class="QWidget" name="centralwidget">
   <layout class="QGridLayout" name="gridLayout_3">
    <property name="leftMargin">
     <number>0</number>
    </property>
    <property name="topMargin">
     <number>0</number>
    </property>
    <property name="rightMargin">
     <number>0</number>
    </property>
    <property name="bottomMargin">
     <number>0</number>
    </property>
    <property name="spacing">
     <number>0</number>
    </property>
    <item row="0" column="0">
     <layout class="QVBoxLayout" name="verticalLayout_3">
      <property name="spacing">
       <number>0</number>
      </property>
      <item>
       <widget class="QWidget" name="widget_2" native="true">
        <property name="sizePolicy">
         <sizepolicy hsizetype="Expanding" vsizetype="Expanding">
          <horstretch>0</horstretch>
          <verstretch>0</verstretch>
         </sizepolicy>
        </property>
        <layout class="QGridLayout" name="gridLayout_2">
         <property name="leftMargin">
          <number>0</number>
         </property>
         <property name="topMargin">
          <number>0</number>
         </property>
         <property name="rightMargin">
          <number>0</number>
         </property>
         <property name="bottomMargin">
          <number>0</number>
         </property>
         <property name="spacing">
          <number>0</number>
         </property>
         <item row="0" column="0">
          <layout class="QVBoxLayout" name="verticalLayoutVideoLayout">
           <property name="spacing">
            <number>0</number>
           </property>
           <property name="sizeConstraint">
            <enum>QLayout::SetMaximumSize</enum>
           </property>
          </layout>
         </item>
        </layout>
       </widget>
      </item>
      <item>
       <widget class="QWidget" name="widget" native="true">
        <layout class="QGridLayout" name="gridLayout">
         <property name="leftMargin">
          <number>0</number>
         </property>
         <property name="topMargin">
          <number>0</number>
         </property>
         <property name="rightMargin">
          <number>0</number>
         </property>
         <property name="bottomMargin">
          <number>0</number>
         </property>
         <property name="spacing">
          <number>0</number>
         </property>
         <item row="0" column="0">
          <layout class="QVBoxLayout" name="verticalLayout_2">
           <property name="leftMargin">
            <number>4</number>
           </property>
           <property name="topMargin">
            <number>4</number>
           </property>
           <property name="rightMargin">
            <number>1</number>
           </property>
           <property name="bottomMargin">
            <number>5</number>
           </property>
           <item>
            <widget class="QSlider" name="horizontalSliderMediaPosition">
             <property name="orientation">
              <enum>Qt::Horizontal</enum>
             </property>
            </widget>
           </item>
           <item>
            <layout class="QHBoxLayout" name="horizontalLayout_5">
             <item>
              <layout class="QHBoxLayout" name="horizontalLayout_3">
               <property name="spacing">
                <number>8</number>
               </property>
```

```
<property name="leftMargin">
 <number>4</number>
</property>
<item>
 <layout class="QVBoxLayout" name="verticalLayout">
  <property name="spacing">
   <number>0</number>
  </property>
  <item>
   <widget class="QToolButton" name="toolButtonFullScreen">
    <property name="maximumSize">
     <size>
      <width>20</width>
      <height>20</height>
     </size>
    </property>
    <property name="toolTip">
     <string>To Full Screen</string>
    </property>
    <property name="text">
     <string>...</string>
    </property>
    <property name="iconSize">
     <size>
      <width>20</width>
      <height>20</height>
     </size>
    </property>
   </widget>
  </item>
  <item>
   <widget class="QToolButton" name="toolButtonToMain">
    <property name="maximumSize">
     <size>
      <width>20</width>
      <height>20</height>
     </size>
    </property>
    <property name="toolTip">
     <string>Switch To Library</string>
    </property>
    <property name="text">
     <string>...</string>
    </property>
    <property name="iconSize">
     <size>
      <width>20</width>
      <height>20</height>
     </size>
    </property>
   </widget>
  </item>
 </layout>
</item>
<item>
 <widget class="QLabel" name="labelNowPlaying">
  <property name="sizePolicy">
   <sizepolicy hsizetype="Fixed" vsizetype="Fixed">
    <horstretch>0</horstretch>
    <verstretch>0</verstretch>
   </sizepolicy>
  </property>
  <property name="minimumSize">
   <size>
    <width>150</width>
    <height>32</height>
   </size>
  </property>
  <property name="maximumSize">
```

```
                  <size>
                   <width>150</width>
                   <height>32</height>
                  </size>
                 </property>
                 <property name="text">
                  <string/>
                 </property>
                 <property name="wordWrap">
                  <bool>true</bool>
                 </property>
                </widget>
               </item>
               <item>
                <spacer name="horizontalSpacer">
                 <property name="orientation">
                  <enum>Qt::Horizontal</enum>
                 </property>
                 <property name="sizeHint" stdset="0">
                  <size>
                   <width>40</width>
                   <height>20</height>
                  </size>
                 </property>
                </spacer>
               </item>
              </layout>
             </item>
             <item>
              <layout class="QHBoxLayout" name="horizontalLayout_2">
               <property name="spacing">
                <number>0</number>
               </property>
               <item>
                <widget class="QLabel" name="labelTime">
                 <property name="text">
                  <string/>
                 </property>
                </widget>
               </item>
               <item>
                <layout class="QHBoxLayout" name="horizontalLayout">
                 <item>
                  <widget class="QToolButton" name="toolButtonPrevious">
                   <property name="minimumSize">
                    <size>
                     <width>32</width>
                     <height>32</height>
                    </size>
                   </property>
                   <property name="maximumSize">
                    <size>
                     <width>32</width>
                     <height>32</height>
                    </size>
                   </property>
                   <property name="text">
                    <string>...</string>
                   </property>
                   <property name="icon">
                    <iconset resource="UiResources.qrc">
                     <normaloff>:/ui/toolButton/icon/Resources/ui/toolButton/icon/
previous.png</normaloff>:/ui/toolButton/icon/Resources/ui/toolButton/icon/previous.png</
iconset>
                   </property>
                   <property name="iconSize">
                    <size>
                     <width>16</width>
                     <height>16</height>
                    </size>
                   </property>
                  </widget>
                 </item>
                 <item>
                  <widget class="QToolButton" name="toolButtonPlayPause">
                   <property name="sizePolicy">
                    <sizepolicy hsizetype="Fixed" vsizetype="Fixed">
                     <horstretch>0</horstretch>
                     <verstretch>0</verstretch>
                    </sizepolicy>
                   </property>
                   <property name="minimumSize">
                    <size>
                     <width>55</width>
                     <height>45</height>
                    </size>
                   </property>
                   <property name="maximumSize">
                    <size>
                     <width>55</width>
                     <height>45</height>
                    </size>
                   </property>
                   <property name="text">
                    <string>...</string>
                   </property>
                   <property name="icon">
                    <iconset resource="UiResources.qrc">
                     <normaloff>:/ui/toolButton/icon/Resources/ui/toolButton/icon/
play.png</normaloff>:/ui/toolButton/icon/Resources/ui/toolButton/icon/play.png</iconset>
                   </property>
                   <property name="iconSize">
                    <size>
                     <width>20</width>
                     <height>20</height>
                    </size>
                   </property>
                   <property name="shortcut">
                    <string>Space</string>
                   </property>
                  </widget>
                 </item>
                 <item>
                  <widget class="QToolButton" name="toolButtonNext">
                   <property name="minimumSize">
                    <size>
                     <width>32</width>
                     <height>32</height>
                    </size>
                   </property>
                   <property name="maximumSize">
                    <size>
                     <width>32</width>
                     <height>32</height>
                    </size>
                   </property>
                   <property name="text">
                    <string>...</string>
                   </property>
                   <property name="icon">
                    <iconset resource="UiResources.qrc">
                     <normaloff>:/ui/toolButton/icon/Resources/ui/toolButton/icon/
next.png</normaloff>:/ui/toolButton/icon/Resources/ui/toolButton/icon/next.png</iconset>
                   </property>
                   <property name="iconSize">
                    <size>
                     <width>16</width>
                     <height>16</height>
```

```xml
                </size>
               </property>
              </widget>
             </item>
            </layout>
           </item>
           <item>
            <widget class="QLabel" name="labelLength">
             <property name="text">
              <string/>
             </property>
            </widget>
           </item>
          </layout>
         </item>
         <item>
          <layout class="QHBoxLayout" name="horizontalLayout_4">
           <property name="spacing">
            <number>1</number>
           </property>
           <item>
            <spacer name="horizontalSpacer_2">
             <property name="orientation">
              <enum>Qt::Horizontal</enum>
             </property>
             <property name="sizeHint" stdset="0">
              <size>
               <width>101</width>
               <height>20</height>
              </size>
             </property>
            </spacer>
           </item>
           <item>
            <widget class="QToolButton" name="toolButtonVolume">
             <property name="minimumSize">
              <size>
               <width>30</width>
               <height>30</height>
              </size>
             </property>
             <property name="maximumSize">
              <size>
               <width>30</width>
               <height>30</height>
              </size>
             </property>
             <property name="text">
              <string>...</string>
             </property>
             <property name="icon">
              <iconset resource="UiResources.qrc">
               <normaloff>:/ui/toolButton/icon/Resources/ui/toolButton/icon/
volume.png</normaloff>:/ui/toolButton/icon/Resources/ui/toolButton/icon/volume.png</
iconset>
             </property>
             <property name="iconSize">
              <size>
               <width>15</width>
               <height>15</height>
              </size>
             </property>
             <property name="checkable">
              <bool>true</bool>
             </property>
             <property name="checked">
              <bool>false</bool>
             </property>
            </widget>
```

```
               </item>
               <item>
                <widget class="QSlider" name="horizontalSliderVolume">
                 <property name="sizePolicy">
                  <sizepolicy hsizetype="Fixed" vsizetype="Fixed">
                   <horstretch>0</horstretch>
                   <verstretch>0</verstretch>
                  </sizepolicy>
                 </property>
                 <property name="minimumSize">
                  <size>
                   <width>7</width>
                   <height>0</height>
                  </size>
                 </property>
                 <property name="orientation">
                  <enum>Qt::Horizontal</enum>
                 </property>
                </widget>
               </item>
              </layout>
             </item>
            </layout>
           </item>
          </layout>
         </item>
        </layout>
       </widget>
      </item>
     </layout>
    </item>
   </layout>
  </widget>
 </widget>
 <tabstops>
  <tabstop>toolButtonPlayPause</tabstop>
  <tabstop>toolButtonNext</tabstop>
  <tabstop>toolButtonPrevious</tabstop>
  <tabstop>horizontalSliderMediaPosition</tabstop>
  <tabstop>toolButtonVolume</tabstop>
  <tabstop>horizontalSliderVolume</tabstop>
  <tabstop>toolButtonFullScreen</tabstop>
  <tabstop>toolButtonToMain</tabstop>
 </tabstops>
 <resources>
  <include location="UiResources.qrc"/>
 </resources>
 <connections/>
</ui>
```

```
<RCC>
    <qresource prefix="/ui/toolButton/icon">
        <file>Resources/ui/toolButton/icon/next.png</file>
        <file>Resources/ui/toolButton/icon/pause.png</file>
        <file>Resources/ui/toolButton/icon/play.png</file>
        <file>Resources/ui/toolButton/icon/previous.png</file>
        <file>Resources/ui/toolButton/icon/repeat.png</file>
        <file>Resources/ui/toolButton/icon/shuffle.png</file>
        <file>Resources/ui/toolButton/icon/volume.png</file>
        <file>Resources/ui/toolButton/icon/volumeMuted.png</file>
    </qresource>
</RCC>
```