Documento Técnico

# Reproductor Multimedia - CodeVibes


## Desarrolladores:

Victor Manuel López Henao

Jhon Mario Aguirre Correa

Juan Esteban Valencia Valdés


November 15, 2024

**Código y Explicación del archivo `QMediaPlayer.pro`:**

```
QT += core gui multimedia multimediawidgets
```

**Explicación:** - Esta línea agrega los módulos necesarios para el proyecto de Qt. - `core`: Es el módulo base de Qt, que proporciona las funcionalidades fundamentales como estructuras de datos, archivos, hilos, etc. - `gui`: Añade funcionalidades para la interfaz gráfica, como widgets, eventos y manejo de gráficos. - `multimedia`: Incluye soporte para audio y video, esencial para proyectos multimedia. - `multimediawidgets`: Proporciona widgets como `QVideoWidget` para la visualización de video.

```
greaterThan(QT_MAJOR_VERSION, 4): QT += widgets
```

**Explicación:** - Esta línea asegura que si la versión de Qt es mayor a la 4, se incluya también el módulo `widgets`. - El módulo `widgets` permite trabajar con interfaces gráficas utilizando controles y vistas de interfaz (como botones, listas, ventanas, etc.).

```
CONFIG += c++17
```

**Explicación:** - Esta línea especifica que el proyecto debe compilarse con el estándar de C++17. - Esto permite usar las nuevas características de C++17 en el proyecto, como las mejoras en los contenedores, el manejo de archivos y otros avances del lenguaje.

```
#DEFINES += QT_DISABLE_DEPRECATED_BEFORE=0x060000
```

**Explicación:** - Esta línea está comentada, lo que significa que no está activa por defecto. - Si se descomenta, deshabilitaría el uso de APIs obsoletas de Qt anteriores a la versión 6.0.0. Esto es útil para asegurarse de que el proyecto no use funcionalidades que puedan ser eliminadas en versiones futuras de Qt.

```
SOURCES += \
    main.cpp \
    mainwindow.cpp
```

**Explicación:** - Aquí se especifican los archivos fuente C++ que forman parte del proyecto. - `main.cpp` es el archivo que contiene la función principal del programa. - `mainwindow.cpp` contiene la implementación de la ventana principal, donde se gestionan las interacciones de la interfaz.

```
HEADERS += \
    mainwindow.h
```

**Explicación:** - Se especifica el archivo de cabecera `mainwindow.h`, que contiene las declaraciones de las clases y funciones que se implementan en `mainwindow.cpp`. - Este archivo es importante para proporcionar las interfaces de clases y funciones que serán utilizadas en otros archivos de código fuente.

```
FORMS += \
    mainwindow.ui
```

**Explicación:** - Aquí se agrega el archivo `mainwindow.ui`, que es un archivo de interfaz de usuario de Qt. - `mainwindow.ui` define la estructura de la interfaz gráfica utilizando el diseñador de Qt, lo que permite que se genere automáticamente el código correspondiente al diseño de la ventana principal.

```
qnx: target.path = /tmp/$${TARGET}/bin
```

**Explicación:** - Esta línea es una regla de instalación específica para el sistema operativo QNX, que es un sistema embebido en tiempo real. - Define la ruta de destino para los archivos compilados, en este caso, `/tmp/$`$TARGET/bin$.

```
else: unix:!android: target.path = /opt/$${TARGET}/bin
```

Explicación: - Esta regla establece la ruta de instalación para sistemas UNIX (pero no Android). Los archivos compilados se instalarán en /opt/$$TARGET/bin.− $Estoaseguraquelosarchivosseubiquenenundirectorioestándarparaaplicacionesensistemas UNIX$.

```
!isEmpty(target.path): INSTALLS += target
```

Explicación: - Esta línea verifica si target.path no está vacío y, si es cierto, agrega el archivo de destino a la lista de instalación. - Si se ha especificado una ruta de destino (target.path), se procederá a instalar el proyecto en esa ubicación.

```
RESOURCES += \
    videofondo.qrc
```

Explicación: - Aquí se especifica el archivo de recursos videofondo.qrc, que contiene información sobre los archivos de recursos (como imágenes, sonidos o videos) que el proyecto utilizará. - En este caso, parece que se está agregando un archivo de video (probablemente como fondo de la interfaz).

Código y Explicación del archivo main.cpp:

```
#include "mainwindow.h"
```

Main.cpp - Explicación: - Esta línea incluye el archivo de cabecera mainwindow.h, que contiene la declaración de la clase MainWindow. - Al incluir este archivo, se accede a las funcionalidades y elementos de la ventana principal definida en mainwindow.h.

```
#include <QApplication>
```

Main.cpp - Explicación: - Se incluye el archivo QApplication, que es una clase de Qt fundamental para cualquier aplicación gráfica. - QApplication gestiona los recursos de la aplicación y el ciclo de eventos, permitiendo la interacción con el sistema operativo y la GUI.

```
int main(int argc, char *argv[])
```

Main.cpp - Explicación: - Esta es la firma de la función principal main, que es el punto de entrada de cualquier aplicación en C++. - Los parámetros argc y argv son utilizados para recibir argumentos desde la línea de comandos al ejecutar la aplicación. argc es el número de argumentos y argv es un arreglo de cadenas con los argumentos.

```
    QApplication a(argc, argv);
```

Main.cpp - Explicación: - Se crea un objeto de la clase QApplication, que es necesario para gestionar la aplicación Qt. Este objeto se encarga de manejar la inicialización de la aplicación, eventos, interacción

con la interfaz, etc. - Los parámetros argc y argv se pasan al constructor para que QApplication pueda procesar cualquier argumento de línea de comandos relacionado con la aplicación.

```
    MainWindow w;
```

Main.cpp - Explicación: - Aquí se declara un objeto w de la clase MainWindow, que representa la ventana principal de la aplicación. - MainWindow es una clase definida en el archivo mainwindow.h y contiene toda la lógica de la ventana principal y su interfaz gráfica.

```
    w.show();
```

Main.cpp - Explicación: - La función show() muestra la ventana principal en la pantalla. Este método es proporcionado por la clase QWidget (de la cual MainWindow hereda) y hace visible la interfaz gráfica para el usuario.

```
    return a.exec();
```

Main.cpp - Explicación: - La función exec() inicia el bucle de eventos de Qt, que es necesario para que la aplicación responda a las interacciones del usuario (como clics, entradas de teclado, etc.). - La llamada a exec() bloquea la ejecución hasta que la aplicación se cierra, y su valor de retorno indica cómo terminó el ciclo de ejecución (normalmente, un código de salida de la aplicación).

Código y Explicación del archivo mainwindow.h:

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H
```

Explicación: - Esta es la directiva de preprocesador para evitar la inclusión múltiple de este archivo. - Si MAINWINDOW_H no ha sido definida previamente, se define y se incluye el contenido del archivo. - Si MAINWINDOW_H ya ha sido definida, el contenido no se incluirá nuevamente.

```
#include <QMainWindow>
#include <QtMultimedia>
#include <QtMultimediaWidgets>
#include <QFileDialog>
```

```
#include <QStyle>
#include <QAudioOutput>
#include <QtCore>
#include <QtWidgets>
#include <QtGui>
#include <QTimer>
#include <QListWidgetItem>
```

Explicación: - Estas son las directivas de inclusión necesarias para utilizar las funcionalidades del framework Qt. - QMainWindow es la clase base para una ventana principal. - QtMultimedia y QtMultimediaWidgets permiten trabajar con multimedia (audio y video). - QFileDialog, QStyle, QAudioOutput, entre otros, proporcionan clases para manejo de archivos, estilo, salida de audio, y más.

```
QT_BEGIN_NAMESPACE
namespace Ui {
class MainWindow;
}
QT_END_NAMESPACE
```

Explicación: - Este bloque encapsula la declaración del espacio de nombres Ui, que es donde se declara la interfaz de usuario (UI) de la ventana principal.

```
class MainWindow : public QMainWindow
{
    Q_OBJECT
```

Explicación: - MainWindow hereda de QMainWindow, lo que permite que este archivo sea la ventana principal de la aplicación. - Q_OBJECT es un macro de Qt que habilita las características del sistema de señales y ranuras (signals/slots) en la clase.

```
public:
    MainWindow(QWidget *parent = nullptr);
    ~MainWindow();
```

Explicación: - El constructor MainWindow(QWidget *parent) crea una instancia de la ventana principal, tomando como parámetro un posible widget padre. - El destructor MainWindow() asegura la correcta liberación de recursos cuando la ventana principal se cierre.

```
private slots:
    void updateSliderPosition(); // Declaraci n de la
        funci n
    void durationChanged(qint64 duration);
    void positionChanged(qint64 position);
```

Explicación: - Estas son funciones que se conectan a señales (signals) y se invocan cuando ocurre un evento, como el cambio en la posición del video o su duración.

```
    void on_horizontalSlide_DurationV_valueChanged(int value
        );
    void on_pushButton_Play_PauseV_clicked();
    void on_pushButton_StopV_clicked();
    void on_pushButton_VolumeV_clicked();
    void on_horizontalSlider_VolumeV_valueChanged(int value)
        ;
    void on_pushButton_Seek_BackwardV_clicked();
    void on_pushButton_Seek_ForwardV_clicked();
```

Explicación: - Estas funciones se conectan a señales de la interfaz gráfica (por ejemplo, cambios en los controles deslizantes o botones del reproductor de video). - Son funciones para controlar el reproductor, como la reproducción, el volumen y la búsqueda hacia adelante o atrás en el video.

```
    void on_treeView_clicked(const QModelIndex &index);
```

Explicación: - Esta función se llama cuando un elemento es seleccionado en la vista de árbol (probablemente un árbol de archivos). - El índice del archivo seleccionado se pasa como parámetro para identificar el archivo que se va a reproducir.

```
    void on_listWidget_itemClicked(QListWidgetItem *item);
```

Explicación: - Esta función maneja el clic en un elemento de la lista (QListWidget), que probablemente contenga los archivos disponibles para reproducir. - Cuando se hace clic en un archivo, la función toma el item correspondiente para cargarlo y reproducirlo.

```
    void on_pushButton_Fullscreen_clicked();
    void onVideoStateChanged(QMediaPlayer::PlaybackState
        state);
```

Explicación: - Estas funciones permiten gestionar el modo de pantalla completa y el estado de reproducción del video (por ejemplo, si está pausado o en reproducción).

```
private:
    Ui::MainWindow *ui;
    QTimer *updateTimer;
    QMediaPlayer *Player;
    QVideoWidget *Video;
    QAudioOutput *videoAudioOutput;
    qint64 Mduration;
    QMediaPlayer *audioPlayer;
    QAudioOutput *audioOutput;
    bool IS_Pause;
    bool IS_Muted;
    QFileSystemModel *directorio;
    QFileSystemModel *archivo;
    QListWidget *listWidget;
    void updateDuration(qint64 duration);
    void createVideoWidget();
};
```

Explicación: - En esta sección se declaran varias variables y objetos que forman parte de la implementación de la clase MainWindow. - Incluye punteros a objetos de QMediaPlayer, QVideoWidget, QAudioOutput, y otras clases de Qt necesarias para la gestión de audio, video y archivos.
- updateDuration() y createVideoWidget() son funciones privadas adicionales utilizadas para actualizar la duración del archivo multimedia y crear el widget de video.

```
#endif // MAINWINDOW_H
```

Explicación: - Esta es la directiva de cierre para el ifndef MAINWINDOW_H, asegurando que el archivo solo se incluya una vez en el código fuente.

mainwindow.cpp

## Constructor: MainWindow::MainWindow

```
MainWindow::MainWindow(QWidget *parent) :
```

```
    QMainWindow(parent), ui(new Ui::MainWindow),
    audioPlayer(nullptr), Player(nullptr), videoAudioOutput(
        nullptr)  // Inicializaci n de punteros
{
    ui->setupUi(this);  // Setup de la UI

    // Inicializaci n del VideoWidget
    Video = new QVideoWidget(this);
    Video->setParent(ui->groupBox_Video);

    // NUEVO
    QVBoxLayout *videoLayout = new QVBoxLayout(ui->
        groupBox_Video);
    videoLayout->addWidget(Video);
    videoLayout->setContentsMargins(0, 0, 0, 0);  // Sin
        m rgenes para un ajuste completo
    ui->groupBox_Video->setLayout(videoLayout);
    //

    audioPlayer = new QMediaPlayer(this);
    videoAudioOutput = new QAudioOutput(this);  //
        Aseg rate de que el QAudioOutput est  configurado
    audioPlayer->setAudioOutput(videoAudioOutput);  //
        Asocia audioPlayer con videoAudioOutput

    // Configuraci n del QFileSystemModel para el treeView
    directorio = new QFileSystemModel(this);
    directorio->setRootPath(QDir::rootPath());
    ui->treeView->setModel(directorio);
    ui->treeView->setRootIndex(directorio->index(QDir::
        rootPath())); // Muestra el root por defecto

    // Inicializar el QListWidget
    listWidget = ui->listWidget; // Asumiendo que tienes un
        QListWidget en tu UI
    listWidget->clear();  // Limpiar el QListWidget al
        inicio
    listWidget->setSelectionMode(QAbstractItemView::
        SingleSelection); // Permitir selecci n de un solo
        item

    // Conectar la se al de selecci n de carpeta
    connect(ui->treeView, &QTreeView::clicked, this, &
        MainWindow::on_treeView_clicked);

    // Conectar la selecci n de archivos en el QListWidget
    connect(listWidget, &QListWidget::itemClicked, this, &
        MainWindow::on_listWidget_itemClicked);

    // Configuraci n del reproductor de video
```

```
40      Player = new QMediaPlayer(this);
41      videoAudioOutput = new QAudioOutput(this);
42      Player->setAudioOutput(videoAudioOutput);
43      Player->setVideoOutput(Video); // Aseg rate de que el
            reproductor
44      connect(Player, &QMediaPlayer::playbackStateChanged,
            this, &MainWindow::onVideoStateChanged);
45
46      // Ajustes visuales del reproductor de video
47      ui->pushButton_Play_PauseV->setIcon(style()->
            standardIcon(QStyle::SP_MediaPlay));
48      ui->pushButton_StopV->setIcon(style()->standardIcon(
            QStyle::SP_MediaStop));
49      ui->pushButton_Seek_BackwardV->setIcon(style()->
            standardIcon(QStyle::SP_MediaSeekBackward));
50      ui->pushButton_Seek_ForwardV->setIcon(style()->
            standardIcon(QStyle::SP_MediaSeekForward));
51      ui->pushButton_VolumeV->setIcon(style()->standardIcon(
            QStyle::SP_MediaVolume));
52
53      // Ajustamos el slider de volumen del video
54      ui->horizontalSlider_VolumeV->setMinimum(0);
55      ui->horizontalSlider_VolumeV->setMaximum(100);
56      ui->horizontalSlider_VolumeV->setValue(30); // Volumen
            inicial
57      videoAudioOutput->setVolume(ui->horizontalSlider_VolumeV
            ->value() / 100.0);
58
59      // Inicializamos las banderas de estado
60      IS_Pause = true;
61      IS_Muted = false;
62
63      // Barra y tiempo de duraci n de los archivos
64      connect(Player, &QMediaPlayer::durationChanged, this, &
            MainWindow::durationChanged);
65      connect(Player, &QMediaPlayer::positionChanged, this, &
            MainWindow::positionChanged);
66
67      // En el constructor
68      ui->horizontalSlide_DurationV->setRange(0, 0);
69      Mduration = 0;
70
71      // Configuraci n del QTimer para actualizar el slider
72      updateTimer = new QTimer(this);
73      updateTimer->setInterval(500);  // Actualiza cada medio
            segundo
74      connect(updateTimer, &QTimer::timeout, this, &MainWindow
            ::updateSliderPosition);
75      updateTimer->start();
76  }
```

Descripción: El constructor MainWindow::MainWindow es responsable de inicializar todos los elementos necesarios para que la interfaz gráfica y los componentes multimedia funcionen correctamente. Aquí se detallan los pasos clave de lo que realiza:

- Inicialización de la Interfaz Gráfica: Se llama a ui->setupUi(this) para configurar la interfaz de usuario que fue diseñada previamente con Qt Designer. Esta línea configura los elementos básicos de la ventana, como botones, sliders, y paneles.
- Inicialización de Video: Se crea un QVideoWidget, que es el widget de video en el cual se mostrará el contenido multimedia. El VideoWidget se añade al grupo de la interfaz groupBox_Video mediante un QVBoxLayout, asegurando que se ajuste al tamaño del contenedor.
- Inicialización de los Reproductores: Se crean dos objetos QMediaPlayer (uno para el audio y otro para el video) y se asocian con el objeto QAudioOutput para gestionar la salida de audio. Los objetos audioPlayer y videoAudioOutput se inicializan y se conectan correctamente.
- Configuración del TreeView y el ListWidget: Se inicializa un QFileSystemModel para mostrar el sistema de archivos en un QTreeView. Este modelo muestra las carpetas y archivos del sistema, permitiendo al usuario seleccionar un archivo para reproducir. Además, se inicializa un QListWidget para mostrar una lista de elementos seleccionados.
- Conexión de Señales y Slots: Se conectan las señales del QTreeView y QListWidget a funciones que gestionan la selección de archivos, como on_treeView_clicked y on_listWidget_itemClicked. Esto permite que, cuando el usuario seleccione un archivo, se inicie la reproducción.
- Configuración Visual del Reproductor de Video: Se configuran los íconos de los botones de control de video (reproducir, pausar, detener, volumen, etc.) utilizando los íconos estándar proporcionados por el estilo de Qt.
- Control de Volumen: El control de volumen del video se configura para tener un rango entre 0 y 100, con un valor inicial de 30. El volumen se ajusta utilizando el objeto videoAudioOutput.
- Inicialización de Bandera de Estado: Se configuran las banderas IS_Pause y IS_Muted para gestionar el estado de pausa y mute en el reproductor.
- Conexión de Señales de Reproducción: Se conectan las señales durationChanged y positionChanged del Player a las funciones correspondientes

durationChanged y positionChanged, para actualizar la interfaz de usuario en tiempo real.

- Configuración de un QTimer para Actualizar el Slider: Se crea un QTimer para actualizar la posición del slider de duración de video cada 500 ms, asegurando que la interfaz esté sincronizada con el progreso de la reproducción.

## Destructor: MainWindow:: MainWindow

```
MainWindow::~MainWindow()
{
    delete Player;
    delete videoAudioOutput;
    delete Video; // Eliminar el VideoWidget
    delete ui;
}
```

Descripción: El destructor de MainWindow se encarga de liberar la memoria ocupada por los objetos creados dinámicamente en el constructor. Esto incluye el QMediaPlayer, QAudioOutput, QVideoWidget y la interfaz de usuario.

Explicación Detallada de Funciones

Este documento describe cada una de las funciones de un programa en C++ que controla la reproducción de audio y video, junto con los controles de volumen, posición y duración. El código está ilustrado en bloques con una breve explicación de su propósito y funcionamiento.

## Función: onVideoStateChanged

Esta función se ejecuta cuando cambia el estado de reproducción del video. Si el estado del reproductor es "detenido", la función reinicia el video y lo reproduce nuevamente desde el inicio.

```
void MainWindow::onVideoStateChanged(QMediaPlayer::
    PlaybackState state) {
    if (state == QMediaPlayer::StoppedState) {
        // Si el video se detuvo, reiniciarlo para que est
            listo para la pr xima reproducci n
        Player->setPosition(0); // Reiniciar el video
        Player->play(); // Reproducir el video en bucle
    }
}
```

Listing 1: onVideoStateChanged

## Función: `updateSliderPosition`

Esta función actualiza la posición del slider que controla la duración del video. Si el audio o el video están en reproducción, actualiza el valor del slider y la duración mostrada en la interfaz gráfica.

```
void MainWindow::updateSliderPosition() {
    if (!ui->horizontalSlide_DurationV->isSliderDown()) {
        ui->horizontalSlide_DurationV->blockSignals(true);

        // Verificar si el audio o el video est n
            reproduci ndose
        if (audioPlayer->playbackState() == QMediaPlayer::
            PlayingState) {
            qint64 position = audioPlayer->position();
            ui->horizontalSlide_DurationV->setValue(position
                / 1000);
            updateDuration(position / 1000);
        } else if (Player->playbackState() == QMediaPlayer::
            PlayingState) {
            qint64 position = Player->position();
            ui->horizontalSlide_DurationV->setValue(position
                / 1000);
            updateDuration(position / 1000);
        }

        ui->horizontalSlide_DurationV->blockSignals(false);
    }
}
```

Listing 2: updateSliderPosition

## Función: updateDuration

Esta función actualiza la duración mostrada en la interfaz de usuario, tanto para la duración actual como para la duración total del archivo que se está reproduciendo. Se utiliza un formato de hora y minuto para mostrar la información.

```
void MainWindow::updateDuration(qint64 duration)
{
    if (duration > 0 && Mduration > 0) {
        QTime CurrentTime((duration / 3600) % 60, (duration
            / 60) % 60, duration % 60);
        QTime TotalTime((Mduration / 3600) % 60, (Mduration
            / 60) % 60, Mduration % 60);

        QString format = "mm:ss";
        if (Mduration > 3600) {
            format = "hh:mm:ss";
        }

        ui->label_Current_Time->setText(CurrentTime.toString
            (format));
        ui->label_Total_Time->setText(TotalTime.toString(
            format));
    }
}
```

Listing 3: updateDuration

## Función: durationChanged

Cuando la duración del video o el audio cambia, esta función ajusta el valor máximo del slider de la duración y actualiza la etiqueta de duración total.

```
void MainWindow::durationChanged(qint64 duration) {
    Mduration = duration / 1000;
    ui->horizontalSlide_DurationV->setMaximum(Mduration);

    QTime TotalTime((Mduration / 3600) % 60, (Mduration /
        60) % 60, Mduration % 60);
    QString format = "mm:ss";
    if (Mduration > 3600) {
        format = "hh:mm:ss";
    }
    ui->label_Total_Time->setText(TotalTime.toString(format)
        );
}
```

Listing 4: durationChanged

## Función: positionChanged

Esta función maneja el cambio de posición tanto en el audio como en el video. Cuando el usuario mueve el slider de duración, actualiza la posición del video o audio.

```
void MainWindow::positionChanged(qint64)
{
    if (!ui->horizontalSlide_DurationV->isSliderDown()) {
        updateSliderPosition();  // Mover la l gica al
            QTimer
    }
}
```

Listing 5: positionChanged

## Función: on_horizontalSlide_DurationV_valueChanged

Este evento controla la posición de reproducción del video, ajustando la posición del reproductor de video cuando el usuario cambia el valor del slider.

```
void MainWindow::on_horizontalSlide_DurationV_valueChanged(
    int value)
{
    Player->setPosition(value * 1000);
}
```

Listing 6: on_horizontalSlide_DurationV_valueChanged

## Función: on_pushButton_Play_PauseV_clicked

Este botón permite reproducir o pausar el video y el audio. Si el video o el audio están en pausa, los reanuda, y si están en reproducción, los pausa. Cambia el icono del botón según el estado de reproducción.

```
void MainWindow::on_pushButton_Play_PauseV_clicked() {
    if (IS_Pause) {
        // Reanudar reproducci n solo del reproductor
            activo (audio o video)
```

```
        if (audioPlayer->playbackState() == QMediaPlayer::
            PausedState) {
            audioPlayer->play();
        }
        if (Player->playbackState() == QMediaPlayer::
            PausedState) {
            Player->play();
        }

        // Cambiar el icono a "Pausar"
        ui->pushButton_Play_PauseV->setIcon(style()->
            standardIcon(QStyle::SP_MediaPause));
    } else {
        // Pausar ambos reproductores
        Player->pause();
        audioPlayer->pause();

        // Cambiar icono a "Reproducir"
        ui->pushButton_Play_PauseV->setIcon(style()->
            standardIcon(QStyle::SP_MediaPlay));
    }
    IS_Pause = !IS_Pause;
}
```

Listing 7: on_pushButton_Play_PauseV_clicked

## Función: on_pushButton_StopV_clicked

Este botón detiene la reproducción del video y lo reinicia desde el principio, volviendo a reproducirlo.

```
void MainWindow::on_pushButton_StopV_clicked()
{
    Player->stop();

    // Regresar al inicio del video (en el tiempo 0)
    Player->setPosition(0);  // Regresa al principio

    // Reproducir el video nuevamente
    Player->play();
}
```

Listing 8: on_pushButton_StopV_clicked

## Función: on_pushButton_VolumeV_clicked

Este botón permite activar o desactivar el sonido del video. Si el sonido está activado, lo apaga, y si está apagado, lo activa nuevamente. También cambia el icono de acuerdo con el estado de volumen.

```
void MainWindow::on_pushButton_VolumeV_clicked()
{
    IS_Muted = !IS_Muted;
    ui->pushButton_VolumeV->setIcon(IS_Muted ? style()->
        standardIcon(QStyle::SP_MediaVolumeMuted) : style()->
        standardIcon(QStyle::SP_MediaVolume));
    videoAudioOutput->setMuted(IS_Muted);
}
```

Listing 9: on_pushButton_VolumeV_clicked

## Función: on_horizontalSlider_VolumeV_valueChanged

Este evento controla el volumen del video, ajustando el volumen del reproductor de video en función del valor del slider.

```
void MainWindow::on_horizontalSlider_VolumeV_valueChanged(
    int value)
{
    videoAudioOutput->setVolume(value / 100.0);
}
```

Listing 10: on_horizontalSlider_VolumeV_valueChanged

## Función: on_pushButton_Seek_BackwardV_clicked

Este botón permite volver al archivo de video o audio anterior de la lista de reproducción.

```
void MainWindow::on_pushButton_Seek_BackwardV_clicked() {
    int currentIndex = listWidget->currentRow();
    if (currentIndex > 0) {
        listWidget->setCurrentRow(currentIndex - 1);
        on_listWidget_itemClicked(listWidget->currentItem())
            ; // Reproducir archivo seleccionado
    }
}
```

Listing 11: on_pushButton_Seek_BackwardV_clicked

## Función: on_pushButton_Seek_ForwardV_clicked

Este botón permite avanzar al siguiente archivo de video o audio en la lista de reproducción.

```
void MainWindow::on_pushButton_Seek_ForwardV_clicked() {
    int currentIndex = listWidget->currentRow();
    if (currentIndex < listWidget->count() - 1) {
        listWidget->setCurrentRow(currentIndex + 1);
        on_listWidget_itemClicked(listWidget->currentItem())
            ; // Reproducir archivo seleccionado
    }
}
```

Listing 12: on_pushButton_Seek_ForwardV_clicked

## Función: on_pushButton_Fullscreen_clicked

Esta función es un *slot* de Qt que se ejecuta cuando el usuario hace clic en un botón de pantalla completa. Se encarga de alternar entre los modos de pantalla completa y ventana normal para el widget de video. Aquí se manejan varias operaciones, como restaurar el tamaño original del widget y agregar un botón flotante (overlay) para controlar la pantalla completa.

```
void MainWindow::on_pushButton_Fullscreen_clicked() {
    if (!Video) {
        qDebug() << "Error: El widget de video no est
            inicializado.";
        return;
    }

    if (isFullscreen) {
        // Salir de pantalla completa
        Video->setWindowFlags(Qt::Widget);    // Restaurar
            como widget normal
        Video->showNormal();  // Restaurar la apariencia
            normal del widget

        // Asegurarse de que el widget de video se ajuste al
             tama o de su contenedor
        Video->resize(ui->groupBox_Video->size()); //
            Ajustar el tama o al groupBox
        Video->setGeometry(5, 5, ui->groupBox_Video->width()
             - 10, ui->groupBox_Video->height() - 10);
        isFullscreen = false;

        // Ocultar y eliminar el overlay
        if (fullscreenOverlay) {
```

```
            fullscreenOverlay->close();
            delete fullscreenOverlay;
            fullscreenOverlay = nullptr;
        }

        ui->pushButton_Fullscreen->setText("[ ]");
    } else {
        // Ir a pantalla completa
        Video->setWindowFlags(Qt::Window);  // Convertir el
            widget de video en una ventana independiente
        Video->showFullScreen(); // Mostrar en pantalla
            completa

        isFullscreen = true;
        ui->pushButton_Fullscreen->setText("[ ]");

        // Crear una ventana flotante (overlay) para el
            bot n en pantalla completa
        fullscreenOverlay = new QWidget(Video);
        fullscreenOverlay->setWindowFlags(Qt::
            FramelessWindowHint | Qt::Tool | Qt::
            WindowStaysOnTopHint);
        fullscreenOverlay->setAttribute(Qt::
            WA_TranslucentBackground, true);
        fullscreenOverlay->resize(50, 50); // Tama o del
            bot n de overlay
        fullscreenOverlay->move(Video->frameGeometry().width
            () - fullscreenOverlay->width() - 10, 700);

        // Crear el bot n en la ventana flotante
        QPushButton *fullscreenButton = new QPushButton("[
            ]", fullscreenOverlay);
        fullscreenButton->resize(fullscreenOverlay->size());

        // Conectar el bot n al mismo slot para alternar
            entre los modos
        connect(fullscreenButton, &QPushButton::clicked,
            this, &MainWindow::
            on_pushButton_Fullscreen_clicked);

        fullscreenOverlay->show();
    }
}
```

Listing 13: on_pushButton_Fullscreen_clicked

Explicación: - La función comienza verificando si el widget de video (Video) ha sido inicializado. Si no es así, se emite un mensaje de error y la función termina. - Si el widget está en modo pantalla

completa (isFullscreen es true), entonces se restaura a su tamaño y posición original, y se elimina el fullscreenOverlay que se muestra sobre el video. - Si el widget no está en pantalla completa, se cambia su configuración para que ocupe toda la pantalla (showFullScreen()) y se agrega un overlay con un botón flotante en la esquina superior derecha, que permite salir de la pantalla completa. Este botón tiene el mismo comportamiento que el botón original.

## Función: createVideoWidget

La función createVideoWidget crea un widget de video (QVideoWidget) si no existe uno ya, y lo coloca en un contenedor específico dentro de la interfaz de usuario. Además, establece el reproductor de video (Player) para que se muestre en este widget.

```
void MainWindow::createVideoWidget() {
    // Solo crear el widget si no existe uno ya
    if (!Video) {
        // Verificar que Player no es nulo antes de
            configurarlo
        if (!Player) {
            qDebug() << "Error: El reproductor de video no
                est  inicializado.";
            return; // No continuar si Player no est
                inicializado
        }

        Video = new QVideoWidget(this);  // Crear una nueva
            instancia si no existe
        Video->setGeometry(5, 5, ui->groupBox_Video->width()
             - 10, ui->groupBox_Video->height() - 10);
        Video->setParent(ui->groupBox_Video);  // Colocar el
             widget en el groupBox

        // Configurar el reproductor de video
        Player->setVideoOutput(Video);
        Video->setVisible(true);
        Video->show();
        qDebug() << "VideoWidget creado y configurado
            correctamente.";
    }
}
```

Listing 14: createVideoWidget

Explicación: - La función comienza verificando si el widget de video (Video) ya existe. Si es así, no realiza ninguna acción. -

Si Video aún no está creado, la función crea una nueva instancia de QVideoWidget, la posiciona dentro de un contenedor groupBox_Video y le asigna el reproductor de video (Player) para que se muestre en ese widget. - Posteriormente, se configura el reproductor para que envíe la salida de video al widget creado y lo hace visible.

## Función: on_treeView_clicked

La función on_treeView_clicked maneja los clics en un árbol de directorios. Si el usuario selecciona una carpeta, se cargan los archivos de video y audio en un QListWidget. Si se selecciona un archivo, el reproductor de video o audio se detiene y carga el nuevo archivo para reproducirlo.

```
void MainWindow::on_treeView_clicked(const QModelIndex &
    index) {
    // Obtener la ruta completa del archivo o directorio
    QString filePath = directorio->filePath(index);
    QFileInfo fileInfo(filePath);

    // Verificar si se ha seleccionado un directorio
    if (fileInfo.isDir()) {
        // Si es un directorio, actualizar el QListWidget
            con los archivos contenidos en esa carpeta
        listWidget->clear();  // Limpiar el QListWidget
            antes de agregar nuevos archivos

        // Cargar los archivos de video y audio en el
            QListWidget
        QDir dir(filePath);
        QStringList audioFiles = dir.entryList(QStringList()
             << "*.mp3" << "*.wav" << "*.flac
        \vspace{1em}

\section*{Funci n: \texttt{on\_listWidget\_itemClicked}}

La funci n \texttt{on\_listWidget\_itemClicked} se ejecuta
    cuando el usuario selecciona un archivo de la lista de
    archivos (audio o video). En esta funci n se gestionan
    los clics sobre los elementos de un \texttt{QListWidget},
     lo que desencadena la reproducci n del archivo
    seleccionado.

\begin{lstlisting}[caption={on\_listWidget\_itemClicked}]
void MainWindow::on_listWidget_itemClicked(QListWidgetItem *
    item) {
    QString fileName = item->text();
    QString dirPath = directorio->filePath(ui->treeView->
        currentIndex());
```

```
24      QString filePath = QDir(dirPath).filePath(fileName);
25      QFileInfo fileInfo(filePath);
26
27      // Parar cualquier reproducci n actual (audio y video)
28      audioPlayer->stop();  // Detener el audio
29      Player->stop();  // Detener el video
30
31      // Limpiar las fuentes de los reproductores antes de
            asignar nuevas
32      audioPlayer->setSource(QUrl());
33      Player->setSource(QUrl());
34
35      // Mostrar el nombre del archivo en la etiqueta
            label_FileName
36      ui->label_Value_File_Name->setText(fileInfo.fileName());
37
38      if (fileInfo.suffix() == "mp4" || fileInfo.suffix() == "
            avi" || fileInfo.suffix() == "mkv") {
39          // Reproducci n de video
40          Player->setSource(QUrl::fromLocalFile(filePath));
41          Player->play();
42
43          // Cambiar el icono de "Pausar"
44          IS_Pause = false;
45          ui->pushButton_Play_PauseV->setIcon(style()->
                standardIcon(QStyle::SP_MediaPause));
46
47          // Asegurarse de que el VideoWidget est  visible
48          if (Video) {
49              Video->setVisible(true);
50              Video->show();
51          }
52
53      } else if (fileInfo.suffix() == "mp3" || fileInfo.suffix
            () == "wav" || fileInfo.suffix() == "flac") {
54          // Reproducci n de audio
55          audioPlayer->setSource(QUrl::fromLocalFile(filePath)
                );
56          audioPlayer->play();
57
58          // Mostrar el video de fondo mientras se reproduce
                el audio
59          Player->setSource(QUrl("qrc:/new/prefix1/
                VideoFondoMP3.mp4")); // Establecer el video de
                fondo
60          Player->play(); // Reproducir el video de fondo
61
62          // Cambiar el icono de "Pausar"
63          IS_Pause = false;
64          ui->pushButton_Play_PauseV->setIcon(style()->
```

```
            standardIcon(QStyle::SP_MediaPause));
65
66          // Asegurarse de que el VideoWidget est  visible
67          if (Video) {
68              Video->setVisible(true);
69              Video->show();
70          }
71      } else {
72          // Si el archivo no es v lido (ni video ni audio),
                mostrar un mensaje de advertencia
73          QMessageBox::warning(this, tr("Formato no soportado
                "), tr("Por favor selecciona un archivo de video
                o audio v lido."));
74      }
75  }
```

Listing 15: on_treeView_clicked

Explicación: - Cuando el usuario hace clic en un ítem dentro de un QListWidget, el archivo seleccionado es identificado por su nombre. - Se obtiene la ruta completa del archivo utilizando filePath y QDir. - Se detienen las reproducciones anteriores de video y audio para evitar conflictos, y se limpia el estado actual de ambos reproductores. - Dependiendo del tipo de archivo (video o audio), se selecciona el reproductor adecuado y se comienza a reproducir el archivo. - Para los archivos de video, el reproductor de video (Player) se configura con la URL del archivo seleccionado y se reproduce. - Para los archivos de audio, se configura un video de fondo predeterminado (VideoFondoMP3.mp4) que se reproduce en paralelo con el audio. - Si el archivo no es de un formato válido, se muestra un mensaje de advertencia al usuario.

## Funcionamiento General y Flujos de Control

El código anterior maneja la reproducción tanto de archivos de audio como de video dentro de una interfaz gráfica utilizando Qt. A continuación se describe el comportamiento general:

- **Interacción con el botón de pantalla completa:**
  - El usuario puede hacer clic en un botón para alternar entre el modo de pantalla completa y el modo ventana normal para el widget de video.
  - En modo pantalla completa, el reproductor ocupa toda la pantalla, y se puede controlar mediante un botón flotante para salir de este modo.

- **Carga y visualización de videos y audios:**
  - El usuario puede navegar por un sistema de directorios (mediante un QTreeView) y seleccionar un archivo de audio o video.
  - Si se selecciona un directorio, se listan los archivos de medios disponibles en un QListWidget.
  - Si se selecciona un archivo, se reproduce dependiendo de su tipo. Si es video, se reproduce en el widget de video; si es audio, se reproduce un video de fondo mientras se reproduce el audio.
- **Control de la interfaz y los reproductores:**
  - Se asegura que los reproductores de video y audio no estén activos simultáneamente, y si es necesario, se limpian las fuentes antes de cargar un nuevo archivo.
  - Además, se actualiza la interfaz con el nombre del archivo que se está reproduciendo.

## Conclusión

El código presentado gestiona de manera eficiente la reproducción de archivos multimedia (video y audio) dentro de una interfaz de usuario construida con Qt. La implementación es flexible y permite alternar entre diferentes modos de visualización, como pantalla completa y modo ventana. Además, permite la selección y reproducción de archivos desde un directorio, con un manejo adecuado de errores y la interfaz visual.

Este tipo de implementación es útil en aplicaciones como reproductores multimedia o reproductores de contenido en línea, donde se necesita alternar entre diferentes tipos de medios y controlar su visualización de manera dinámica.