OnStream MediaPlayer+
Player SDK Integration Guide

for
Android Platforms
SDK Version 3.8 and later

VisualOn, Inc.
http://www.visualon.com

October, 2013

Version 1.7
201310001

Copyright/Confidentiality Notice

VisualOn, Inc., 4675 Stevens Creek Blvd, Santa Clara, CA 95051, USA

CONFIDENTIALITY NOTICE

Table of Contents

Table of Figures

1 About This Manual

1.1 OVERVIEW

This manual describes the integration of the OnStream® MediaPlayer+ Player SDK (or "SDK") release with Android projects. This document includes the following topics:

- Pre-requisites
- Basic SDK Integration
- Advanced SDK Integration (by topic)

1.2 SCOPE

This manual is intended for Android developers who need to create a flexible and high-performance media player that supports playback of live or VOD streaming, progressive download, and local media sources.

Android developers are assumed to be familiar with: the Android SDK/ADT; the Eclipse IDE; the Java Native Interface (JNI) and Android NDK; and the Java and C/C++ programming languages.

1.3 REVISION HISTORY

Rev	Product Version	Date	Description
1.2	V3.5	2013-02-15	Document Creation
1.3	V3.5	2013-03-25	Integration Topics: CC/Subtitles; Video Track Switching; Suspend/Resume
1.4	V3.6	2013-07-25	Updated Troubleshooting Guide with Licensing Error Info
1.5	V3.6	2013-07-25	Added info on Restarting Playback
1.6	V3.7	2013-09-17	Updated with API3 changes
1.7	V3.7	2013-10-30	Updated with changes to open media source in Async Mode. Added a Section to explain APK file generation

1.4 RELATED DOCUMENTS

The following documents (included with your installation package) provide additional information related to this user guide:

- *OnStream MediaPlayer+ SDK Project Setup for Android Platforms*

- *OnStream MediaPlayer+ Player API Reference Manual for Android Platforms*

1.5 ABBREVIATIONS

Acronym	Description
API	Application Programming Interface
DRM	Digital Rights Management
IDE	Integrated Development Environment
JNI	Java Native Interface
OSMP+	OnStream MediaPlayer+
SDK	Software Development Kit

1.6 TYPOGRAPHIC CONVENTIONS

- **Directory Contents** are shown in "Calibri" font in blue.
- **File and Directory Names** are shown in "Calibri" font in *blue italics*.
- **File Contents and Source Code** are shown single-spaced in "`Courier New`" font.
- **Menu Options, Commands, and Windows/Views** are shown single-spaced in **bold**.
- **Project and Document Titles** are shown in *italics*.

Examples:

1. Select the **Select root directory** radio button, and input or browse to *<SDK_INSTALL_DIR>\Android\SamplePlayer*.
2. Under **Projects**, make sure that the *SamplePlayer* checkbox is selected. Click **Finish** to complete the import.
3. Customer module integration is discussed in the *OnStream MediaPlayer+ Player API Reference Manual for Android Platforms*.
4. Set the format for the surface using `SurfaceHolder.setFormat(PixelFormat.RGBA_8888);`.

2 Prerequisites

This section describes prerequisite documentation and examples for SDK integration.

2.1 INTEGRATION LABS

This guide uses source code examples that are included with the SDK release. These examples are provided in series of lab projects that incrementally add integration features. The integration labs can be found at *<SDK_INSTALL_DIR>\Android\Doc\Labs*.

Note: The integration labs require setup before use, including SDK and license file installation. SDK project setup is described in the *OnStream MediaPlayer+ SDK Project Setup for Android Platforms* manual.

2.2 VOCOMMONPLAYER API REFERENCE MANUAL

Access to the SDK is provided through the `VOCommonPlayer` interface, shown in Figure 2-1.

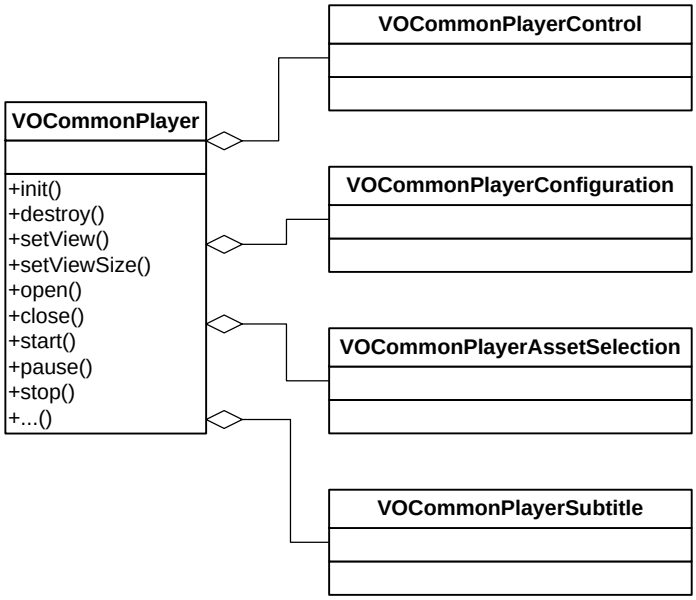

Figure 2-1: VOCommonPlayer SDK Interface Hierarchy

The `VOCommonPlayer`, `VOCommonPlayerControl`, `VOCommonPlayerConfiguration`, `VOCommonPlayerAssetSelection`, and `VOCommonPlayerSubtitle` interfaces are described in the *OnStream MediaPlayer+ Player API Reference Manual for Android Platforms*.

3 Basic Integration

This section describes the SDK integration with an Android client that implements a basic live stream media player, including open, playback, background, and home functionality.

3.1 INTEGRATION LAB

This section uses examples from *Integration Lab #1*. The source code examples can be found at *<SDK_INSTALL_DIR>\Android\Doc\Labs\Lab1\src\com\visualon\LabPlayer\player.java*.

3.2 INTEGRATION FLOW

Figure 3-1 below illustrates the integration flow of a basic SDK client media player.

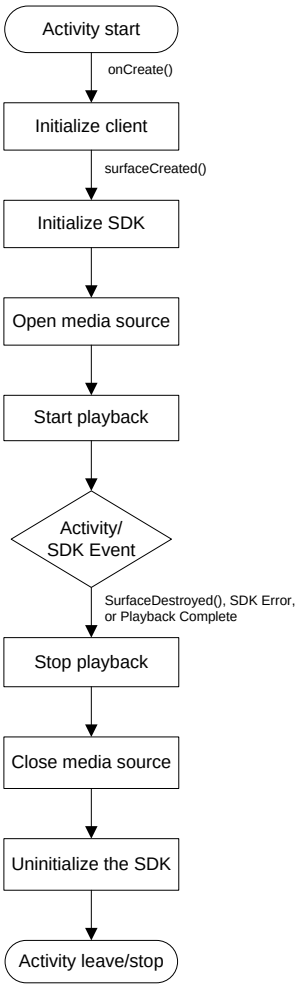

Figure 3-1: Flow Diagram for Basic SDK Client

3.3 SDK CLIENT CLASS DEFINITIONS

To integrate the SDK, a client class must be defined. Figure 3-2 illustrates the hierarchy of a basic SDK client called `Player`, which extends the Android `Activity` class. The `Player` class implements all of the SDK integration features, including media source open and playback, and Android home functionality. The `Player` class implements `SurfaceHolder.Callback` and `VOCommonPlayerListener` in order to manage both Android `Activity` and SDK events.

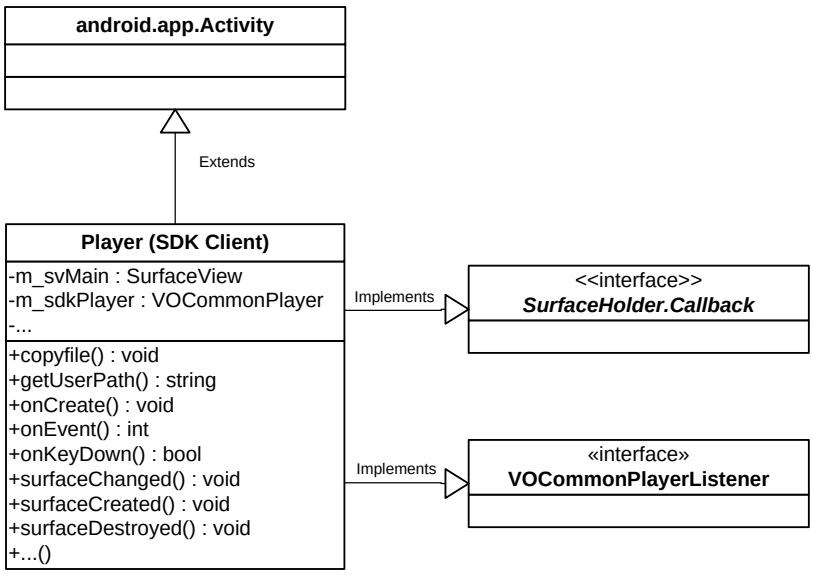

Figure 3-2: Basic Player (SDK Client) Class Hierarchy

3.3.1 Requirements and Recommendations

The SDK client shall:

- Include an instance of `VOCommonPlayer`, and `SurfaceView`.
- Implement the `SurfaceHolder.callback` interface.
- Implement the `VOCommonPlayerListener` interface.

3.3.2 More Information

For more information on the `VOCommonPlayerListener` interface, refer to the *OnStream MediaPlayer+ Player API Reference Manual for Android Platforms*.

For more information on `SurfaceHolder` and `SurfaceHolder.callback`, refer to the *Android Developers Reference* (android.view package).

3.4 INITIALIZING THE CLIENT

When the SDK client activity is started, it must be initialized with its user interface components and global parameters. Basic SDK client initialization includes:

1. Surface Identification and Configuration. The drawing surface must be identified and configured for use with the SDK.
2. SDK File Transfer. A license file and device capability file (optional) must be transferred to the device for access by the SDK.
3. Media Source Identification. A data source must be identified for playback.

Note: For SecurePlayer applications, you can start using this guide at section 3.6 (Starting Playback).

3.4.1 Surface Identification and Configuration

An Android `SurfaceView` and its `SurfaceHolder` provide the drawing surface for the SDK. The `SurfaceHolder` must be configured to the RGB32 pixel format, and its callbacks must be initiated to notify the SDK of any surface changes.

In the following example, the `onCreate()` method, which is called when an Android activity launches, finds the `SurfaceView` and its `SurfaceHolder`, adds the `SurfaceHolder` callback, and sets the `SurfaceHolder` pixel format.

Sample code:

```
public void onCreate(Bundle savedInstanceState)
{
    ...
    // Find SurfaceView and Surface holder from the layout
    m_svMain = (SurfaceView) findViewById(R.id.svMain);
    // Add a Callback for this holder
    m_svMain.getHolder().addCallback(this);
    // Set pixel format to RGB32
    m_svMain.getHolder().setFormat(PixelFormat.RGBA_8888);
    ...
}
```

3.4.2 SDK File Transfer

The SDK requires a license file (provided by VisualOn, Inc.) that must be locally installed with the SDK client application on the device. The SDK may also leverage a local device capability file, which optimizes playback by limiting bit rates and resolutions to those supported by the particular device/processor.

Files can be transferred from an Android project to the device as assets, but it may be desirable to copy the asset files to the package directory on the device.

In the following example, `onCreate()` calls the `copyfile()` method, which implements the copy, for the license and device capability files.
Sample code:

```
public void onCreate(Bundle savedInstanceState)
{
   ...
   // Copy license file and device capability file
   copyfile(this, "voVidDec.dat", "voVidDec.dat");
   copyfile(this, "cap.xml", "cap.xml");
}
...

private static void copyfile(Context context, String filename,
      String desName) {
   try {
      InputStream InputStreamis = context.getAssets().open(filename);
      File desFile = new File(getUserPath(context) + "/" + desName);
      desFile.createNewFile();
      FileOutputStream fos = new FileOutputStream(desFile);
      int bytesRead;
      byte[] buf = new byte[4 * 1024]; // 4K buffer

      while ((bytesRead = InputStreamis.read(buf)) != -1) {
         fos.write(buf, 0, bytesRead);
      }
      fos.flush();
      fos.close();
   } catch (IOException e) {
      e.printStackTrace();
   }
}
```

3.4.3 Media Source Identification

The SDK will require a media source (e.g., URL or file path) for playback. For a basic SDK client implementation, this may be provided using a global `String` variable.

In the following example, `onCreate()` identifies the media source using the `m_strVideoPath` variable.

Sample code:

```
public void onCreate(Bundle savedInstanceState)
{
   ...
   // Define your playback URL/local media file here
   m_strVideoPath =
      "http://devimages.apple.com/iphone/samples/bipbop/bipbopall.m3u8"
      ;
   ...
}
```

3.4.4 Requirements and Recommendations

The SDK client shall:

- Find the `SurfaceView` and its `SurfaceHolder`.
- Initiate the `SurfaceHolder` callback and set the pixel format to RGB32.
- Copy the *voVidDec.dat* license file to the local package directory.
- Identify a media source for playback.

The SDK client should:

- Copy the *cap.xml* configuration file to the local package directory.

3.4.5 More Information

For more information on `SurfaceView` and `SurfaceHolder`, refer to the *Android Developers Reference* (SurfaceView and SurfaceHolder).

3.5 INITIALIZING THE SDK

SDK initialization and configuration is required before any media playback can occur. The initialization and configuration tasks are described in Table 3-1.

Table 3-1: SDK Initialization and Configuration Tasks

Task	VOCommonPlayer Method(s)	Description
Initialize the SDK Instance	`init()`	Provide SDK with library file path using package `context` and set media framework to `VO_OSMP_VOME2_PLAYER`.
Configure SDK View Settings	`setView()`	Provide SDK with the current `SurfaceView`.
	`setViewSize()`	Provide SDK with the display size, based on the default `DisplayMetrics`.
Register SDK Event Listener	`setOnEventListener()`	Register SDK event listener callback to manage SDK events.
Configure Device Capability Settings (optional)	`setDeviceCapabilityByFile()`	Provide SDK with location of file containing device-specific display sizes and bit rates. Location based on `copyfile()` method described in section 3.4.2 (SDK File Transfer).
Configure License Settings	`setLicenseFilePath()`	Provide SDK with the location of the license file. Location based on `copyfile()` method described in section 3.4.2 (SDK File Transfer).

In the following example, the `surfaceCreated()` callback method, which is called when the surface is created, implements the SDK initialization and configuration tasks from Table 3-1. The `getUserPath()` method is a utility that provides the path to the package data, where the SDK library files and configuration files are located.

Sample code:

```
public void surfaceCreated(SurfaceHolder surfaceholder) {
   ...
   // Initialize the SDK
   VO_OSMP_RETURN_CODE nRet;

   m_sdkPlayer = new VOCommonPlayerImpl();

   m_svMain.getHolder().setType(SurfaceHolder.SURFACE_TYPE_NORMAL);

   // Retrieve location of libraries
   String apkPath = getUserPath(this) + "/lib/";
   String cfgPath = getUserPath(this) + "/";

   // SDK player engine type
   VO_OSMP_PLAYER_ENGINE eEngineType =
      VO_OSMP_PLAYER_ENGINE.VO_OSMP_VOME2_PLAYER;

   // Initialize SDK player
   nRet = m_sdkPlayer.init(eEngineType, init);

   // Set view
   m_sdkPlayer.setView(m_svMain);

   // Set surface view size
   DisplayMetrics dm = new DisplayMetrics();
   getWindowManager().getDefaultDisplay().getMetrics(dm);
   m_sdkPlayer.setViewSize(dm.widthPixels, dm.heightPixels);

   // Register SDK event listener
   m_sdkPlayer.setOnEventListener(m_listenerEvent);
   ...
   // Set device capability file location
   String capFile = cfgPath + "cap.xml";
   m_sdkPlayer.setDeviceCapabilityByFile(capFile);
   // Set license file location
   m_sdkPlayer.setLicenseFilePath(cfgPath);

   // Start playing the video
   ...
}

public static String getUserPath(Context context) {
      PackageManager m = context.getPackageManager();
   String path = context.getPackageName();
   String userPath = "/data/data/" + path;
   try {
       PackageInfo p = m.getPackageInfo(path, 0);
       userPath = p.applicationInfo.dataDir;
   } catch (NameNotFoundException e) {
   }
   return userPath;
}
```

3.5.1 Requirements and Recommendations

The SDK client shall:

- Initialize and configure the SDK instance following the task order listed in Table 3-1.

3.5.2 Opening Media Source

After the SDK is initialized, a source media file or streaming link must first be opened before starting playback. The `VOCommonPlayer.open()` method opens the source and prepares the media pipeline for playback. The `VOCommonPlayer.open()` method must be provided with:

- The source location (URL or file path) as a `String*` type.
- Source flags (currently, `VO_OSMP_SRC_FLAG.VO_OSMP_FLAG_SRC_OPEN_SYNC` and `VO_OSMP_SRC_FLAG.VO_OSMP_FLAG_SRC_OPEN_ASYNC` are supported).
- The source format (H.264, MP4, etc., or auto-detect) as a `VO_OSMP_SRC_FORMAT` type.
- Initialization parameters/flags (not currently used).

Media source can be opened in either Sync or Async Mode. Opening the media source in sync mode causes the client application to block until the API call to `VOCommonPlayer.open()` method completes/returns. In Async mode however, the API call returns immediately, and the client application is not blocked waiting for the media source to be opened. The client application can then wait for an event to indicate that the media source was opened successfully before starting the media playback.

Sample code:

```
public void surfaceCreated(SurfaceHolder surfaceholder) {
   // Initialize the player
   ...
   // Start playing the video
   // First open the media source
   // Auto-detect source format
   VO_OSMP_SRC_FORMAT format =
      VO_OSMP_SRC_FORMAT.VO_OSMP_SRC_AUTO_DETECT;
   // Set source flag to asynchronous
   VO_OSMP_SRC_FLAG eSourceFlag;
   eSourceFlag = VO_OSMP_SRC_FLAG.VO_OSMP_FLAG_SRC_OPEN_ASYNC;

   VOOSMPOpenParam openParam = new VOOSMPOpenParam();
   openParam.setDecoderType(VO_OSMP_DECODER_TYPE.VP_OSMP_DEC_VIDEO_SW.g
      etValue() | VO_OSMP_DECODER_TYPE.VO_OSMP_DEC_AUDIO_SW.getValue());

   //Open media source
   if (mRet = m_sdkPlayer.open(m_strVideoPath, eSourceFlag, format,
      openParam);
      ...
   }
   ...

}
```

3.5.3 Basic SDK Event Handling

When SDK events such as VO_OSMP_SRC_CB_OPEN_FINISHED occur, the SDK sends a status notification to the client through the SDK event listener. The SDK event listener, which is activated through the `VOCommonPlayer.setEventListener()` method as described in Section 3.5 (Initializing the SDK), sends event notifications as parameters through the `onVOEvent()` method, where user code then handles the event. The SDK client is responsible for handling all SDK events.

In the following example, the `onVOEvent()` method parses the event ID (`nID`) from the SDK. Upon receiving a `VO_OSMP_SRC_CB_OPEN_FINISHED` ID, the `onVOEvent()` method starts playback, and initializes any other user defined features such as Subtitle File Path etc.

Sample code:

```
private VOCommonPlayerListener m_ListenerEvent = new
   VOCommonPlayerListener(){
   // SDK Event Handling
   ...
   public VO_OSMP_RETURN_CODE onVOEvent(VO_OSMP_CB_EVENT_ID nID, int
      nParam1, int nParam2, Object obj) {
      switch(nID) {

      ...

      case VO_OSMP_SRC_CB_OPEN_FINISHED: { // Async Open Completed
         if (nParam1 == VO_OSMP_RETURN_CODE.VO_OSMP_ERR_NONE.getValue())
         {  // MediaPlayer is opened
            VO_OSMP_RETURN_CODE nRet;
            // Start (play) media pipeline
            mRet = m_sdkPlayer.start();
            ...
         } else {
            onError(m_sdkPlayer, nParam1, 0);
         }
         break;
      }
      ...
   }
   return VO_OSMP_RETURN_CODE.VO_OSMP_ERR_NONE;
}
```

3.5.4 More Information

For more information on the `VOCommonPlayer init()`, `setView()`, `setViewSize()`, `setOnEventListener()`, `setDeviceCapabilityByFile()`, and `setLicenseFilePath()` methods, refer to the *OnStream MediaPlayer+ Player API Reference Manual for Android Platforms*.

For more information on `DisplayMetrics`, refer to the *Android Developers Reference* (Display class).

3.6 STARTING PLAYBACK

To start playback, a source media file or streaming link must first be successfully opened by the SDK (as described in Section 3.5.2 – Opening Media Source). Once the media source has been opened, playback can be initiated or resumed using the `VOCommonPlayer.start()` method.

In the following example, the `surfaceCreated()` method, after initializing the SDK, opens the source link in Async Mode. An event indicating that the media source was opened (or an error event) is received in the `VOCommonPlayerListener()` method. If the media source file was opened successfully, then the event handler in the `onVOEvent()` method starts playback of the opened media source.

Sample code:

```
public void surfaceCreated(SurfaceHolder surfaceholder) {

   // Initialize the player

   ...

   // Start playing the video
   // First open the media source
   // Auto-detect source format
   VO_OSMP_SRC_FORMAT format =
      VO_OSMP_SRC_FORMAT.VO_OSMP_SRC_AUTO_DETECT;

   // Set source flag to asynchronous
   VO_OSMP_SRC_FLAG eSourceFlag;
   eSourceFlag = VO_OSMP_SRC_FLAG.VO_OSMP_FLAG_SRC_OPEN_ASYNC;

   VOOSMPOpenParam openParam = new VOOSMPOpenParam();
   openParam.setDecoderType(VO_OSMP_DECODER_TYPE.VP_OSMP_DEC_VIDEO_SW.g
      etValue() | VO_OSMP_DECODER_TYPE.VO_OSMP_DEC_AUDIO_SW.getValue());

   // Open media source
   if (mRet = m_sdkPlayer.open(m_strVideoPath, eSourceFlag, format,
      openParam);

      ...

   }

   ...

}
```

```
private VOCommonPlayerListener m_ListenerEvent = new
   VOCommonPlayerListener(){

   // SDK Event Handling

   ...

   public VO_OSMP_RETURN_CODE onVOEvent(VO_OSMP_CB_EVENT_ID nID, int
      nParam1, int nParam2, Object obj) {
      switch(nID) {

      case VO_OSMP_CB_ERROR:
      case VO_OSMP_SRC_CB_CONNECTION_FAIL:
      case VO_OSMP SRC_CB_DOWNLOAD FAIL:
      case VO_OSMP_SRC_CB_DRM_FAIL:
      case VO_OSMP_SRC_CB_PLAYLIST_PARSE_ERR:
      case VO_OSMP_SRC_CB_CONNECTION_REJECTED:
      case VO_OSMP_SRC_CB_DRM_NOT_SECURE:
      case VO_OSMP_SRC_CB_DRM_AV_OUT_FAIL:
      case VO_OSMP_SRC_CB_LICENSE_FAIL: { //Error
         // Display error dialog and stop player
         onError(m_sdkPlayer, nID.getValue(), 0);
         break;
         }

      case VO_OSMP_SRC_CB_OPEN_FIINISHED: { // Async Open Completed
         if (nParam1 == VO_OSMP_RETURN_CODE.VO_OSMP_ERR_NONE.getValue())
         {  // MediaPlayer is opened
            VO_OSMP_RETURN_CODE nRet;

            // Start (play) media pipeline
            mRet = m_sdkPlayer.start();
            if (nRet == VO_OSMP_RETURN_CODE.VO_OSMP_ERR_NONE) {
               // MediaPlayer started
               ...
            } else {
               onError(m_sdkPlayer, nRet.getValue(), 0);
            }
         } else {
            onError(m_sdkPlayer, nParam1, 0);
         }
         break;
      }
      ...
   }
   return VO_OSMP_RETURN_CODE.VO_OSMP_ERR_NONE;
}
```

3.6.1 Requirements and Recommendations

The SDK client shall:

- Open the media source before starting playback.

3.6.2 More Information

For more information on the `VOCommonPlayer open()` and `start()` methods, refer to the *OnStream MediaPlayer+ Player API Reference Manual for Android Platforms*.

3.7 STOPPING PLAYBACK

Playback is stopped using the `VOCommonPlayer.stop()` method. If playback will not be resumed (e.g., if playback is complete) or if a new source will be opened, then the current media source must also be closed using the `VOCommonPlayer.close()` method. When leaving the SDK client activity, the SDK must be uninitialized using the `VOCommonPlayer.destroy()` method. A basic SDK client may implement all three methods in a row whenever playback is stopped.

In the following example, the `surfaceDestroyed()` callback method, which is called when the surface is destroyed (e.g., when the **Home** button is pressed or when the activity leaves the foreground), stops playback, closes the media source, and uninitializes the SDK.

Sample code:

```
public void surfaceDestroyed(SurfaceHolder surfaceholder) {
   if (m_sdkPlayer !=null) {
      m_sdkPlayer.stop();
      m_sdkPlayer.close();
      m_sdkPlayer.destroy();
      m_sdkPlayer = null;
   }
}
```

Playback should also be stopped upon playback completion (when the media pipeline is empty) and on error conditions.

3.7.1 Playback Completion (SDK Event Handling)

When SDK events such as playback completion occur, the SDK sends a status notification to the client through the SDK event listener. The SDK event listener sends event notifications as parameters through the `onVOEvent()` method, where user code then handles the event. The SDK client is responsible for handling all SDK events.

In the following example, the `onVOEvent()` method parses the event ID (`nID`) from the SDK. Upon receiving a `VO_OSMP_CB_PLAY_COMPLETE` ID, the `onVOEvent()` method stops playback, closes the media source, and uninitializes the SDK.

Sample code:

```
public VO_OSMP_RETURN_CODE onVOEvent(VO_OSMP_CB_EVENT_ID nID, int nParam1,
   int nParam2, Object obj)
{
   switch(nID)
```

```
    {
        ...
        case VO_OSMP_CB_PLAY_COMPLETE:
        {
            if (m_sdkPlayer != null) {
                m_sdkPlayer.stop();
                m_sdkPlayer.close();
                m_sdkPlayer.destroy();
                m_sdkPlayer = null;
                this.finish();    // MediaPlayer is released
            }
        }
        ...
    }
    return VO_OSMP_RETURN_CODE.VO_OSMP_ERR_NONE;
}
```

3.7.2 Error Conditions (Basic Error Handling)

SDK errors are sent as events to the `onVOEvent()` method for user processing.

In the following example, the `onVOEvent()` method, upon receiving a `VO_OSMP_CB_ERROR`, `VO_OSMP_SRC_CB_CONNECTION_FAIL`, or `VO_OSMP_SRC_CB_CONNECTION_REJECTED` error ID, calls the `onError()` method. After displaying the error message and receiving user acknowledgement, the `onError()` method stops playback, closes the media source, and uninitializes the SDK.

Sample code:

```
public VO_OSMP_RETURN_CODE onVOEvent(VO_OSMP_CB_EVENT_ID nID, int nParam1,
    int nParam2, Object obj)
{
    switch(nID)
    {
        case VO_OSMP_CB_ERROR:
        case VO_OSMP_SRC_CB_CONNECTION_FAIL:
        case VO_OSMP_SRC_CB_CONNECTION_REJECTED:
        {
            // Display error dialog and stop the player
            onError(m_sdkPlayer, nID.getValue(), 0);
            break;
        }
        ...
    }
    return VO_OSMP_RETURN_CODE.VO_OSMP_ERR_NONE;
}

public boolean onError(VOCommonPlayer mp, int what, int extra) {

    Log.v(TAG, "Error message, what is " + what + " extra is " + extra);
    String errStr = getString(R.string.str_ErrPlay_Message)
          + "\nError code is " + Integer.toHexString(what);
```

```
        // Dialog to display error message;
        // stop player and exit on Back key or OK
        AlertDialog ad = new AlertDialog.Builder(Player.this)
           .setIcon(R.drawable.icon).setTitle(R.string.str_ErrPlay_Title)
           .setMessage(errStr).setOnKeyListener(new OnKeyListener() {
              public boolean onKey(DialogInterface arg0, int arg1,
                    KeyEvent arg2) {
                 if (arg1 == KeyEvent.KEYCODE_BACK) {
                    if (m_sdkPlayer != null) {
                       m_sdkPlayer.stop();
                       m_sdkPlayer.close();
                       m_sdkPlayer.destroy();
                       m_sdkPlayer = null;
                    }

                 }

                 return false;
              }
           }).setPositiveButton(R.string.str_OK, new OnClickListener() {
              public void onClick(DialogInterface a0, int a1) {
                 if (m_sdkPlayer != null) {
                    m_sdkPlayer.stop();
                    m_sdkPlayer.close();
                    m_sdkPlayer.destroy();
                    m_sdkPlayer = null;
                 }
              }

           }).create();

        ad.show();
        return true;
     }
```

The `onError()` method can also be called when SDK methods return error codes.

3.7.3 Requirements and Recommendations

The SDK client shall:

- Close the media source after stopping playback, if playback will not be resumed or if a new source will be opened.
- Uninitialize the SDK when leaving the activity.
- Manage SDK messages and errors.

3.7.4 More Information

For more information on the `VOCommonPlayer stop()`, `close()`, and `destroy()` methods and a complete listing of SDK events and error codes, refer to the *OnStream MediaPlayer+ Player API Reference Manual for Android Platforms*.

3.8 RESTARTING PLAYBACK

To restart playback from a source media file, playback is first stopped using the `VOCommonPlayer.stop()` method. There is no need to close the source file before restarting playback. The position needs to be reset by using the `VOCommonPlayer.setPosition(0)` method. Playback can now be initiated using the `VOCommonPlayer.start()` method.

3.9 MANAGING SURFACE CHANGES

The Android `SurfaceView` and `SurfaceHolder` provide the drawing surface for the SDK. The SDK must be notified of changes to the surface via `SurfaceHolder` callbacks, as shown in Table 3:2.

Table 3:2: SurfaceHolder Callback Notifications

SurfaceHolder Callback	Callback Description	SDK Actions
`surfaceChanged`	Notification of surface change.	Notify SDK of surface change.
`surfaceCreated`	Notification of surface creation (foreground).	Initialize SDK (if necessary) and (re)start playback.
`surfaceDestroyed`	Notification of surface destruction (background).	Stop/suspend playback and uninitialize SDK (if necessary).

The `surfaceCreated()` implementation for a basic SDK client is discussed in sections 3.5 and 3.6 (Initializing the SDK and Starting Playback). The `surfaceDestroyed()` implementation is discussed in section 3.7 (Stopping Playback).

In the following example, the `surfaceChanged()` method sends notification to the SDK via the `VOCommonPlayer.setSurfaceChangeFinished()` method.

Sample code:

```
public void surfaceChanged (SurfaceHolder surfaceholder, int format, int
   w, int h) {

   if (m_sdkPlayer != null)
      m_sdkPlayer.setSurfaceChangeFinished();
}
```

3.9.1 Requirements and Recommendations

The SDK client shall:

- Inform the SDK of surface changes when they occur.

3.9.2 More Information

For more information on the `surfaceChanged()`, `surfaceCreated()` and `surfaceDestroyed()` callbacks, refer to the *Android Developers Reference* (SurfaceHolder.Callback).

4 Advanced Integration: Pause/Play Control

This section describes the integration of pause/play control with the SDK. Pause/play control provides the basis for a simple user interface.

4.1 Integration Lab

This section uses examples from *Integration Lab #2*. The source code examples can be found at *<SDK_INSTALL_DIR>\Android\Doc\Labs\Lab2\src\com\visualon\LabPlayer\player.java*.

4.2 Integration Flow

Figure 4-1 below illustrates the integration flow of a basic SDK client (*Integration Lab #1*) with additional pause/play control.

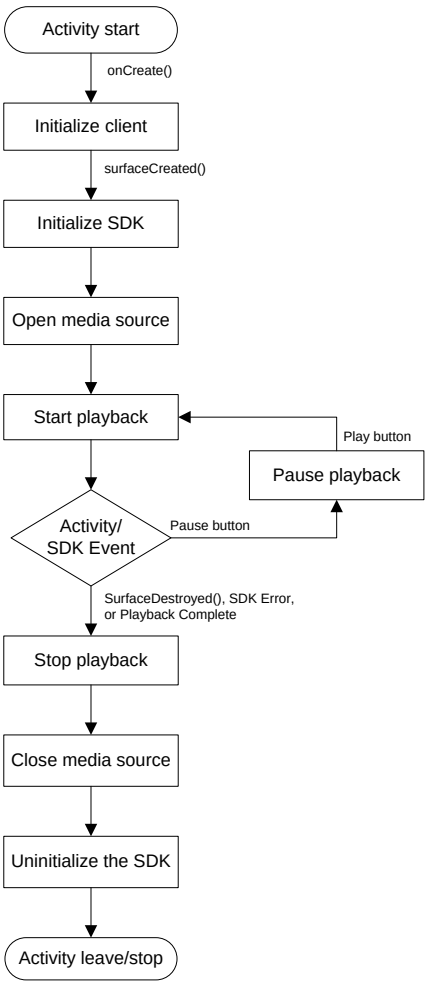

Figure 4-1: Flow Diagram for Basic SDK Client Plus Pause/Play Control

4.3 SDK CLIENT CLASS DEFINITIONS (IMAGEBUTTON INTERFACE)

Implementing pause/play functionality requires a basic user interface, which can be implemented using a shared button. The Android `ImageButton` class can be used to create a simple player control using custom images.

In the following example, the `Player` SDK client includes an `ImageButton` member to implement a shared Pause/Play button. A `m_bPaused` flag is also included to manage the shared button.

Sample code:

```
public class Player extends Activity implements SurfaceHolder.Callback {
   ...
   private SurfaceView m_svMain; // Drawing surface (must be passed to SDK)
   private VOCommonPlayer m_sdkPlayer = null; // SDK player

   // Media controls and User interface
   private ImageButton m_ibPlayPause; // Play/Pause button
   private boolean m_bPaused = false; // Pause flag private Boolean
   ...
}
```

4.4 SDK INTEGRATION

Pause/play functionality is integrated using the `VOCommonPlayer.pause()` and `VOCommonPlayer.start()` methods. The `VOCommonPlayer.pause()` method pauses playback of the media pipeline. The `VOCommonPlayer.start()` method restarts playback.

Some live streaming media sources cannot be paused. This can be verified using the `VOCommonPlayer.canBePaused()` method.

In the following example, the `playerPauseRestart()` method checks the `m_bPaused` flag, and then implements the complementary feature. Before pausing playback, the method first verifies that the media source can be paused. The `m_ibPlayPause` button image is also changed based on the flag value, updating the user interface.

Sample code:

```
private void playerPauseRestart() {
   if (m_sdkPlayer != null) {
      if (m_bPaused == false) {
         // If playing non-live streaming contents
         // Pause media pipeline and change button to "Play" icon
         if (m_sdkPlayer.canBePaused()) {
            m_sdkPlayer.pause();
            m_ibPlayPause.setImageResource(R.drawable.play_button);
            m_bPaused = true;
         }
      } else {
         // Else, play media pipeline and change button to "Pause" icon
```

```
            m_sdkPlayer.start();
            m_ibPlayPause.setImageResource(R.drawable.pause);
            m_bPaused = false;
        }
    }
}
```

4.4.1 Initiating Pause/Play (Implementing the ImageButton Interface)

The pause/play functionality must be called from the user interface when the `ImageButton` is clicked.

In the following example, the `onCreate()` method, which is called when an Android activity launches, registers the `playerPauseRestart()` method to be called when the `m_ibPlayPause` button is clicked.

Sample code:

```
public void onCreate(Bundle savedInstanceState)
{
    setContentView(R.layout.player);
    ...
    // Activate listener for Play/Pause button
    m_ibPlayPause = (ImageButton) findViewById(R.id.ibPlayPause);
    m_ibPlayPause.setOnClickListener(new ImageButton.OnClickListener() {
        public void onClick(View view) {
            playerPauseRestart();
        }
    });
    ...
}
```

4.4.2 Requirements and Recommendations

The SDK client shall:

- Verify if playback can be paused before calling `VOCommonPlayer.pause()`.

4.4.3 More Information

For more information on `ImageButton` and `setImageResource`, refer to the *Android Developers Reference* (ImageButton).

5 Advanced Integration: +/- Seek Control

This section describes the integration of a +/-5 second seek control with the SDK through a button interface. +/- seek control enables the user to advance and rewind playback from the current position.

Note: +/- seek control applies only to local file playback. For live streaming and progressive download applications, refer to section 6 (Advanced Integration: Seekbar Control).

5.1 INTEGRATION LAB

This section uses examples from *Integration Lab #3a*. The source code examples can be found at *<SDK_INSTALL_DIR>\Android\Doc\Labs\Lab3a\src\com\visualon\LabPlayer\player.java*.

5.2 INTEGRATION FLOW

Figure 5-1 below illustrates the integration flow of a basic SDK client (*Integration Lab #1*) with additional +/- seek control.

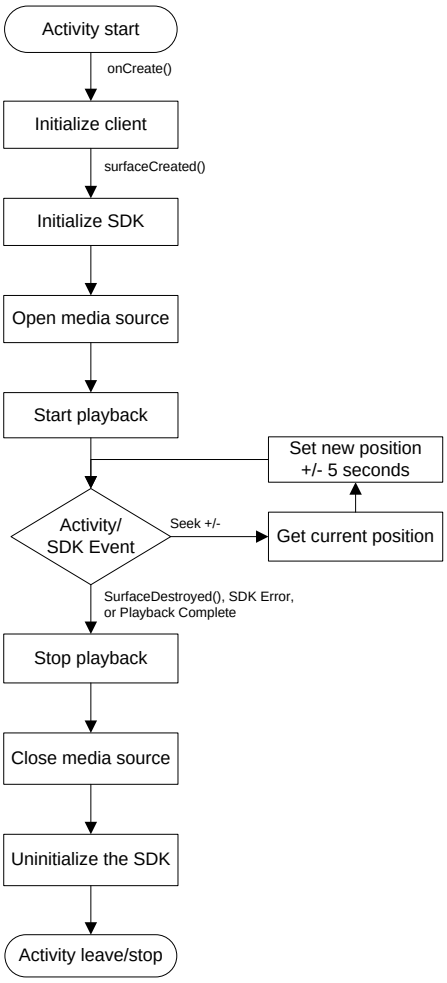

Figure 5-1: Flow Diagram for Basic SDK Client Plus +/- Seek Control

5.3 SDK CLIENT CLASS DEFINITIONS (BUTTON INTERFACE)

Implementing +/- seek functionality requires a basic user interface, which can be implemented using buttons to advance or rewind the playback position. The Android `Button` class can be used to create simple player controls with text labels.

In the following example, the `Player` SDK client includes two `Button` members to implement the forward and backward seek buttons.

Sample code:

```
public class Player extends Activity implements SurfaceHolder.Callback {
   ...
   private Button    m_bFwd5s; // Forward seek 5s button
   private Button    m_bBwd5s; // Backward seek 5s button
   ...
}
```

5.4 SDK INTEGRATION

+/- seek functionality is integrated using the `VOCommonPlayer.getDuration()`, `VOCommonPlayer.getPosition()`, and `VOCommonPlayer.setPosition()` methods. Table 5-1 describes each method and its usage in +/- seek integration.

Table 5-1: SDK +/- Seek Integration Methods

VOCommonPlayer Method	Description/Integration
`getDuration()`	Retrieves the total duration of the media source. Seek functionality is only supported if `getDuration() > 0`.
`getPosition()`	Retrieves the current playback position.
`setPosition()`	Sets the new (seek) playback position. Playback immediately jumps to the new position (does not need to be restarted).

In the following example, the `onSeekingTouch5s()` method retrieves the current playback position, and then checks the `bIsForward` flag to determine if the seek action is forward or backward. If the duration is greater than 0, the new position is calculated (between 0 and the total duration) and set in the SDK.
Sample code:

```
public void onSeekingTouch5s(boolean bIsForward) {
   // Get duration of the media
   int m_nDuration = (int) m_sdkPlayer.getDuration();
   // Get current position
   int m_nPos = (int) m_sdkPlayer.getPosition();
   long lNewPosition = 0;
   // Seek is enabled only when Duration is great than 0s
   if (m_nDuration > 0) {
      if (bIsForward)
         lNewPosition = m_nPos + 5000;
      else
         lNewPosition = m_nPos - 5000;
      if (lNewPosition > m_nDuration)
         lNewPosition = m_nDuration;
      else if (lNewPosition < 0)
         lNewPosition = 0;
      if (m_sdkPlayer != null) {
         m_sdkPlayer.setPosition(lNewPosition); // Set new position
      }
   }
}
```

5.4.1 Initiating +/- Seek (Implementing the Button Interface)

The +/- seek functionality must be called from the user interface when the appropriate `Button` is clicked.

In the following example, the `onSeekForward5s()` and `onSeekBackward5s()` methods are wrappers that call the `onSeekingTouch5s()` method, which implements the seek. The `onCreate()` method, which is called when an Android activity launches, registers `onSeekForward5s()` and `onSeekBackward5s()` as the methods to be called when the `m_bFwd10s` or `m_bFwd10s` buttons (respectively) are clicked.

Sample code:

```
protected void onSeekForward5s()
{
   if (m_sdkPlayer!= null) {
      onSeekingTouch5s(true);
   }
}
protected void onSeekBackward5s()
{
   if (m_sdkPlayer!= null) {
      onSeekingTouch5s(false);
   }
}
public void onCreate(Bundle savedInstanceState)
{
   setContentView(R.layout.player);
   ...
   // Activate listener for Seek Fwd 5s button
   m_bFwd5s = (Button) findViewById(R.id.bSeekFwd5s);
   m_bFwd5s.setOnClickListener(new Button.OnClickListener() {
      public void onClick(View view) {
         onSeekForward5s();
      }
   });
   // Activate listener for Seek Bwd 5s button
   m_bBwd5s = (Button) findViewById(R.id.bSeekBwd5s);
   m_bBwd5s.setOnClickListener(new Button.OnClickListener() {
      public void onClick(View view) {
         onSeekBackward5s();
      }
   });
   ...
}
```

5.4.2 More Information

For more information on `Button`, refer to the *Android Developers Reference* (Button).

6 Advanced Integration: Seekbar Control

This section describes the integration of a seekbar with the SDK. A seekbar displays the current playback position and enables the user to advance and rewind the playback to any position within the seekable area. For local files, the seekable area includes the total file duration; for live streaming and progressive download applications, the seekable area includes the current buffer or downloaded area.

6.1 INTEGRATION LAB

This section uses examples from *Integration Lab #3b*. The source code examples can be found at *<SDK_INSTALL_DIR>\Android\Doc\Labs\Lab3b\src\com\visualon\LabPlayer\player.java*.

6.2 INTEGRATION FLOW

Figure 6-1 below illustrates the integration flow of a basic SDK client (*Integration Lab #1*) with additional seekbar control.

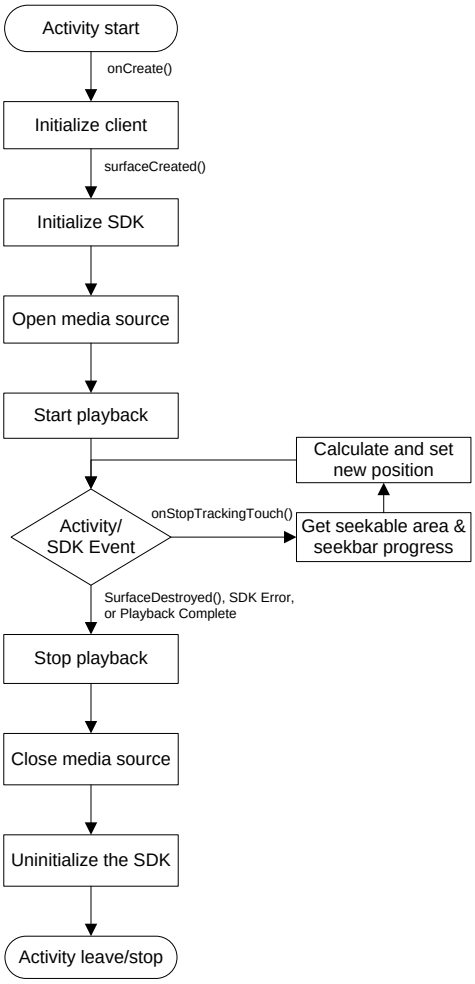

Figure 6-1: Flow Diagram for Basic SDK Client Plus Seekbar Control

6.3 SDK CLIENT CLASS DEFINITIONS (SEEKBAR INTERFACE)

Implementing seekbar functionality requires a user interface to allow the user to dynamically select a new playback position. The Android `SeekBar` object can be used to create the player control. The `SeekBar onStopTrackingTouch()` callback, which is called when the user releases the seekbar, can be used to initiate the seek.

In the following example, the `Player` SDK client includes a `SeekBar` member and implements `SeekBar.OnSeekBarChangeListener`, including the `onStopTrackingTouch()`

method, which indicates when the seekbar has been released. The `Player` class also includes a flag (`m_bTrackProgressing`) to indicate when the seekbar is being dragged by the user. Sample code:

```
public class Player extends Activity implements SurfaceHolder.Callback,
   SeekBar.OnSeekBarChangeListener {
   ...
   private SeekBar   m_sbMain; // Seekbar
   ...
   private boolean   m_bTrackProgressing = false; // Seekbar flag
   ...
   public void onStopTrackingTouch(SeekBar arg0)
   {
      ...
   }
}
```

6.4 SDK INTEGRATION

Seekbar functionality is integrated using the tasks and methods listed in Table 6-1.

Table 6-1: SDK Seekbar Integration Tasks

Task	VOCommonPlayer Method(s)	Description
Calculate the Seekbar Progress	`N/A`	Seekbar progress percentage is calculated using the `SeekBar.getProgress()` and `SeekBar.getMax()` values.
Determine the Seekable Area	`getDuration() > 0`	The seekable area is between 0 and `getDuration()`.
	`getDuration() <= 0`	The seekable area is between `getMinPosition()` and `getMaxPosition()`.
	`getMinPosition()`	Retrieve the minimum available position from the SDK.
	`getMaxPosition()`	Retrieve the maximum available position from the SDK.
Calculate and Set New Playback Position	`setPosition()`	New position is calculated as a percentage of the seekable area. Playback immediately jumps to the new position (does not need to be restarted).

In the following example, the `onStopTrackingTouch()` method implements the tasks listed in Table 6-1.

Sample code:

```
public void onStopTrackingTouch(SeekBar arg0)
{
    m_bTrackProgressing = false; // Disable Seekbar tracking flag
```

```
    // Calculate new position as percentage of total duration
    int iCurrent = arg0.getProgress();
    int iMax = arg0.getMax();
    long m_nDuration = m_sdkPlayer.getDuration();

    long lNewPosition;
    if (m_nDuration > 0) {
        lNewPosition = iCurrent * m_nDuration / iMax;
    }
    else {
        long nMinPos = m_sdkPlayer.getMinPosition();
        long nMaxPos = m_sdkPlayer.getMaxPosition();
        int nDuration = (int) (nMaxPos - nMinPos);
        lNewPosition = iCurrent * nDuration / iMax;
        lNewPosition = lNewPosition + nMinPos;
    }

    if (m_sdkPlayer != null)
    {
       Log.v(TAG,"Seek To " + lNewPosition);
       m_sdkPlayer.setPosition(lNewPosition); // Set new position
    }
}
```

6.4.1 Initiating the Seek (Registering the SeekBar Listener)

The seek functionality is initiated when the `SeekBar` is released (after being dragged). This is indicated to the SDK client via the `onStopTrackingTouch()` callback from the `SeekBar.setOnSeekBarChangeListener` listener.

In the following example, the `onCreate()` method, which is called when an Android activity launches, registers the listener to the SDK client's seekbar (`m_sbMain`) button is clicked.

Sample code:

```
public void onCreate(Bundle savedInstanceState)
{
   setContentView(R.layout.player);
   ...
   m_sbMain = (SeekBar) findViewById(R.id.sbMain);
   ...
   m_sbMain.setOnSeekBarChangeListener(this);
   ...
}
```

6.4.2 Updating the Seekbar Display (Handler Interface)

The seekbar user interface differs from button-based interfaces in that its display should be updated periodically with the current playback position, which can be retrieved using the `VOCommonPlayer.getPosition()` method. In order to update the `SeekBar` progress using the `SeekBar.setProgress()` method, the current position must be provided as a percentage of the total seekable area.

In the following example, the `Handler.handleMessage()` method, which processes internal messages, updates the seekbar display with the current playback position upon receiving the `MSG_UPDATE_UI` message. The current position is calculated relative to the total duration for local files (`getDuration() > 0`), or to the minimum/maximum position boundaries for live stream and progressive download media sources (`getDuration() <= 0`).

Sample code:

```
// Messages for managing the user interface
private static final int MSG_UPDATE_UI = 3;
   ...
private Handler handler = new Handler() {
// Handler to manage user interface during playback

   public void handleMessage(Message msg)
   {
      if(m_sdkPlayer == null)
         return;

      if (msg.what == MSG_UPDATE_UI) {
         // Update UI
         ...
         // Update the Seekbar and Time display with current position
         long m_nDuration = m_sdkPlayer.getDuration();
         long m_nPos = m_sdkPlayer.getPosition();

         if (m_nDuration <= 0)
         {
            ...
            int nDuration = (int) (m_sdkPlayer.getMaxPosition() -
               m_sdkPlayer.getMinPosition());
            int nPos = (int) (m_nPos - m_sdkPlayer.getMinPosition());

            m_sbMain.setProgress(100 * nPos/ nDuration);
         }
         else {
            m_sbMain.setProgress((int)(100 * m_nPos/ m_nDuration));
            ...
         }
      }
   }
}
```

6.4.3 Scheduling Seekbar Display Updates (Timer and TimerTask Interface)

The seekbar display requires periodic updates. The Android `Timer` and `TimerTask` objects can be used to schedule periodic tasks.

In the following example, the `onVOEvent()` method, which begins playback when the media player is opened successfully, creates a new `TimerTask` to send the `MSG_UPDATE_UI` message to the `handler` to update the seekbar. A `timer` is used to the schedule repeated calls to the `TimerTask` every 0.2 seconds.

Sample code:

```
private Timer     mTimer = null;
private TimerTask tmTask = null;
...
public VO_OSMP_RETURN_CODE onVOEvent(VO_OSMP_CB_EVENT_ID nID, int
   nParam1, int nParam2, Object obj) {

   switch)nID) {
   ...
   case VO_OSMP_SRC_CB_OPEN_FINISHED: {
   // Start playing the video
   ...
   // Start timer to update seekbar
   if(tmTask!=null)
   {
      tmTask = null;
   }
   tmTask= new TimerTask()
   {
      public void run()
      {
         handler.sendEmptyMessage(MSG_UPDATE_UI);
      }
   }
   if(mTimer == null)
   {
      mTimer = new Timer();
   }
   mTimer.schedule(tmTask, 0, 200);
   }
   ...
}
```

6.4.4 More Information

For more information on `SeekBar`, `Handler`, `Timer`, and `TimerTask`, refer to the *Android Developers Reference* (SeekBar, Handler, Timer, and TimerTask).

7 Advanced Integration: Channel Switching

This section describes the integration of channel switching with the SDK. Channel switching enables the user to switch media sources (e.g., streaming links) from within the SDK client.

7.1 Integration Lab

This section uses examples from *Integration Lab #4*. The source code examples can be found at *<SDK_INSTALL_DIR>\Android\Doc\Labs\Lab4\src\com\visualon\LabPlayer\player.java*.

7.2 Integration Flow

Figure 7-1 below illustrates the integration flow of a basic SDK client (*Integration Lab #1*) with additional channel switching.

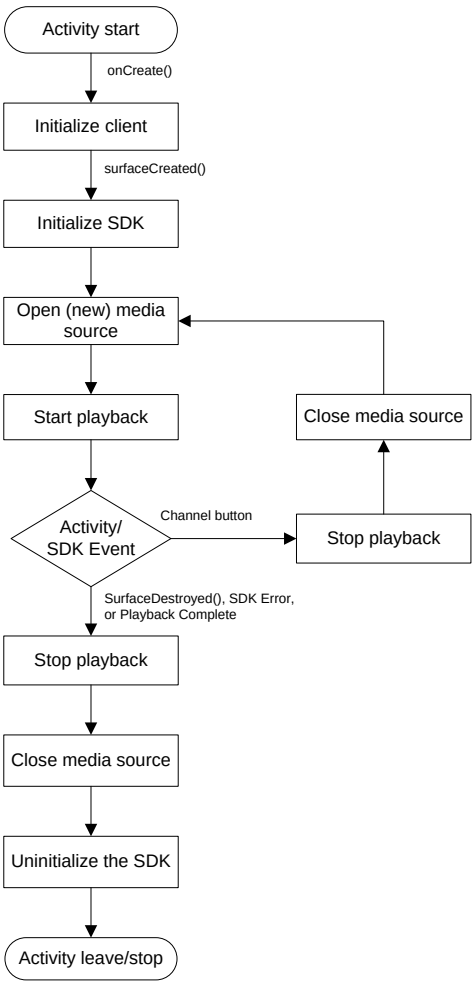

Figure 7-1: Flow Diagram for Basic SDK Client Plus Channel Switching

7.3 SDK CLIENT CLASS DEFINITIONS (BUTTON INTERFACE)

Implementing channel switching functionality requires a basic button interface. The Android `Button` class can be used to create simple player controls with text labels.

In the following example, the `Player` SDK client includes a `Button` member to implement the channel switch button.

Sample code:

```
public class Player extends Activity implements SurfaceHolder.Callback {
    ...
private Button    m_bChannelSwitch; // Channel toggle button
...
}
```

7.4 SDK INTEGRATION

Channel switching is integrated using the following procedure:

1. Stop playback and close the current media source.
2. Open the new media source and start playback.

Stopping playback and closing the media source is discussed in section 3.7 (Stopping Playback). Opening a source and starting playback is discussed in section 3.6 (Starting Playback).

In the following example, the `channelswitch()` method, which is called when the `m_bChannelSwitch` button is pressed, implements channel switching between two sources. It toggles the channel (`m_bIsChannelOne`) and updates the source location (`m_strVideoPath`), then stops playback and closes the current media source. The new source is opened and playback is started.

Sample code:

```
private boolean m_bIsChannelOne      = true;
...
private void channelSwitch() {

   VO_OSMP_RETURN_CODE nRet;

   if (m_bIsChannelOne) {
      m_strVideoPath = m_strVideoPath_Two;
      m_bIsChannelOne = false;
   } else {
      m_strVideoPath = m_strVideoPath_One;
      m_bIsChannelOne = true;
   }

   if (m_sdkPlayer != null) {
      // Stop the current playback and close source
```

```
        m_sdkPlayer.stop();
        m_sdkPlayer.close();
        ...

        // First open the media source
        VO_OSMP_SRC_FORMAT format
              = VO_OSMP_SRC_FORMAT.VO_OSMP_SRC_AUTO_DETECT;

        VO_OSMP_SRC_FLAG eSourceFlag;
        eSourceFlag = VO_OSMP_SRC_FLAG.VO_OSMP_FLAG_SRC_OPEN_ASYNC;

        VOOSMPOpenParam openParam = new VOOSMPOpenParam();
        openParam.setDecoderType(VO_OSMP_DECODER_TYPE.VO_OSMP_DEV_VIDEO_S
           W.getValue() |
           VO_OSMP_DECODER_TYPE.VO_OSMP_DEC_AUDIO_SW_getValue());
        ...

        }
}

// Look for VO_OSMP_SRC_CB_OPEN_FINISHED Event
public VO_OSMP_RETURN_CODE onVOEvent(VO_OSMP_CB_EVENT_ID nID, int nParam1,
   int nParam2, Object obj) {
   switch(nID) {
   ...
   case VO_OSMP_SRC_CB_OPEN_FINISHED: {
      // MdeiaPlayer opened successfully
      ...
      // Start (play) media pipeline
      nRet = m_sdkPlayer.start();
      ...
   }
}
```

7.4.1 Requirements and Recommendations

The SDK client shall:

- Stop and close the current source before opening the new source.

7.4.2 More Information

For more information on `Button`, refer to the *Android Developers Reference* (Button).

8 Advanced Integration: Enabling Closed Captions and Subtitles

This section describes the integration of closed captions (CC) and subtitles rendering with the SDK. CC/subtitles can be automatically rendered from an embedded or external source.

8.1 Integration Lab

This section uses examples from *Integration Lab #5a*. The source code examples can be found at *<SDK_INSTALL_DIR>\Android\Doc\Labs\Lab5a\src\com\visualon\LabPlayer\player.java*.

8.2 Integration Flow

Figure 8-1 below illustrates the integration flow of a basic SDK client (*Integration Lab #1*) with additional CC/subtitles rendering.

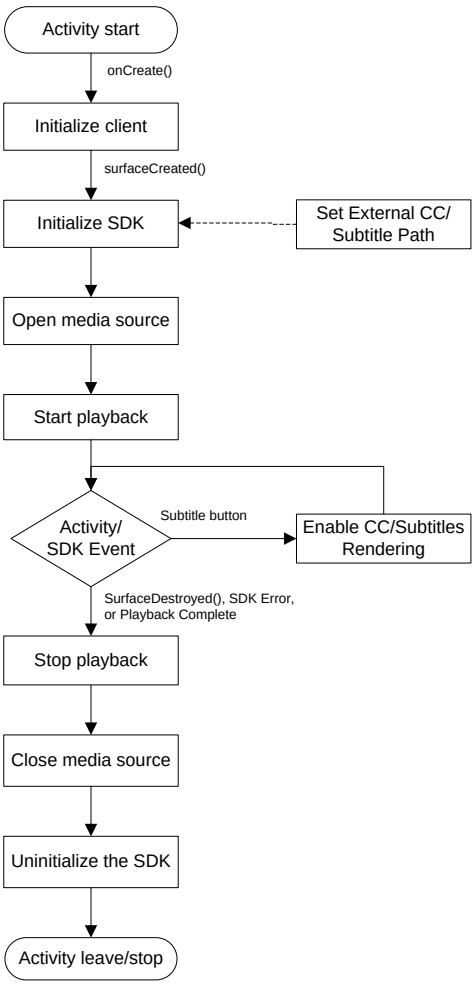

Figure 8-1: Flow Diagram for Basic SDK Client Plus CC/Subtitles Rendering

8.3 SDK CLIENT CLASS DEFINITIONS (BUTTON INTERFACE)

Implementing CC/subtitles rendering requires a basic interface. The Android `Button` class can be used to create simple player controls with text labels.

In the following example, the `Player` SDK client includes a `Button` member to implement a button that toggles CC/subtitles rendering. A `m_bSubtitleToggle` flag is also included to manage the button functionality.

Sample code:

```
public class Player extends Activity implements SurfaceHolder.Callback {
   ...
   private Button m_bSubtitleToggle; // Subtitle Enable/Disable button
   private boolean m_bSubtitleEnable = false; // Subtitle Enable/Disable
      flag
   ...
}
```

8.4 SDK INTEGRATION

The built-in SDK support for CC/Subtitles rendering is easily enabled/disabled using the `VOCommonPlayer.enableSubtitle()` method.

In the following example, the `subtitleToggle()` method, which is called when the `m_bSubtitleToggle` button is pressed, toggles the subtitle flag (`m_bSubtitleEnable`), and then enables the CC/subtitles rendering in the SDK.

Sample code:

```
private boolean m_bSubtitleEnable = false; // Subtitle Enable/Disable flag
...
public void subtitleToggle() {
   // Toggle subtitle enable/disable flag
   m_bSubtitleEnable = !m_bSubtitleEnable;
   if (m_sdkPlayer != null) {
      // Enable/Disable subtitles
      m_sdkPlayer.enableSubtitle(m_bSubtitleEnable);
      ...
   }
}
```

8.4.1 Rendering CC/Subtitles from an External File

By default the SDK will render the CC/subtitles information that is embedded in the media source. This can be overridden by providing the SDK with an external file (e.g. **.srt*, **.smi*) using the `VOCommonPlayer.setSubtitlePath()` method.

In the following example, the `onCreate()` method, which is called when an Android activity launches and is used to initialize the SDK client, identifies the CC/subtitles source using the

`m_strExternalSubtitlePath` variable. The `surfaceCreated()` method initializes the SDK and opens the media source, and the `onVOEvent()` method, which begins playback when the media source is opened successfully, sets the external CC/subtitle path in the SDK just after playback begins.

Sample code:

```
private String m_strExternalSubtitlePath = ""; // External Subtitle Path
...
public void onCreate(Bundle savedInstanceState)
{
   // Initialize the SDK client
   ...
   // Define your playback URL/local media file here
   m_strVideoPath = "/* video asset with embedded subtitles */";
   // Define your external subtitle path here (if needed)// Define your
   external subtitle path here (if needed)
   m_strExternalSubtitlePath = "/* external subtitle file */";
   ...
}

public void surfaceCreated(SurfaceHolder surfaceholder) {
   ...
   // Initialize the SDK player
   ...
   // Open the media source in Async mode
   VO_OSMP_SRC_FLAG eSourceFlag;
   eSourceFlag = VO_OSMP_SRC_FLAG.VO_OSMP_FLAG_SRC_OPEN_ASYNC;

   VOOSMPOpenParam openParam = new VOOSMPOpenParam();
   openParam.setDecoderType(VO_OSMP_DECODER_TYPE.VO_OSMP_DEC_VIDEO_SW.g
   etValue() | VO_OSMP_DECODER_TYPE.VO_OSMP_DEC_AUDIO_SW.getValue());

   nRet = m_sdkPlayer.open(m_strVideoPath, eSourceFlag, format,
   openParam);

   ...

}

public VO_OSMP_RETURN_CODE onVOEvent(VO_OSMP_CB_EVENT_ID nID, int nParam1,
   int nParam2, Onject obj) {
   // Handle Events
   switch(nID) {
   ...
   case VO_OSMP_SRC_CB_OPEN_FINISHED: {
      ...
      if (nParam1 == VO_OSMP_RETURN_CODE.VO_OSMP_ERR_NONE.getValue()) {
         ...
         VO_OSMP_RETURN_CODE nRet;

         // Start (play) media pipeline
         nRet = m_sdkPlayer(start();
         ...
```

```
            // If an external subtitle file path is defined, set it;
            // otherwise, embedded subtitles will be used
            if (m_strExternalSubtitlePath.Length() > 0) {
               m_sdkPlayer.setSubtitlePath(m_strExternalSubtitlePath);
            }
            ...
         }
            ...
      }
   }
```

Note: In the original Lab #5a implementation, `m_strExternalSubtitlePath` is not set.

8.4.2 More Information

For more information on `Button`, refer to the *Android Developers Reference* (Button).

9 Advanced Integration: Video Track Switching

This section describes the integration of video track switching with the SDK. Track switching enables the playback of different video tracks from within a single media source.

Note: A video track is defined as a combination of angle and bit rate.

9.1 INTEGRATION LAB

This section uses examples from *Integration Lab #6a*. The source code examples can be found at *<SDK_INSTALL_DIR>\Android\Doc\Labs\Lab6a\src\com\visualon\LabPlayer\player.java*.

9.2 INTEGRATION FLOW

Figure 9-1 below illustrates the integration flow of a basic SDK client (*Integration Lab #1*) with additional video track switching.

Figure 9-1: Flow Diagram for Basic SDK Client Plus Video Track Switching

9.3 SDK CLIENT CLASS DEFINITIONS (BUTTON INTERFACE)

Implementing video track switching functionality requires a basic interface. The Android `Button` class can be used to create simple player controls with text labels.

In the following example, the `Player` SDK client includes two `Button` members: one to retrieve a list of video tracks, and another to advance to the next available track.

Sample code:

```
public class Player extends Activity implements SurfaceHolder.Callback {
    ...
    private Button m_bNextVideo; // Advance video button
    private Button m_bGetVideo; // Get video track info button
    ...
}
```

9.4 SDK INTEGRATION

Video track switching is integrated using the tasks and methods listed in Table 9-1.

Table 9-1: Video Track Switching Integration Tasks

Task	VOCommonPlayer Method(s)	Description
Get the total track count	`getVideoCount()`	Retrieve total number of video tracks. Used to provide maximum value when advancing the track.
Get the current track	`getPlayingAsset()`	Retrieve a `VOOSMPAssetIndex` object with the current video, audio, and subtitle indexes.
	`VOOSMPAssetIndex.getVideoIndex()`	Retrieve the current video index. `INDEX_VIDEO_AUTO` indicates automatic selection—start at index 0.
Find next available track	`isVideoAvailable()`	Advance index (loop to 0 if maximum reached) and check if the track is available. If no new track is available, `return` without changing track.
Select track	`selectVideo()`	Select the next available track in the SDK.
Commit selection	`commitSelection()`	Commit the new track selection in the SDK. The new video will be played immediately.

In the following example, the `nextVideo()` method, which is called when the `m_bNextVideo` button is pressed, advances to the next available video track using the procedure listed in Table 9-1.

Sample code:

```
public void nextVideo() {
    ...
    VO_OSMP_RETURN_CODE nRet;
    // Get number of video tracks from SDK
    int nVideoCount = m_sdkPlayer.getVideoCount();

    // Get current video index
    VOOSMPAssetIndex assetIndex= m_sdkPlayer.getPlayingAsset();
```

```
    int nCurrentIndex = assetIndex.getVideoIndex();
    int nNewIndex = nCurrentIndex;

    // Find next available track
    boolean bFoundNextAvailableTrack = false;
    // Loop until next track is found; or until you have reached the original
       track
    int i = 0;
    while ((!bFoundNextAvailableTrack) && (i < nVideoCount-1)) {
       i++;
       nNewIndex++;
       if (nNewIndex >= nVideoCount) {
          // Loop to first track
          nNewIndex = 0;
       }
       if ((m_sdkPlayer.isVideoAvailable(nNewIndex))) {
          // Track is available
          bFoundNextAvailableTrack = true;
       }
    }

    if (!bFoundNextAvailableTrack) {
       // If there is only one track, nothing to do
       ...
          return;
    }

    // Select new video track
    nRet = m_sdkPlayer.selectVideo(nNewIndex);
    ...

    // Commit selection
    nRet = m_sdkPlayer.commitSelection();
    ...
}
```

9.4.1 Retrieving Video Track Properties

Each video track contains embedded properties that are retrievable by the SDK client. Track properties, such as the description and codec, are managed using key/property string pairs in a `VOOSMPAssetProperty` object, retrieved using the `VOCommonPlayer.getVideoProperty()` method. Keys are retrieved from the `VOOSMPAssetProperty` object using the `getKey()` method, and properties are retrieved using the `getProperty()` method. Valid keys for video tracks include:

- “description”
- “codec”
- “bitrate”
- “width”
- “height”

In the following example, the `getVideo()` method, which is called when the `m_bGetVideo` button is pressed, retrieves the description of each video track. For each video track, it retrieves the track properties, and then loops through each key until it finds the "description." The corresponding property is then added to the list.

Sample code:

```
public void getVideoList()
{
   ArrayList<String> lstString = new ArrayList<String>();
   // Get number of video tracks from SDK
   int nVideoCount = m_sdkPlayer.getVideoCount();
   ...

   // Populate list with track descriptions
   for (int nAssetIndex = 0; nAssetIndex < nVideoCount; nAssetIndex++) {
         // Get track properties
      VOOSMPAssetProperty propImpl
         =m_sdkPlayer.getVideoProperty(nAssetIndex);

      String strDescription = "";
      int nPropertyCount = propImpl.getPropertyCount();
      boolean bPropertyDescription = false;
      for (int i = 0; i < nPropertyCount; i++) {
         // Look for the description
         String strPropertyKey = propImpl.getKey(i);
         if (strPropertyKey.equals("description")) {
            bPropertyDescription = true;
            strDescription = (String) propImpl.getValue(i);
         }
      }
      ...
      // Add description to the list
      lstString.add(strDescription);
   }
   ...
}
```

9.4.2 Requirements and Recommendations

The SDK client shall:

- Verify that the desired track is available before selecting/committing.

9.4.3 More Information

For more information on `Button`, refer to the *Android Developers Reference* (Button).

10 Advanced Integration: Suspend/Resume

This section describes the integration of suspend/resume functionality with the SDK. This functionality enables the SDK client to suspend playback when leaving the foreground (e.g., for an incoming call or due to the **Home** button being pressed), and resume playback when re-entering.

10.1 INTEGRATION LAB

This section uses examples from *Integration Lab #7*. The source code examples can be found at *<SDK_INSTALL_DIR>\Android\Doc\Labs\Lab7\src\com\visualon\LabPlayer\player.java*.

10.2 INTEGRATION FLOW

Figure 10-1 below illustrates the integration flow of a basic SDK client (*Integration Lab #1*) with additional suspend/resume functionality.

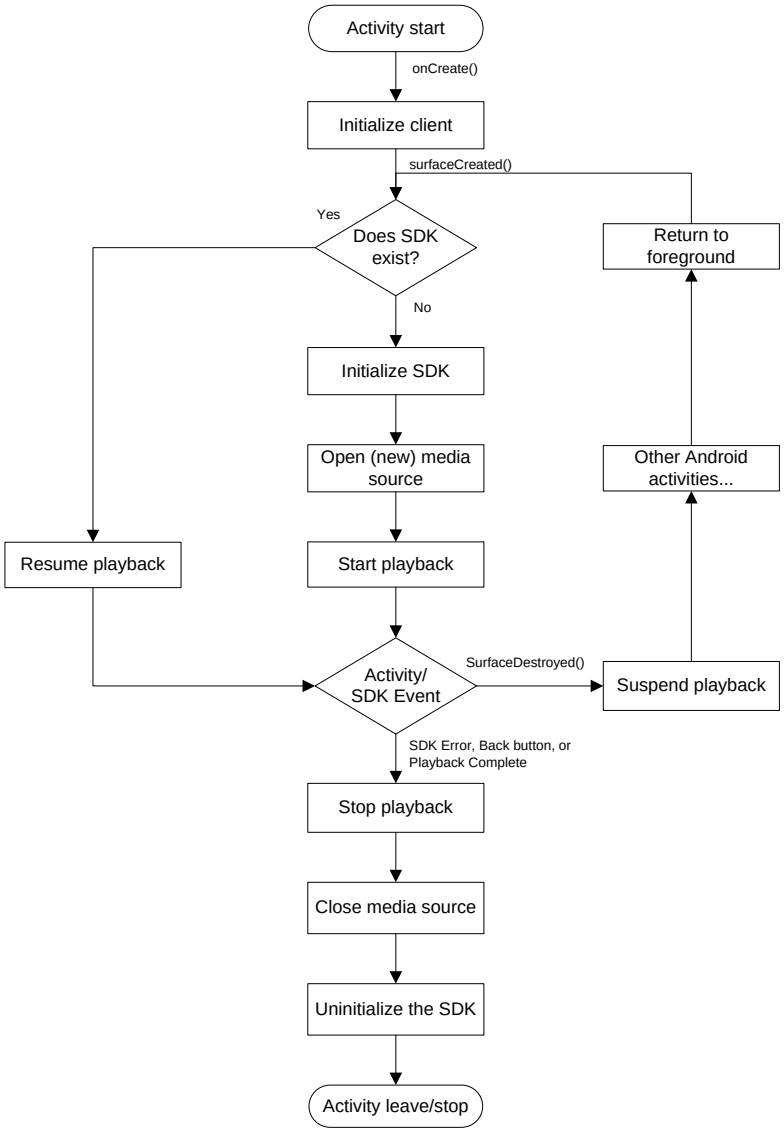

Figure 10-1: Flow Diagram for Basic SDK Client Plus Suspend/Resume

10.3 SDK CLIENT CLASS DEFINITIONS (SURFACEHOLDER CALLBACKS)

Implementing suspend/resume functionality requires the implementation of the Android surfaceHolder callbacks surfaceCreated() and surfaceDestroyed(). These callbacks are discussed in section 3.9 (Managing Surface Changes).

In the following example, the `Player` SDK client class definition implements `SurfaceHolder.Callback`.

Sample code:

```
public class Player extends Activity implements SurfaceHolder.Callback {
    ...
}
```

10.4 SDK INTEGRATION

10.4.1 Suspending Playback (surfaceDestroyed)

Suspend functionality is integrated using the `VOCommonPlayer.suspend()` method. This method suspends video playback, keeping the media source open and SDK framework available. Audio playback can optionally be suspended as well, or continue while the video playback is suspended.

In the following example, the `surfaceDestroyed()` method, which is called when the Android `Activity` leaves the foreground, suspends the video but continues audio playback.

Sample code:

```
public void surfaceDestroyed(SurfaceHolder surfaceholder) {

    ...

    if (m_sdkPlayer != null) {
        // Suspend video playback but continue audio; change to "false" to
           suspend audio
        boolean bAudioContinue = true;
        m_sdkPlayer.suspend(bAudioContinue);
        ...
    }
}
```

10.4.2 Resuming Playback (surfaceCreated)

Resume functionality is integrated using the `VOCommonPlayer.resume()` method. This method resumes video and audio (if also suspended) playback.

In the following example, the `surfaceCreated()` method, which typically initializes the SDK and begins playback, now includes a check to verify if the SDK already exists. If so, the playback is resumed.

Sample code:

```
public void surfaceCreated(SurfaceHolder surfaceholder) {
    ...
    if (m_sdkPlayer != null){
        m_sdkPlayer.resume(m_svMain);
```

```
    }
    else {
        // Initialize the SDK
        ...
        // Open media source (in Async mode) and wait for completion event.
        ...
    }
        ...
}
public VO_OSMP_RETURN_CODE onVOEvent(VO_OSMP_CB_EVENT_ID nID, intParam1,
    int nParam2, Object obj) {
    switch(nID) {
    ...
    case VO_OSMP_SRC_CB_OPEN_FINISHED: {
        ...
        // Start playing the video after media source is opened successfully
        ...
    }
    ...
}
```

Note: When returning to the foreground, an Android `activity` recreates the `surfaceView`, which must be provided to the SDK. The `surfaceView` is provided through the `VOCommonPlayer.resume()` method and not through the `VOCommonPlayer.setView()` method used during SDK initialization.

10.4.3 Stopping Playback (onKeyDown)

The `surfaceDestroyed()` method will suspend playback whenever the surface is destroyed, reagardless of the reason. Therefore, it is necessary to provide an alternative means to stop playback and exit the SDK client when desired. Stopping playback and closing the media source is discussed in section 3.7 (Stopping Playback).

In the following example, the `onKeyDown()` method, which is called when the Android `Activity` receives a key event, stops playback and exits the SDK client when the **Back** button is pressed.

Sample code:

```
// Stop player and exit on Back key
public boolean onKeyDown(int keyCode, KeyEvent event) {
    if (keyCode == KeyEvent.KEYCODE_BACK) {
        if (m_sdkPlayer != null) {
            m_sdkPlayer.stop();
            m_sdkPlayer.close();
            m_sdkPlayer.destroy();
            m_sdkPlayer = null;
        }
        return super.onKeyDown(keyCode, event);
    }
    return super.onKeyDown(keyCode, event);
}
```

10.4.4 Requirements and Recommendations

The SDK client shall:

- Provide the `surfaceView` to the SDK through the `VOCommonPlayer.resume()` method.
- Implement a means to stop playback to exit the application.

10.4.5 More Information

For more information on the `surfaceCreated()` and `surfaceDestroyed()` callbacks, refer to the *Android Developers Reference* (SurfaceHolder.Callback).

11 APK File Generation

This section briefly describes the generation of Application Package (APK) files for the Integration Labs.

11.1 Application Package File

Android Application Package (APK) File is one of the file formats used to distribute and install applications onto Android devices. To generate APK files, applications are first built, and then components such as resources, libraries, certificates, assets, manifest, etc. are packaged into an APK file for distribution.

11.2 APK File Build

After a successful compilation and build, the project is ready to be exported as an Application Package (APK) file. To build an APK file:

1. Right click on the project root folder in the Package Explorer name (for example "*Lab1*") and then select **Export** from the context sensitive menu

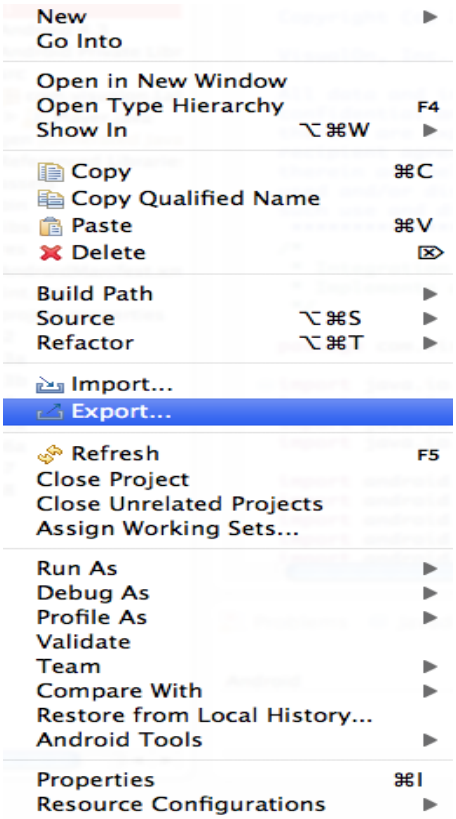

Figure 11-1: Menu Item to Select to Start Exporting

2. Select **Android->Export Android Application** and click on **Next**

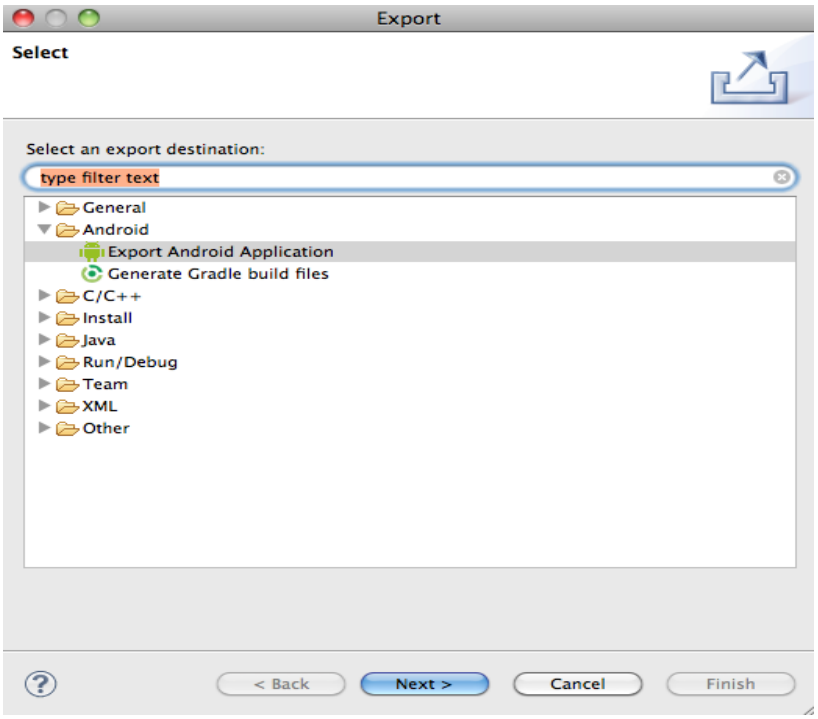

Figure 11-2: Next Menu Item to Select

3. Select the project to export and click on **Next**

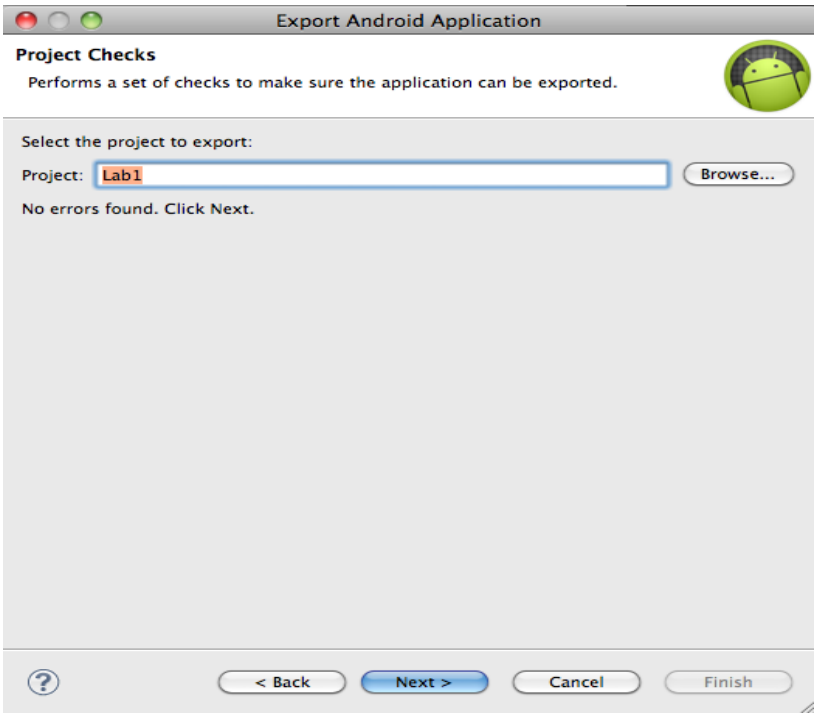

Figure 11-3: Entering the Project Name

4. Create new keystore or use an existing keystore and click on **Next**

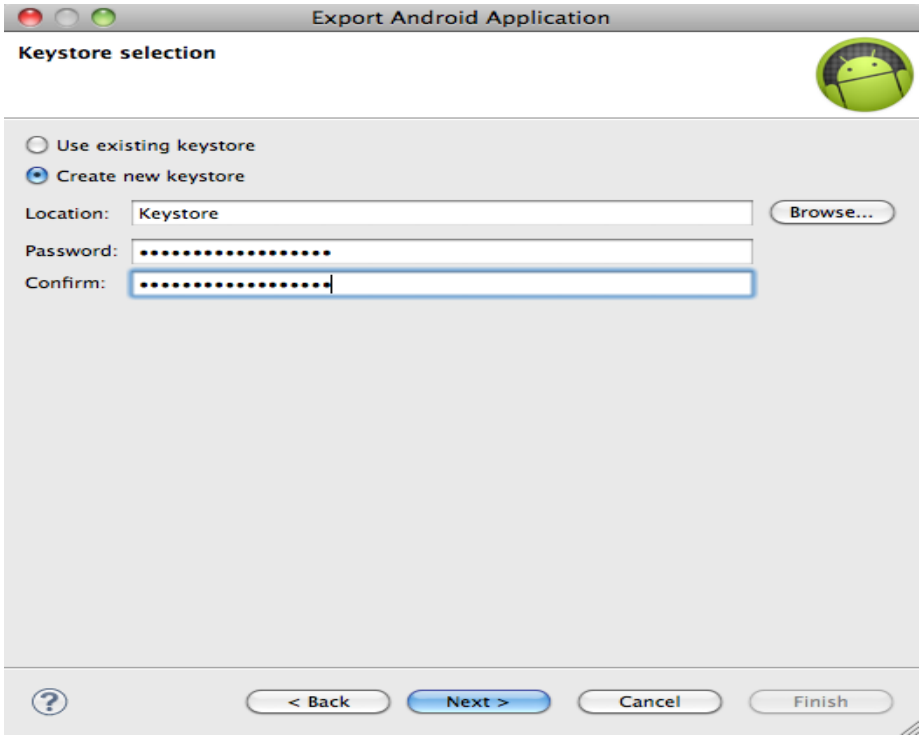

Figure 11-4: Creating the Keystore

5. Create new key or use an existing key and click on **Next**

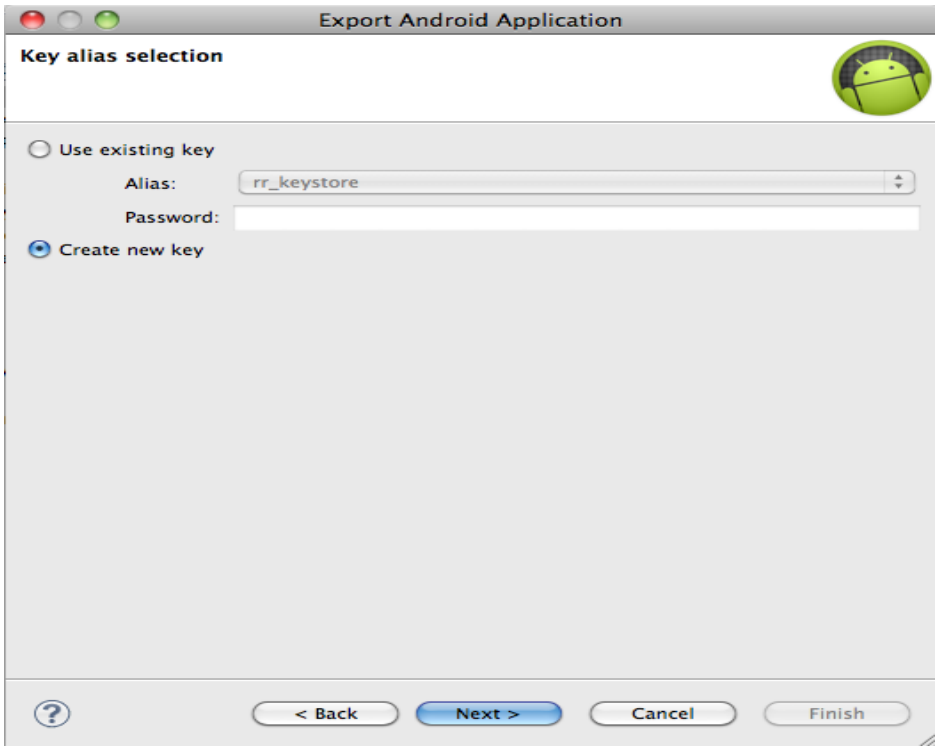

Figure 11-5: Creating a New Key

6. Select a destination to store the APK file and click on **Finish**. Key and Certificate checks are automatically performed and the Application Package (APK) file is created in the selected location.

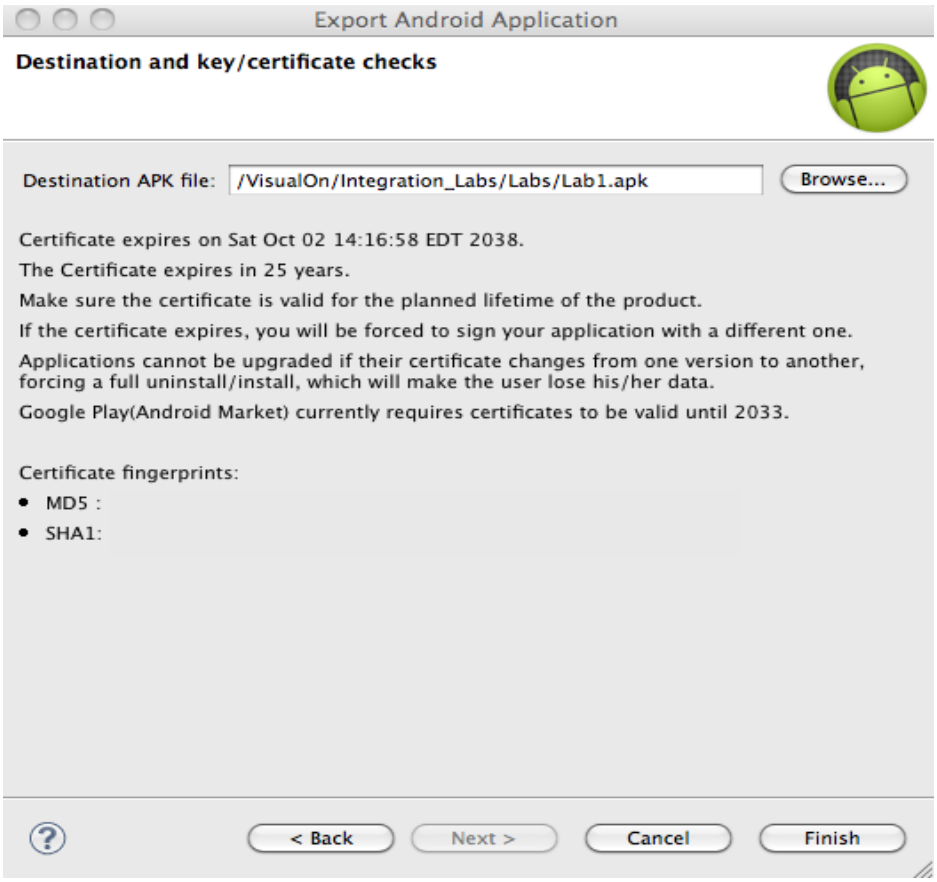

Figure 11-6: Final Screen When Generating the APK

12 Troubleshooting Guide

Problem	Possible Cause(s)	Action(s)
Flashing green screen during playback with V3.5 or earlier	License file has expired	Contact VisualOn for new license
Error code is VO_OSMP_ERR_LICENSE_FAIL with V3.6 or later	Received event VO_OSMP_CB_LICENSE_FAIL, notification that the licensing check failed	Contact VisualOn for new license