# Integrating CSP with Android®

Connected Sentinel Player 1.x

Document reference number: 21828

Document version number: 1.0 Release

# Contents

# Introduction

The *Integrating CSP with Android®* manual provides guidelines to integrate the Connected Sentinel Player within an application. It explains how to set up, configure and interface an Android project with the Viaccess-Orca Connected Sentinel Player SDK over Win32 using Eclipse Helios.

The interface described in this document is specific to Android®-based platforms.

## Target Audience

This document is intended for software developers writing an Android player application based on the Connected Sentinel Player.

### Glossary

This manual contains a lot of acronyms or terms that are specific to the field of Viaccess-Orca. If they are not defined within the text, refer to the *Appendix* on page 29 at the end of the manual for a complete definition.

## IDE Prerequisites

### Android SDK

Android SDK is installed on the development machine.

The following packages from the SDK manager must be installed:

- SDK Platforms for Android 2.1 and up (e.g., Android 2.2)
- Google® APIs by Google Inc. for Android API 7 and up

### Eclipse IDE

Eclipse® IDE Helios Version is installed on the development machine.

### Android ADT Plug-in

Android ADT Plug-in is installed via Eclipse IDE.

## Android Project Prerequisites

Android project must use Android 3.0 (API 11) or above.

# Chapter 1: Setup Procedure

This chapter informs you on how to proceed to prepare your integration project. It informs you on the content of the package received from Viaccess-Orca, and then describes step-by-step how to prepare your environment.

## Package Content

The Customer receives a ZIP file containing the JavaDoc APIS:

- Connected Sentinel Player SDK API Reference,
- Connected Sentinel Player SDK Common Integration Guide,
- The `libs` folder containing the Connected Sentinel Player SDK and a Connected Sentinel Player API Demo source code.

The relevant material to integrate the Connected Sentinel Player SDK is contained in the `libs` folder.

### Libs Folder Content

| **Component** | **Description** |
|---|---|
| `DxDrmDlc.jar` File | This file (Viaccess-Orca DRM Download Client) contains the Connected Sentinel Player Java API. |
| `voOSBasePlayer.jar` File | This file contains a part of the software player Java API. |
| `voOSBaseSource.jar` File | This file contains a part of the software player Java API. |
| `voOSEngine.jar` File | This file contains a part of the software player Java API. |
| `voOSHDMICheck.jar` File | This file contains a part of the software player Java API. |
| `voOSUtils.jar` File | This file contains a part of the software player Java API. |
| `assets.jar` File | This file is required for local personalization, for debug purposes only. |
| `armeabi` Subfolder | This folder contains native shared objects used by the Connected Sentinel Player Java API. |

## Unzip Package

1. Extract the `libs` folder from the ZIP provided.
2. Verify that the structure of the folder `libs` is not changed (see Package Content - Libs Folder Content).

# Create/Modify Android Project

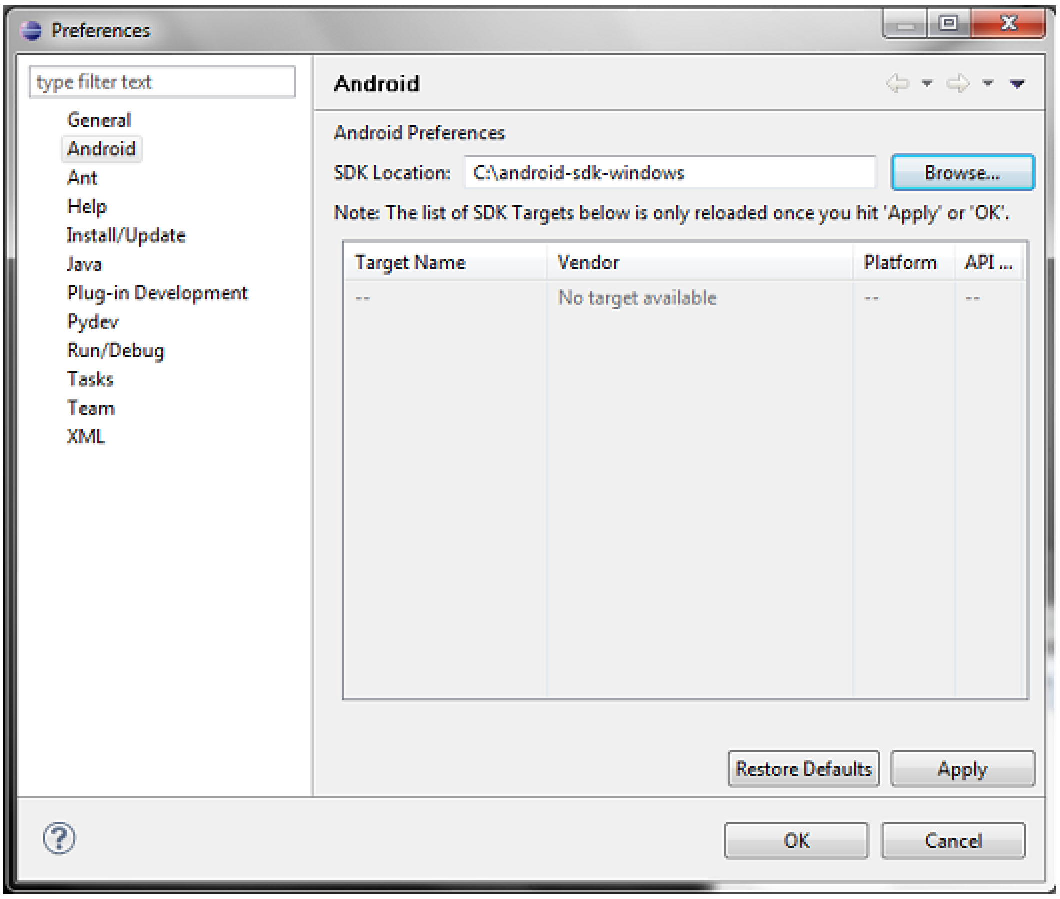
Listed below are steps to create/modify an Android Project that uses the Connected Sentinel Player API.

## Create an Android Project

1. Open Eclipse Helios IDE.

Verify that the SDK Location is set. To do so, proceed as follows:

2. Click **Menu: Window > Preference**.
3. Select **Android** on the left panel.
4. Type in the path of the Android SDK within the **SDK Location** text box.



Android Preferences

SDK Location: C:\android-sdk-windows

Note: The list of SDK Targets below is only reloaded once you hit 'Apply' or 'OK'.

| Target Name | Vendor | Platform | API ... |
|---|---|---|---|
| -- | No target available | -- | -- |

5. Click **Apply > OK**.

Open your Android project or create a new one. To create a new Android Project:

1. Click **File > New > Android Project**
2. Set:
   - Project name: Sample
   - Build target: Select Android 3.0
   - Application name: Sample
   - Package name: com.dxdrmdlc.sample
   - Create Activity: Sample
   - Min SDK Version: 7

3. Click **Finish**.

**note**

The values given for Application name, Package name and activity are just examples. These values may be changed according to the project specifics.

## Add Connected Sentinel Player API

### Create Libs Folder

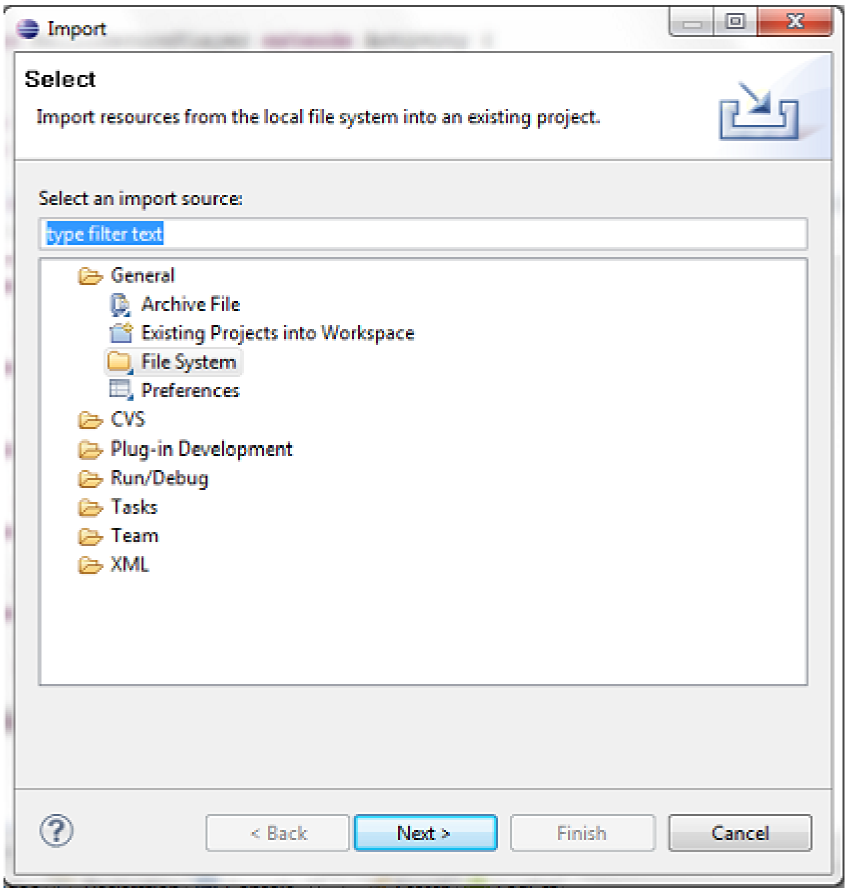
Check whether the `libs` folder exists in your project. If not, create it. To create the folder:

1. Display the **Package Explorer.**
2. Right-click the package root node (project name).
3. In the popup menu select **New > Folder.**
4. Add **Folder name**: `libs`.
5. Click **Finish**.

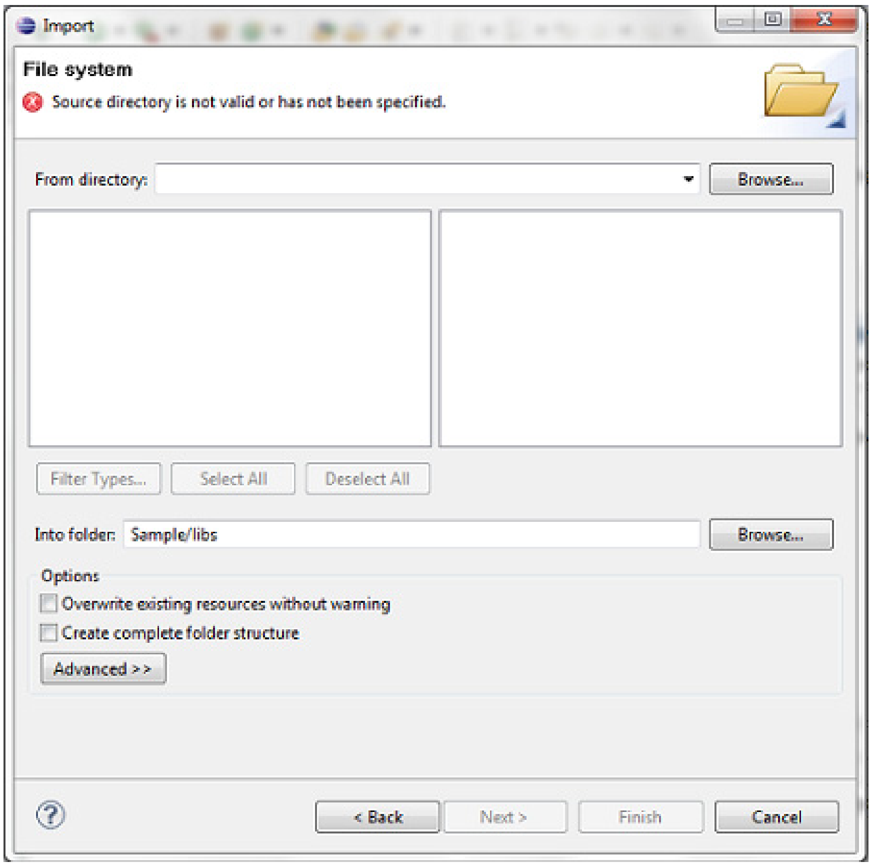
### Import Jar Files

To import .Jar files:

1. Right-click the `libs` Folder.
2. In the popup menu select **Import**.
3. In the **Import** window select **General > File System**.



4. Click **Next**.



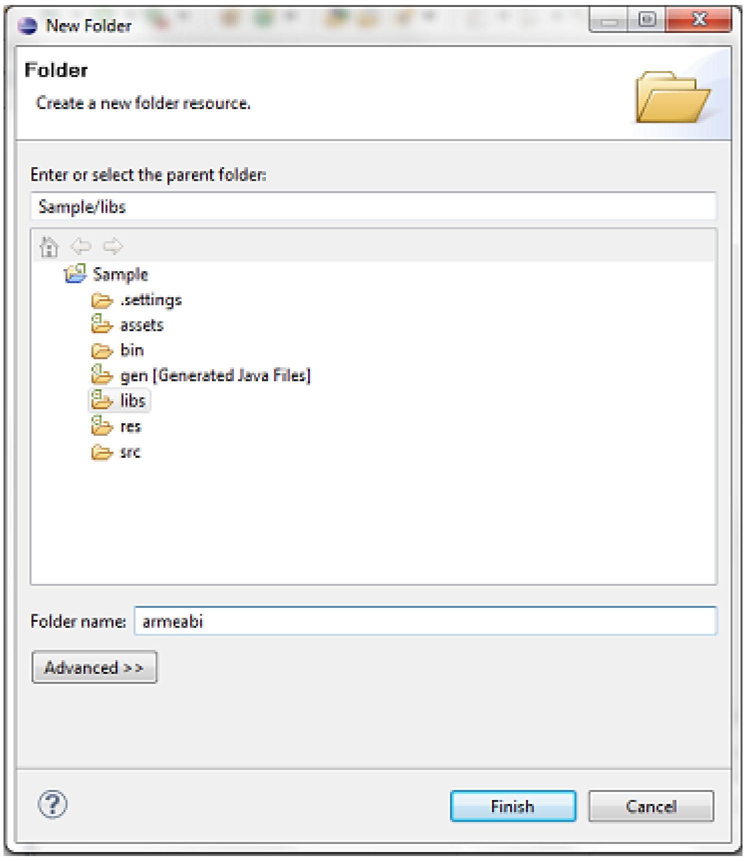
5. Browse to the location where you placed the libs folder that you unzipped from the provided Connected Sentinel Player SDK.
6. Select `assets.jar`, `DxDrmDlc.jar`, `voOSBasePlayer.jar`, `voOSDataSource.jar`, `voOSEngine.jar`, `voOSHDMICheck.jar` and `voOSUtils.jar.`
7. Click **Finish**.

## Import SO Files

To import `.SO` files:

1. Right-click the `libs` Folder.
2. In the popup menu select **New > Folder**

3. Add **Folder name:** `armeabi`.


New Folder

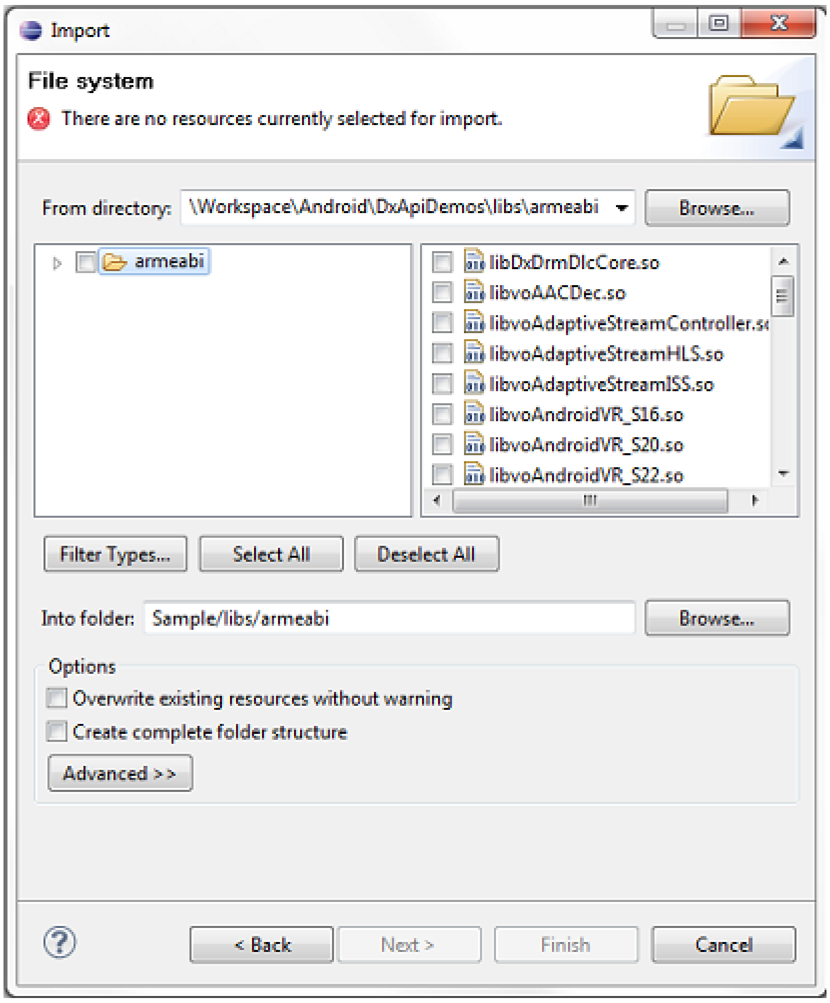
Folder
Create a new folder resource.

Enter or select the parent folder:
Sample/libs

Folder name: armeabi

4. Click **Finish.**
5. Right-click on the `armeabi` Folder.
6. In the popup menu select **Import**
7. In the Import window select **General > File System**
8. Browse to the location where you placed the `Libs/armeabi` folder that you unzipped from the provided Connected Sentinel Player SDK.

9. Click on **Select All** (select all `*.so` files).



There are no resources currently selected for import.

From directory: \Workspace\Android\DxApiDemos\libs\armeabi

armeabi

libDxDrmDlcCore.so
libvoAACDec.so
libvoAdaptiveStreamController.so
libvoAdaptiveStreamHLS.so
libvoAdaptiveStreamISS.so
libvoAndroidVR_S16.so
libvoAndroidVR_S20.so
libvoAndroidVR_S22.so

Into folder: Sample/libs/armeabi

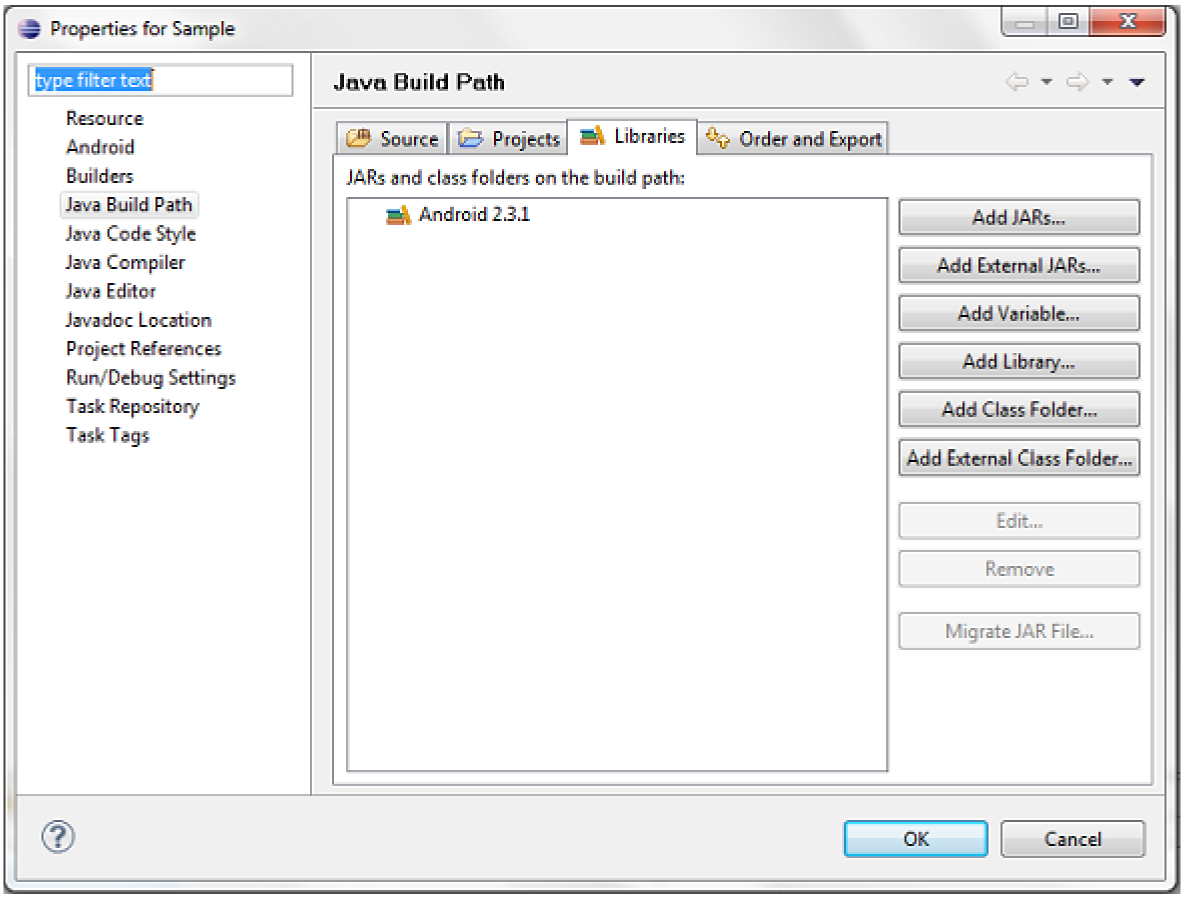
Overwrite existing resources without warning
Create complete folder structure

10. Click **Finish**.

## Set Project Properties

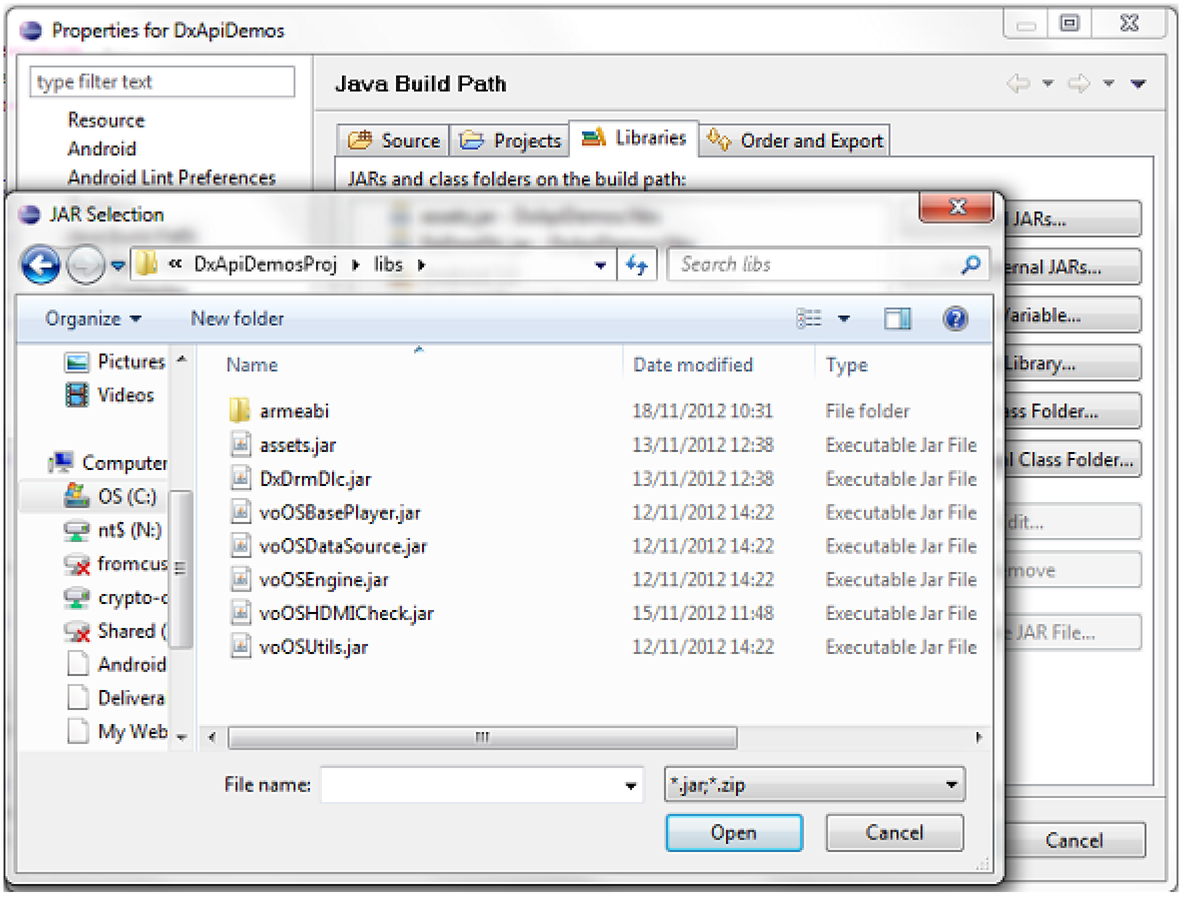
1. Right-click the package root node (project name).
2. In the popup menu select **Properties**.

3. Select **Java Build Path** on the left panel and then the **Libraries** Tab.



4. Click **Add JARs**.

5. Select the `assets.jar`, `DxDrmDlc.jar`, `voOSBasePlayer.jar`, `voOSDataSource.jar`, `voOSEngine.jar`, `voOSHDMICheck.jar` and `voOSUtils.jar` files under <Project name> -> libs


Properties for DxApiDemos

Java Build Path

JAR Selection

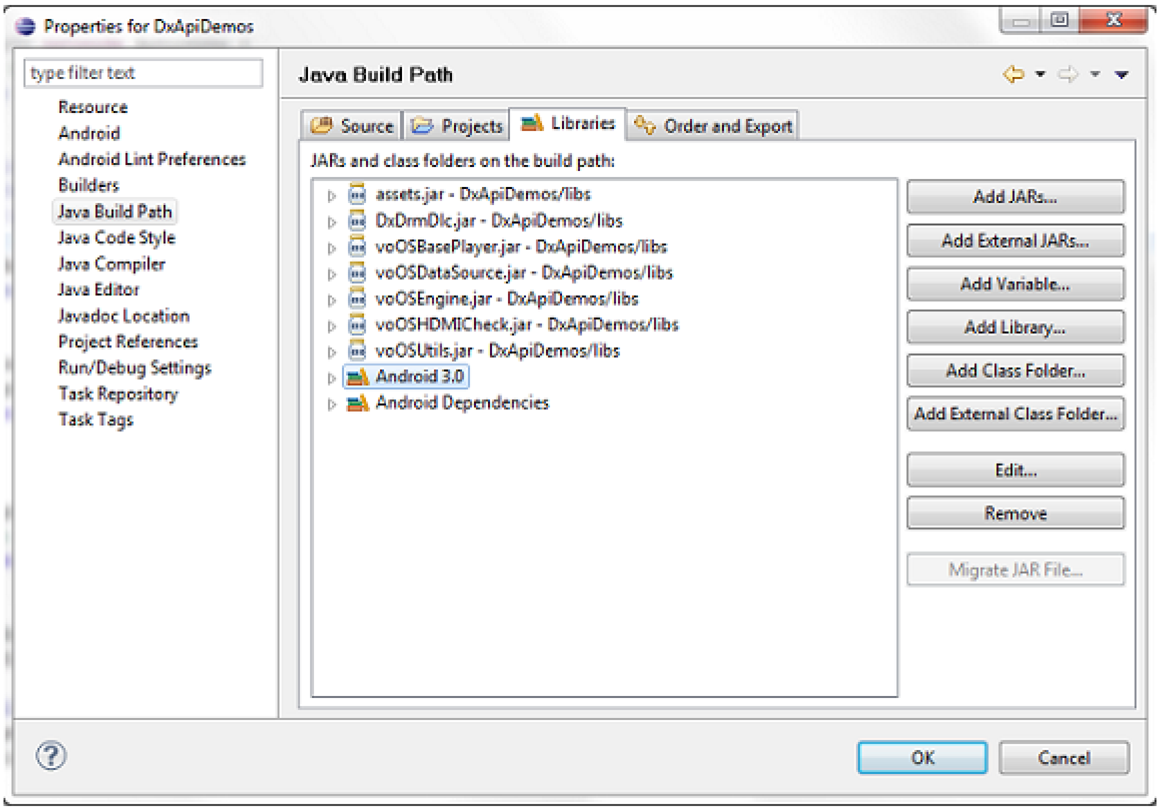
DxApiDemosProj › libs

| Name | Date modified | Type |
|---|---|---|
| armeabi | 18/11/2012 10:31 | File folder |
| assets.jar | 13/11/2012 12:38 | Executable Jar File |
| DxDrmDlc.jar | 13/11/2012 12:38 | Executable Jar File |
| voOSBasePlayer.jar | 12/11/2012 14:22 | Executable Jar File |
| voOSDataSource.jar | 12/11/2012 14:22 | Executable Jar File |
| voOSEngine.jar | 12/11/2012 14:22 | Executable Jar File |
| voOSHDMICheck.jar | 15/11/2012 11:48 | Executable Jar File |
| voOSUtils.jar | 12/11/2012 14:22 | Executable Jar File |

File name: *.jar;*.zip

Open Cancel

6. Click **OK** > **OK**.


Properties for DxApiDemos

Java Build Path

JARs and class folders on the build path:

- assets.jar - DxApiDemos/libs
- DxDrmDlc.jar - DxApiDemos/libs
- voOSBasePlayer.jar - DxApiDemos/libs
- voOSDataSource.jar - DxApiDemos/libs
- voOSEngine.jar - DxApiDemos/libs
- voOSHDMICheck.jar - DxApiDemos/libs
- voOSUtils.jar - DxApiDemos/libs
- Android 3.0
- Android Dependencies

7. Double click `AndroidManifest.xml` in the **Package Explorer.**
8. Add the following user permissions into the manifest, right above the application tag:

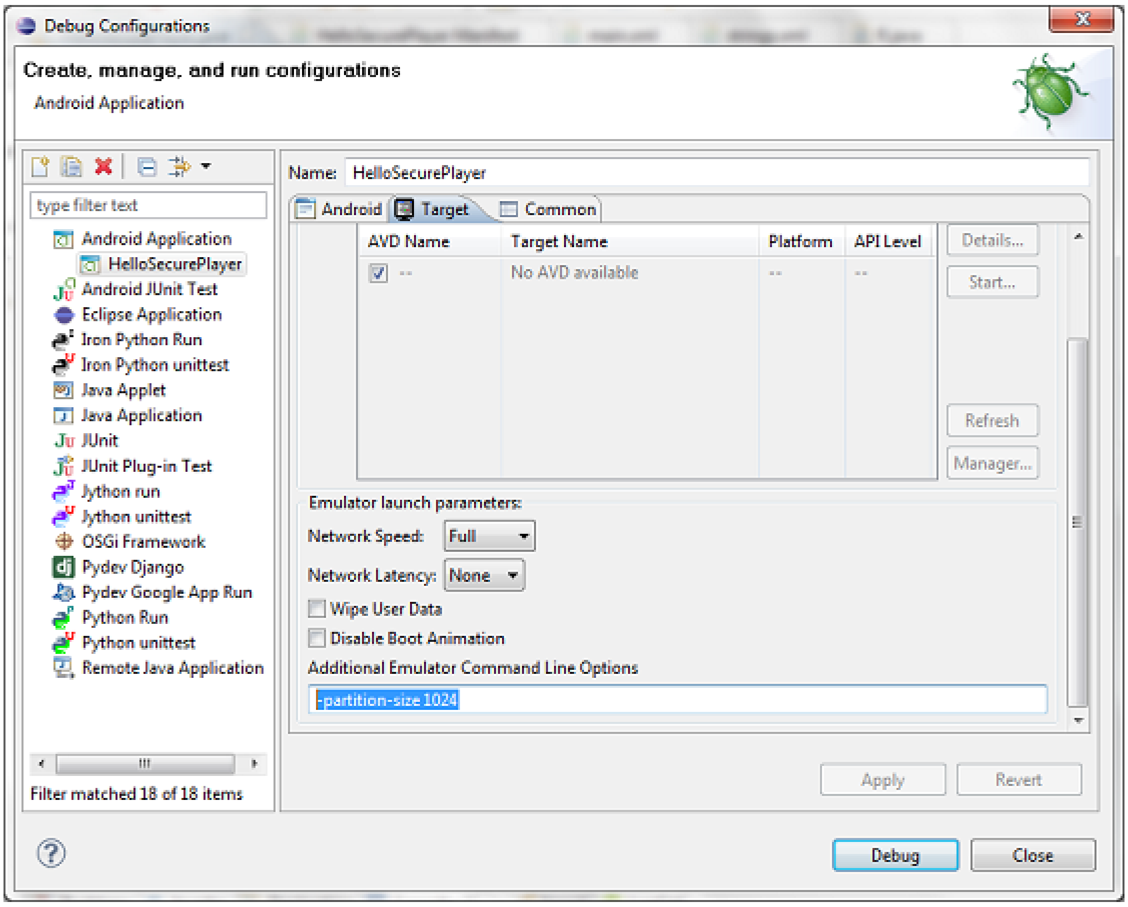
```
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
<uses-permission android:name="android.permission.ACCESS_WIFI_STATE" />
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
<uses-permission android:name="android.permission.WAKE_LOCK" />
<uses-permission android:name="android.permission.READ_PHONE_STATE"/>
<uses-permission android:name="android.permission.DOWNLOAD_WITHOUT_NOTIFICA-
TION"/>
<uses-permission android:name="android.permission.BLUETOOTH" />
<uses-permission android:name="android.permission.BLUETOOTH_ADMIN" />
```

9. Save the file (Ctrl+S).

# Troubleshooting

If you encounter the error `INSTALL_FAILED_INSUFFICIENT_STORAGE` during an attempt to debug the application, proceed as follows:

1. Select in menu: **Run > Debug Configuration**
2. Select Target tab.
3. In the **Additional Emulator Command Line Option** textbox type the following text: `-partition-size 1024`


Debug Configurations
Create, manage, and run configurations
Android Application

Name: HelloSecurePlayer

| AVD Name | Target Name | Platform | API Level |
|---|---|---|---|
| ✓ -- | No AVD available | -- | -- |

Emulator launch parameters:
Network Speed: Full
Network Latency: None
Wipe User Data
Disable Boot Animation
Additional Emulator Command Line Options
-partition-size 1024

Filter matched 18 of 18 items

4. Click **Apply> Close**

# Chapter 2: CSP-Client SDK

## Client side architecture

The following diagram illustrates the device application internal architecture:

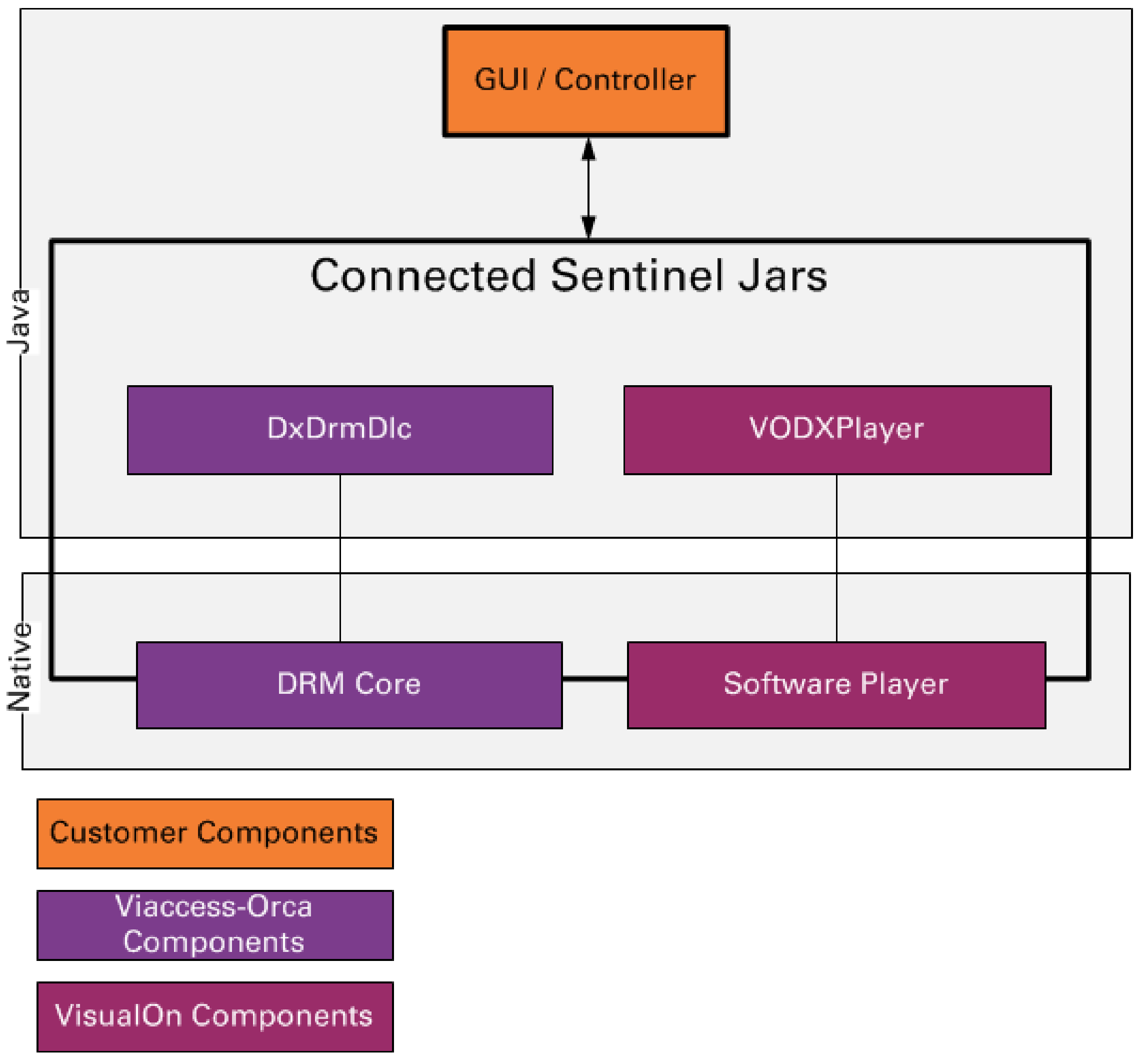


*figure 1.* Client Side Architecture

The diagram components are explained below:

- **GUI/Controller**: GUI front-end and logic for the client application, locally-stored content.
- Connected Sentinel **Player Jars (Java API)**: Java packages that provide the customer application for Android devices with interfaces to download and manage content licenses and content playback operation.
- **DRM core**: native library which implements the DRM core functionality.
- **Software Player:** native libraries which implement the video player that displays the content.

# Connected Sentinel Player integration

This section provides a guide when integrating the Viaccess-Orca CSP-SDK within the customer application (client-side) by use-cases related to PlayReady®. The following table provides the summary of such use-cases, along with references to the sections in this document describing the use-case and the relevant CSP-SDK components and interface.

| Types of Use Cases | Use cases |
|---|---|
| Personalization | *Personalization, Verification and Initiation* on page 18 |
| DRM management | *Rights acquisition* on page 20 |
| | *Retrieving License Details* on page 22 |
| | *Determine if Content is DRM-Protected* on page 23 |
| | *Determine if Content can be played* on page 24 |
| | *Deleting Content License* on page 24 |
| | *Retrieving DRM Version Details* on page 24 |
| SOAP | *SOAP Error Handling on page 24* |
| Content Playback | *Displaying video using VODXPlayer* on page 25 |
| Debug API | *Deleting Personalization Credentials* on page 27 |
| | *Configuring Logs* on page 27 |
| | *Set Client-Side Test Personalization Mode* on page 27 |

## Personalization

### Personalization, Verification and Initiation

Upon invocation, the application must verify the personalization status by calling the `DxDrmDlc.personalizationVerify()` method. This operation can be performed from any context. Successful personalization (`DxDrmDlc.personalizationVerify()` returns `True`) is a pre-condition for using any of the other CSP-SDK methods provided. If `False` is returned, personalization was not yet performed, and the application should call the `DxDrmDlc.performPersonalization()` method to accomplish it.

The `DxDrmDlc.performPersonalization()` method is synchronous. It initiates network operations and may block the caller until the operation is complete. It is recommended to call it from a separate thread, to preserve the application responsiveness.

The following figure shows the interactions between the Client Application, the CSP-SDK, and the Personalization server in a fresh activation of the client (e.g., the device was not personalized yet)

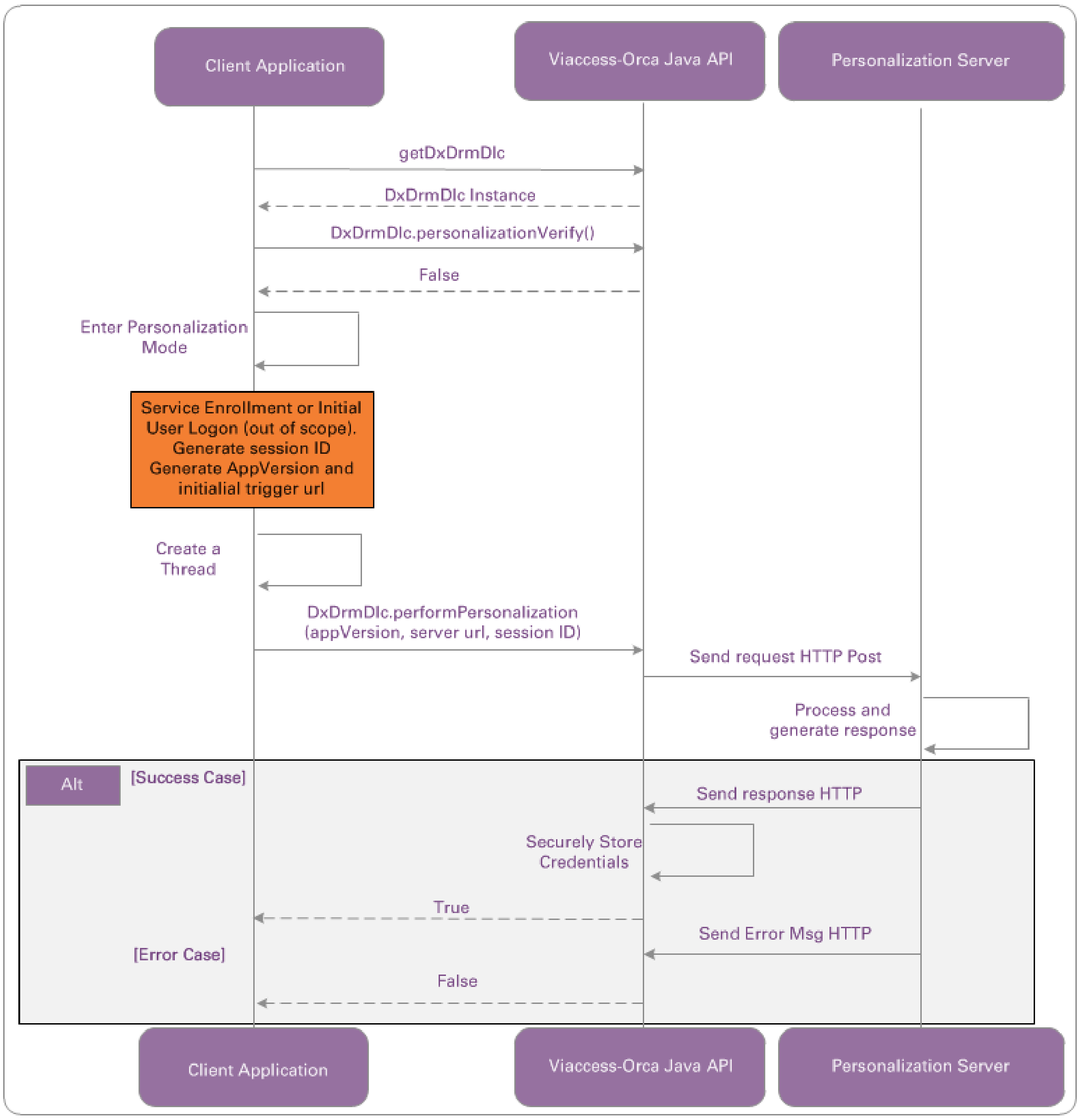

Client Application
Viaccess-Orca Java API
Personalization Server
getDxDrmDlc
DxDrmDlc Instance
DxDrmDlc.personalizationVerify()
False
Enter Personalization Mode
Service Enrollment or Initial User Logon (out of scope). Generate session ID Generate AppVersion and initialial trigger url
Create a Thread
DxDrmDlc.performPersonalization (appVersion, server url, session ID)
Send request HTTP Post
Process and generate response
Alt
[Success Case]
Send response HTTP
Securely Store Credentials
True
Send Error Msg HTTP
[Error Case]
False
Client Application
Viaccess-Orca Java API
Personalization Server

*figure 2.* Personalization Sequence Diagram

The personalization flow works as follows:

1. Following the download and installation of the client, the user activates the client for the first time. The device does not contain any previous personalization information.
2. The player application creates an instance of the `DxDrmDlc` object by calling `DxDrmDlc.getDxDrmDlc()` with the current activity context. To enable logging, a `DxLogConfig` object should be supplied.
3. The CSP-SDK Java API responds with a reference to the `DxDrmDlc` instance.
4. The player application calls `DxDrmDlc.personlizationVerify()` to query if it needs to perform personalization.
5. As the device does not contain personalization information, the CSP-SDK returns `False`.
6. Based on the returned value, the player application master state machine enters **Personalization Mode**, in which the user is presented with a limited set of screens and methods.
7. The player application notifies the user when entering the **Personalization Mode** (a “Please Wait” spinner animation or something similar).

8. The application should invoke a thread before calling `DxDrmDlc.performPersonalization()` to begin the personalization operation. Invoking a thread is highly recommended as the personalization process takes time. The application should stay responsive and avoid receiving an 'ANR' message from the Android OS.
9. The personalization request is sent to the personalization server.
10. The personalization server processes the message, retrieves the relevant device credentials and creates a response message. If an error occurs, an error message is sent. This is described in detail in *Integrating CSP Server-Side,* section *Personalization Response*.
11. The personalization server sends an HTTP response using the message as the response body.
12. The player application is in control as soon as the Java layer of the CSP-SDK finishes the personalization process.
13. The client application updates the user on completion of the personalization procedure, and moves on to the normal operation mode.

## DRM Management

### Rights acquisition

There are two ways of acquiring rights:

- Content-based rights acquisition: the application should call the `DxDrmDlc.acquireRights()` method.
- Initiator-based rights acquisition: the application should call the `DxDrmDlc.executeInitiator()` method.

Both methods are synchronous. They establish network connection and block the caller until the acquisition process is done. Therefore, to ensure application responsiveness these methods should not be called from the UI thread.

**note**

The license acquisition operation must complete successfully prior to content playback.

The following diagram illustrates a typical interaction between the application, the CSP-SDK and the PlayReady® license server when license acquisition is performed.

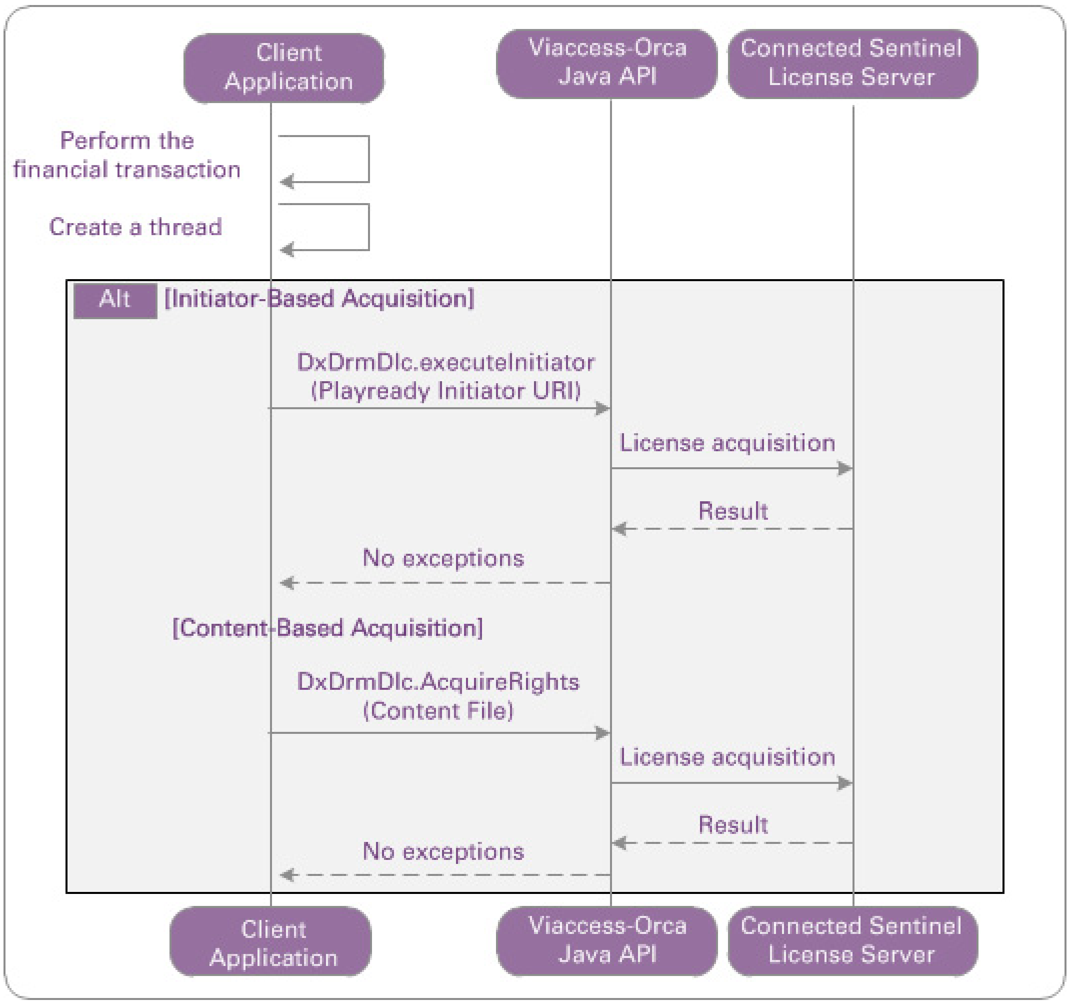

Client Application
Viaccess-Orca Java API
Connected Sentinel License Server
Perform the financial transaction
Create a thread
Alt
[Initiator-Based Acquisition]
DxDrmDlc.executeInitiator (Playready Initiator URI)
License acquisition
Result
No exceptions
[Content-Based Acquisition]
DxDrmDlc.AcquireRights (Content File)
License acquisition
Result
No exceptions
Client Application
Viaccess-Orca Java API
Connected Sentinel License Server

*figure 3.* License acquisition sequence diagram

The license acquisition works as follows:

1. User requests to acquire a content item with a specific license initiator (provided earlier from a content server).
2. The player application handles the financial aspects of the purchase.
3. The application should invoke a thread before calling `DxDrmDlc.acquireRights()` or `DxDrmDlc.executeInitiator()` to begin the acquisition operation. Invoking a thread is highly recommended as acquisition takes time, and the application should stay responsive and avoid receiving an 'ANR' message from the Android OS.
4. The client application instructs the Viaccess-Orca Java API to acquire the license by invoking `DxDrmDlc.executeInitiator()` method, passing the URL to the PlayReady initiator.
5. Alternatively, the client application instructs the Viaccess-Orca CSP-SDK framework to acquire the license by invoking the `DxDrmDlc.acquireRights()` method with the content filename.
6. The license acquisition completes without exception.
7. The player application closes the thread and jumps to the next operation in its flow: downloading the file or initiating the playback of the remote file.

## Acquiring Rights for Different Content Types

The CSP-SDK on Android platform supports the following content types:

- Envelope
- Smooth Streaming (PIFF)

Initiator-based rights acquisition is independent of the content type. If an initiator is not available, the application is required to acquire rights using Content-based rights acquisition.

Acquiring rights for Envelope:

1. The Envelope file must reside on the local file system.
2. The application should call `DxDrmDlc.acquireRights()` with the Envelope file path as the first parameter.

Acquiring rights for Smooth Streaming (PIFF):

1. The application should download the Manifest file.
2. The application should call `DxDrmDlc.acquireRights()` with the Manifest file path as the first parameter.

## Adding HTTP Cookies to the Acquiring Rights Request

To add HTTP cookies to the request from the license server do the following:

1. `DxDrmDlc.setCookies` (<cookies Array>).
2. Perform the HTTP request using the `DxDrmDlc.acquireRights()` or the `DxDrmDlc.executeInitiator()`.
3. Clear the cookies by calling `DxDrmDlc.setCookies`(null)

## Retrieving License Details

To retrieve license information for a specific content, the CSP-SDK provides a method called `DxDrmDlc.getRightsInfo()`. This method accepts a filename/URI and a context object and returns an (array of) `IDxRightsInfo` object.

The `IDxRightsInfo` object provides license information (e.g., the license state and the restrictions for the specified content).

A string representation of the `IDxRightsInfo` object can be provided. However, this ability is not part of the CSP-SDK. A reference code for this is part of the provided reference code.

**note**

`IDxRightsInfo` objects contain static information from the time `DxDrmDlc.getRightsInfo()` was called. These objects should not be cached or reused at later stages.

The following diagram illustrates the process:

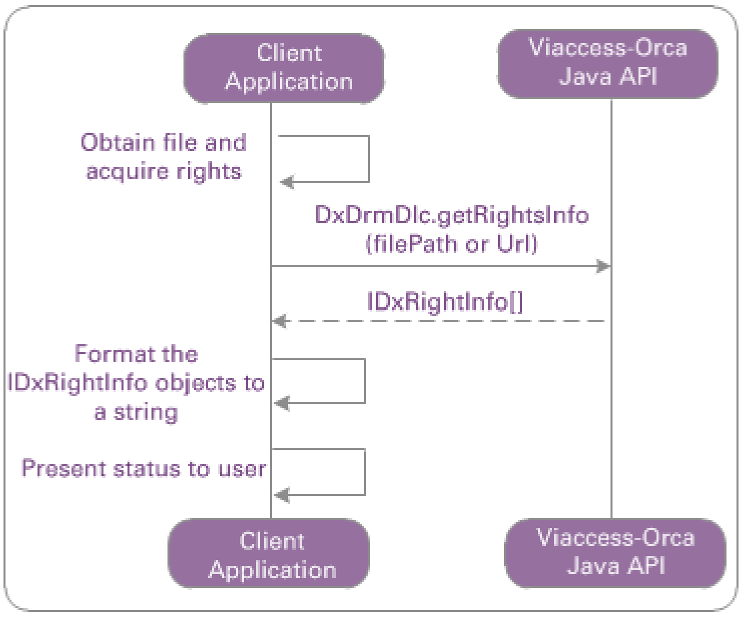


*figure 4.* Retrieving license details

An alternative method to retrieve rights information is to download the initiator (XML-based CMS file) used to acquire rights for the same content and apply the `DxDrmDlc.getRightsInfo()` method to it.

## Determine if Content is DRM-Protected

When presenting a user with a list of downloaded content, the application may require displaying an indication showing whether a content file is DRM protected (for example, showing a padlock icon).

For this purpose, the `DxDrmDlc.isDrmContent()` method should be used. It is passed a path or URI of the queried DRM content file and returns `True` if the file is DRM protected, or `False` otherwise.

The following diagram illustrates this process:

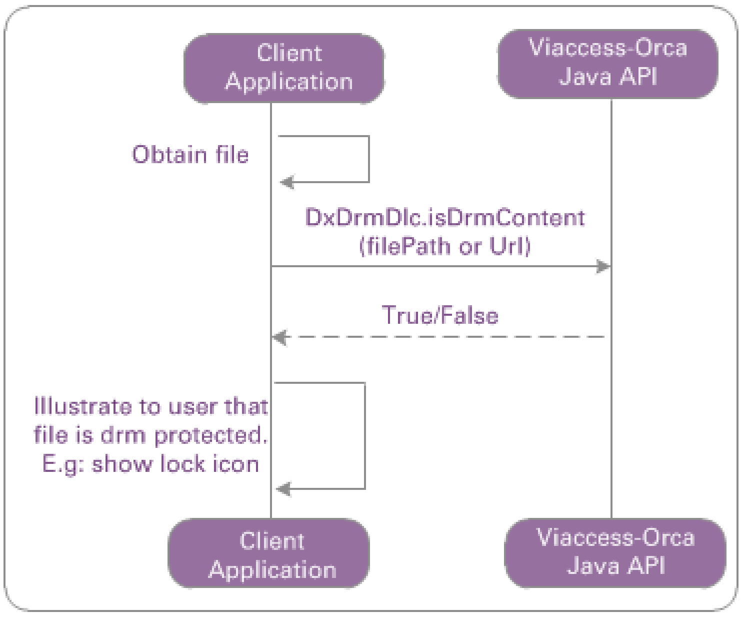

Client Application
Viaccess-Orca Java API
Obtain file
DxDrmDlc.isDrmContent (filePath or Url)
True/False
Illustrate to user that file is drm protected. E.g: show lock icon
Client Application
Viaccess-Orca Java API

*figure 5.* Determining if content is protected

An alternative method to identify whether a specific media file is content-DRM protected or not is to download the initiator (XML-based CMS file) used to acquire rights for the same content and apply the `DxDrmDlc.isDrmContent()` method on it.

## Determine if Content can be played

When presenting a user with a list of downloaded content, the application may require displaying an indication showing whether a valid license was obtained for content files (for example, showing either play or purchase icons).

For this purpose, the `DxDrmDlc.verifyRights()` method should be used. It is passed a path or URI of the queried DRM content file and returns `True` if there is a valid license installed for this file, or `False` otherwise.

The following diagram illustrates this process:

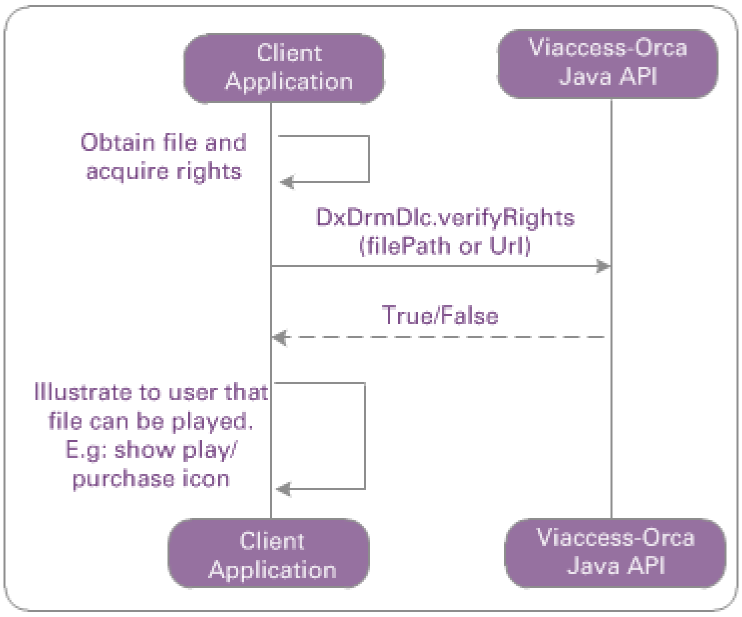

Client Application
Viaccess-Orca Java API
Obtain file and acquire rights
DxDrmDlc.verifyRights (filePath or Url)
True/False
Illustrate to user that file can be played. E.g: show play/ purchase icon
Client Application
Viaccess-Orca Java API

*figure 6.* Determining if content can be played

An alternative method to verify rights is to download the initiator (XML-based CMS file) used to acquire rights for the same content and apply the `DxDrmDlc.verifyRights()` method on it.

## Deleting Content License

If the application needs to remove a previously-purchased license it can call the `DxDrmDlc.deleteRights()` method with either a URI or a filename. The method will remove any license associated with the file/URI.

An alternative method to delete rights is to download the initiator (XML-based CMS file) used to acquire rights for the same content and apply the `DxDrmDlc.deleteRights()` method on it.

## Retrieving DRM Version Details

The CSP-SDK provides a method called `DxDrmDlc.getDrmVersion()` which returns a string that represents the DRM version defined in the build process.

# SOAP Error Handling

The CSP-SDK redirects SOAP errors to the Customer Application.

The SOAP errors are propagated via an exception called `DrmServerSoapErrorException`. This specific exception holds a structure maintaining 3 string parameters: `SoapMessage`, `RedirectUrl` and `CustomData`.

For additional information, refer to *Common Integration Guide*, section *SOAP Error Handling*.

# Content playback

Viaccess-Orca and VisualOn provide support for local and remote playback of video media via the VODXPlayer interface. This interface exposes an API to control the software player.

The CSP-SDK Client Application must supply Android 's SurfaceView to VODXPlayer to render the video.

The application must implement a UI for controlling playback and seeking actions, for example, a pause button or a seek bar.

The VODXPlayer requires to be tied with an activity that is playing the video as follows:

- The activity's `onResume()` method must call `VODXPlayer resume()`
- The activity's `onPause()` method must call `VODXPlayer suspend()`

The supported playback scenarios are described in VisualOn Player SDK Integration Guide for Android and the supported DRM file formats are described in the *Common Integration Guide*, section *Content Playback*.

## Precondition for Protected Content Playback

A valid license must be available on the device for successful playback of a protected content file.

The application should call the `DxDrmDlc.verifyRights()` method to ascertain license availability. This method returns `True` if there is a license available, or `False` otherwise.

When no valid license exists, the application may suggest that the user purchases a new content license, as described in the *Common Integration Guide*, section *Obtaining Rights Information*, or it may display an appropriate error message.

## Displaying video using VODXPlayer

To add VODXPlayer to your application, proceed as follows:

1. Create an instance of a class implementing the VODXPlayer interface, for instance `VODXPlayerImpl`
2. Add `SurfaceView` to your Android layout XML file `video.xml`

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout android:id="@+id/LinearLayout01"
     android:layout_height="fill_parent"
     xmlns:android=http://schemas.android.com/apk/res/android
     android:paddingLeft="2px" android:paddingRight="2px"
     android:paddingTop="2px" android:paddingBottom="2px"
     android:layout_width="fill_parent"
     android:orientation="vertical">
     <SurfaceView android:id="@+id/dxvoSurfaceView"
           android:layout_height="fill_parent"
           android:layout_width="fill_parent"
           android:layout_centerInParent="true" />
</LinearLayout>
```

3. Implement the `VOCommonPlayerListener` interface to get notifications from the player and register an instance of the listener with the player object
4. Play content using the `VODXPlayer` object.

# Progressive download playback

VisualOn Player SDK Integration Guide for Android describes how to play content in progressive download mode.

Protected content in the following DRM formats can be played in progressive download mode:

- PlayReady Envelope file (.eny)

- IIS Smooth Streaming Video (.ismv)

**note**

The application must not allow seeking forward beyond the actually downloaded portion of an.ismv file in progressive download mode.

## Playback accompanying features

VisualOn Player SDK Integration Guide for Android describes how to use playback accompanying features such as enabling subtitles or switching between multiple audio channels.

# Developing Client Application

This section explains how to integrate and test the CSP-SDK Client without the personalization server, and with the Microsoft® PlayReady test server.

Providing the client application with PlayReady assets is a prerequisite for all secure playback and DRM functionality. In the final production application, these assets should be obtained from the personalization server. However, this requires a working personalization server (which is described in detail in the *Connected Sentinel Player Personalization API*).

Viaccess-Orca provides two alternatives for developing and testing the client application:

- Local personalization
- Python® test server

Both alternatives install test certificates and keys which enable testing with the Microsoft PlayReady test server. For more information on the Microsoft PlayReady test server see http://playready.directtaps.net/.

## Local Personalization

This option performs a local personalization process and does not require personalization-server involvement.

The CSP-SDK solution package includes built-in required credentials for working with the Microsoft PlayReady test server. Prior to performing the *Personalization, Verification and Initiation* use case, call `DxDrmDlcDebug.setClientSideTestPersonalization(True)` (refer to *Set Client-Side Test Personalization Mode* on page 27) to enter the client-side test personalization mode. A call to `DxDrmDlc.performPersonalization()` will provision the local credentials without contacting the personalization server.

## Python Test Personalization server

Viaccess-Orca provides a Python implementation of a test personalization server which provides the client application with the required credentials.

Run the personalization server on the desired IP and port. To find out how to install and configure the test server refer to *Integrating CSP Server-Side,* section *Personalization Project Setup*. Follow the personalization initiation and verification use-case (described in *Personalization, Verification and Initiation*) as if working with a regular personalization server.

# Debug API

The debug API provides additional functionality required in a debugging environment.

To retrieve the debug API call `DxDrmDlc.getDebugInterface()`, which returns an `DxDrmDlcDebug` interface object. Use this object for the scenarios described in the following subsections.

## Deleting Personalization Credentials

Normally, personalization is executed once in an application's lifetime, or when the software is updated.

When debugging, it may be helpful to delete the personalization data. This can be done by calling `DxDrmDlcDebug.deletePersonalization()`. This method throws a `DrmGeneralFailureException` upon error.

## Configuring Logs

The logging levels can be configured by passing a `DxLogConfig` object to the `DxDrmDlc.getDxDrmDlc()` method. Logs are disabled by default. This `DxLogConfig` object allows to configure the log level, the path for the logs, and which of the modules will print logs. For exact details on input parameters refer to *Connected Sentinel Player Android API Reference*.

## Set Client-Side Test Personalization Mode

Developing and testing with the Connected Sentinel Player client-side can be done without requiring a personalization server. The CSP-SDK solution package includes built-in credentials for working with the Microsoft PlayReady test server. This is achieved by calling `DxDrmDlcDebug.setClientSideTestPersonalization(True)` prior to `DxDrmDlc.performPersonalization()`. For more information refer to *Developing Client Application* on page 26.

# Appendix

This appendix contains a glossary and the list of reference documentation.

## Glossary

The following table shows typical terms and acronyms found in this document

| Term | Definition |
|---|---|
| ANR | Application Not Responding |
| API | Application Programming Interface |
| AV | Audio Video |
| CA | Certificate Authority |
| CID | Content ID |
| Client/Customer Application | The secure media player application comprised of the customer code and the Connected Sentinel Player |
| CSP | Connected Sentinel Player |
| CSP-SDK | Viaccess-Orca Connected Sentinel Player SDK |
| DRM | Digital Rights Management |
| GUI | Graphical user Interface |
| IDE | Integrated Development Environment |
| NDK | Native Development Kit |
| SDP | Session Description Protocol |
| SO | Shared Object |
| TLV | Type Length Value Encoding |

## Reference Documentation

- Common Integration Guide
  Reference number: 21816
- Integrating CSP Server-Side
  Reference number: 21826
- Connected Sentinel Player SDK Android API Reference
  Reference number: xxxxx
- Connected Sentinel Player Personalization API
  Reference number: xxxxx
- Connected Sentinel Player Android API
  Reference number: xxxxx
- VisualOn Player SDK Integration Guide for Android - version: 1.2
  Provided by Viaccess-Orca as part of the CSP 1.x documentation package