SECUREPLAYER SDK

ANDROID INTEGRATION GUIDE

VERSION 2.0.1

LAST UPDATED: FEB 2013

External Revision History

Rev	Date	Description
0.94	06-Nov-11	First Official Release.
1.0	20-Mar-12	• Modified section: *Content Playback*. Splitting the section into two subsections – *Precondition for Protected Content Playback* and *Displaying Video using DxVideoView*. • Added section: *SOAP Error Handling*. • Added section: *Progressive Download Playback*. • Modified section: *Determine if Content is DRM Protected*. Alternative method using initiators. • Modified section: *Determine if Content can be Played*. Alternative method using initiators. • Modified section: *Deleting Content License*. Alternative method using initiators.
1.0.2	24-May-12	• Modified section: *Rights Acquisition* – swap referenced methods
1.1	16-Aug-12	• Modified section: *Retrieving License Details*. Alternative method using initiators.
2.0	15-Nov-12	• Modified section: *Related Documents*. Adding reference to the documentation by VisualOn • Modified section: *Client Side Architecture*. • Modified section: *Content Playback*. • Modified section: *Debug API*. Renaming IDxDrmDlcDebug to DxDrmDlcDebug to comply with the new naming convention

Contents

List of Figures

List of Tables

1 Introduction

This document provides guidelines for integrating the SecurePlayer SDK within an application. The interface described in this document is specific to Android-based platforms.

1.1 Intended Audience

This document is intended for developers writing an Android player application based on the SecurePlayer SDK.

1.2 Related Documents

Table 1: Reference Documentation

Reference	Document
[SP_SDK_COMM_IG]	SecurePlayer SDK Common Integration Guide
[SP_PERS_SDK_IG]	SecurePlayer Server-Side Personalization SDK Integration Guide
[SP_PERS_PROJ_STP]	SecurePlayer Server-Side Personalization SDK Project Setup
[SP_PERS_SDK_API]	SecurePlayer Personalization SDK API Guide
[SP_ANDR_PROJ_STP]	SecurePlayer SDK Android Project Setup
[SP_ANDR_SDK_API]	SecurePlayer SDK Android API Reference
[VO_ANDR_IG]	VisualOn Player SDK Integration Guide for Android (Version: 1.2)

1.3 Terms and Abbreviations

Table 2: Glossary

Term	Description
DRM	Digital Rights Management
CA	Certificate Authority
CID	Content ID
Client Application/ Customer Application	The secure media player application, comprised of the customer code and the SecurePlayer-SDK.
API	Application Programming Interface
AV	Audio Video
GUI	Graphical User Interface
NDK	Native Development Kit
SDP	Session Description Protocol
TLV	Type-Length-Value Encoding

Term	Description
SP-SDK	Discretix SecurePlayer SDK. Refers to components provided by Discretix

2 SecurePlayer Client SDK

2.1 Client Side Architecture

The following diagram illustrates the device application's internal architecture:

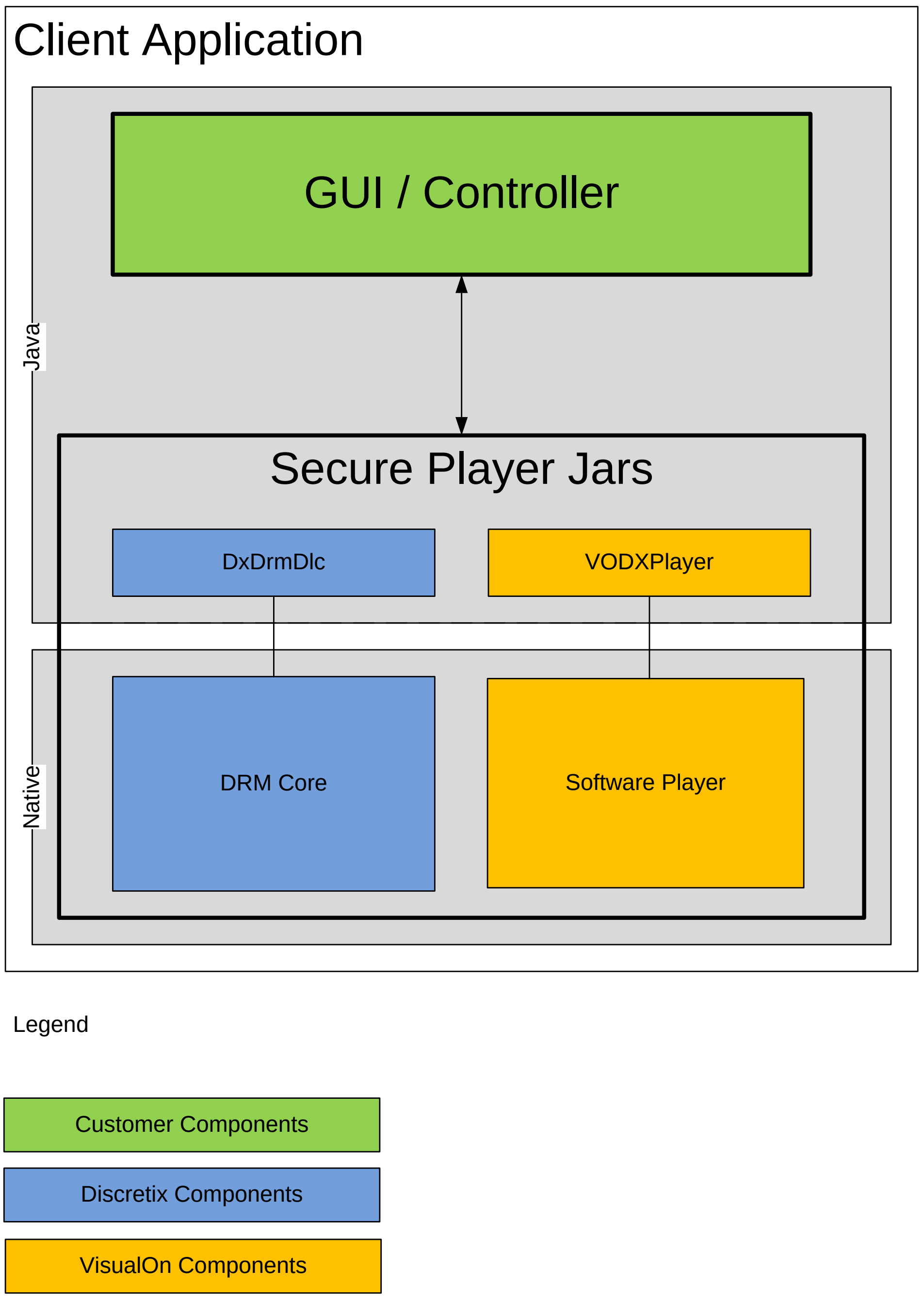

Figure 1: Typical Client Application Architecture

Drawing-Component Descriptions:

- **GUI/Controller** - GUI front-end and logic for the client application, locally stored content.
- **Secure Player Jars (Java API)** - Java packages that provide the customer application for Android devices with interfaces for downloading and managing content licenses and content playback operations.
- **DRM Core** - A native library, which implements the DRM core functionality.
- **Software Player** – Native libraries, which implement the video player that displays the content.

2.2 SecurePlayer SDK Integration

This section provides a guide for integrating the Discretix SP-SDK with the customer application (client-side) by use-cases related to PlayReady® DRM content protection.

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 SP-SDK components and interface.

Table 3: Integration Use Cases

Use Case	Discussed in Section
Personalization	
Personalization Verification and Initiation	Personalization Verification and Initiation (p. 10)
DRM Management	
Acquiring a License	Rights Acquisition (p. 12)
Retrieving License Details	Retrieving License Details (p. 14)
Determine if Content is Protected	Determine if Content is DRM Protected (p. 15)
Determine if Content can be Played	Determine if Content can be Played (p. 16)
Deleting Content License	Deleting Content License (p. 17)
Retrieving DRM Version Details	Retrieving DRM Version Details (p. 17)
SOAP Error Handling	
SOAP Error Handling	
Content Playback	
Displaying video using `VODXPlayer`	Displaying Video using VODXPlayer (p. 18)
Debug API	
Deleting Personalization Credentials	Deleting Personalization Credentials (p. 20)
Configuring Logs	Configuring Logs (p. 20)
Set Client Side Test Personalization Mode	Set Client-Side Test Personalization Mode (p. 21)

2.2.1 Personalization

2.2.1.1 Personalization Verification and Initiation

Upon invocation, the application must verify the personalization status by calling the `DxDrmDlc.personlizationVerify()` method; this is a short operation, and can be performed from any context.

Successful personalization (`DxDrmDlc.personlizationVerify()` returns `True`) is a pre-condition for using any of the other SP-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 application responsiveness.

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

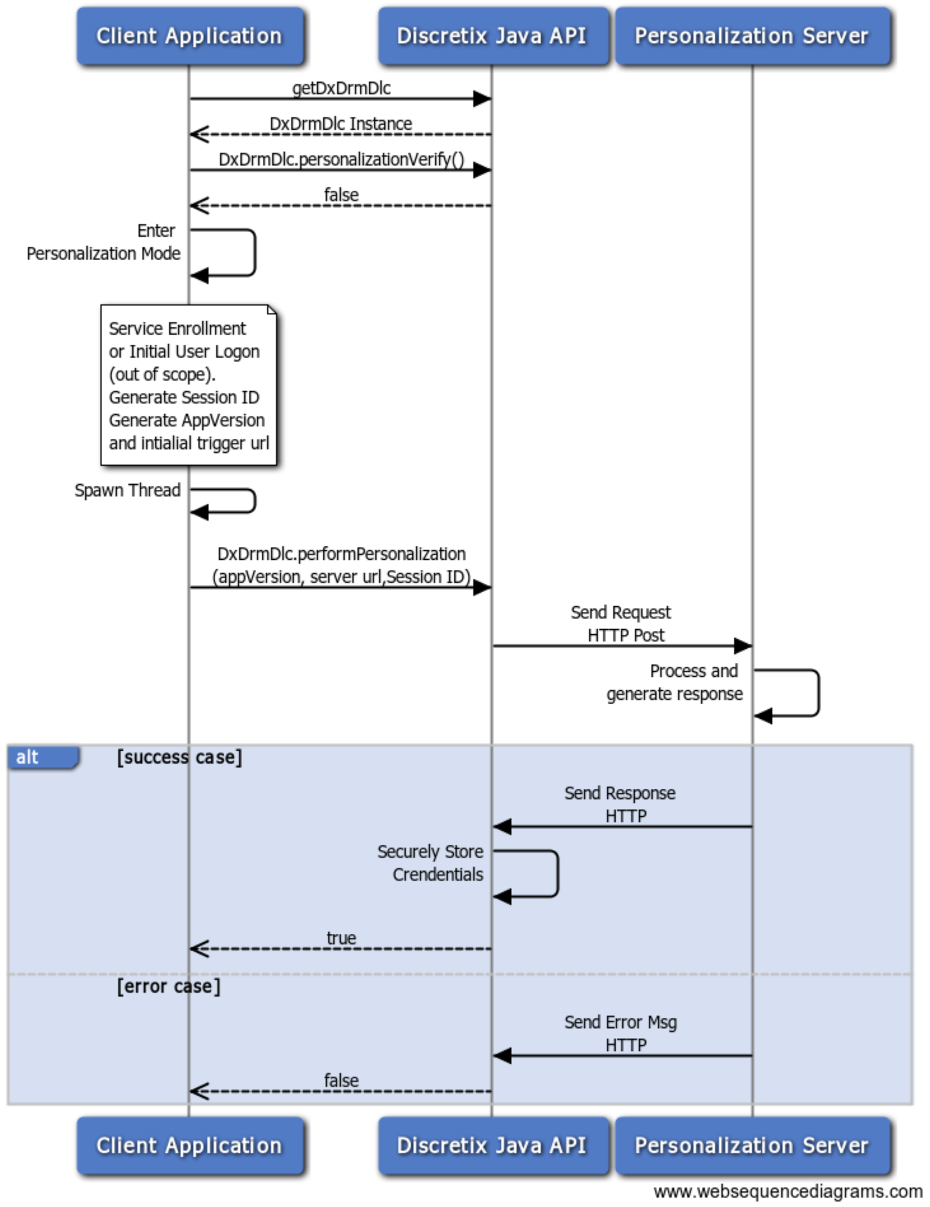
Client Application
Discretix Java API
Personalization Server
getDxDrmDlc
DxDrmDlc Instance
DxDrmDlc.personalizationVerify()
false
Enter Personalization Mode
Service Enrollment or Initial User Logon (out of scope). Generate Session ID Generate AppVersion and intialial trigger url
Spawn Thread
DxDrmDlc.performPersonalization (appVersion, server url,Session ID)
Send Request HTTP Post
Process and generate response
alt
[success case]
Send Response HTTP
Securely Store Crendentials
true
[error case]
Send Error Msg HTTP
false
www.websequencediagrams.com

Figure 2: Personalization Sequence Diagram

Personalization Flow:

1. Following 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 SP-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 SP-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 about entry to **Personalization Mode** (a "Please Wait" spinner animation or something alike).
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 and the application should stay responsive and avoid receiving an 'ANR' message from the Android OS.
9. The personalization request is sent to 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 created instead. This is described in detail in referenced document **[SP_PERS_SDK_IG]** - *Personalization Response*.
11. The personalization server sends an HTTP response using the message as the response body.
12. The Java layer of the SP-SDK finishes the personalization process and returns control to the player application.
13. The client application updates the user on completion of the personalization procedure, and moves on to normal operation mode.

2.2.2 DRM Management

NOTE! That external subtitle files cannot be protected at this point.

2.2.2.1 Rights Acquisition

There are two ways to acquire 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.

For additional information see referenced document **[SP_SDK_COMM_IG]** - *Rights Acquisition (PlayReady).*

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

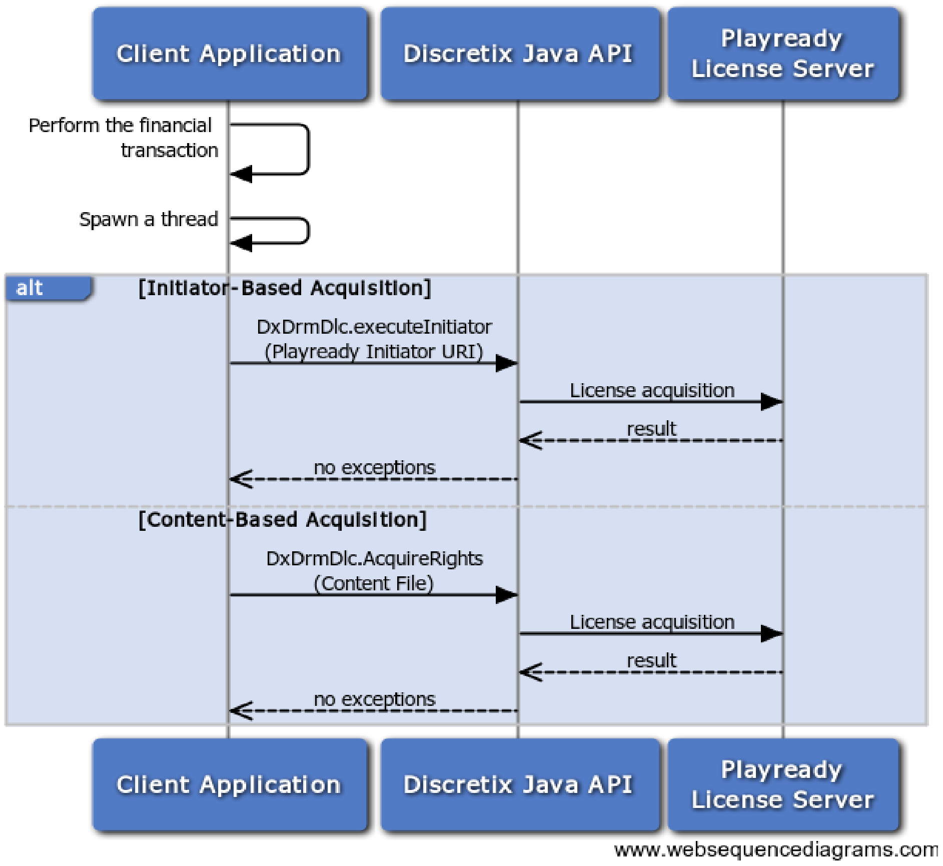
Client Application
Discretix Java API
Playready License Server
Perform the financial transaction
Spawn a thread
alt
[Initiator-Based Acquisition]
DxDrmDlc.executeInitiator (Playready Initiator URI)
License acquisition
result
no exceptions
[Content-Based Acquisition]
DxDrmDlc.AcquireRights (Content File)
License acquisition
result
no exceptions
www.websequencediagrams.com

Figure 3: License Acquisition Sequence Diagram

License Acquisition Flow:

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 Discretix Java API to acquire the license by invoking `DxDrmDlc.executeInitiator()` method passing it the URL to the PlayReady® initiator.

   Alternatively, the client application instructs the Discretix SP-SDK framework to acquire the license by invoking the `DxDrmDlc.acquireRights()` method with the content filename.
5. The license acquisition completes without an exception.
6. 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.

2.2.2.1.1 Acquiring Rights for Different Content Types

The SP-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.

2.2.2.2 Retrieving License Details

To retrieve license information for a specific content, the SP-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 SP-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 this process:

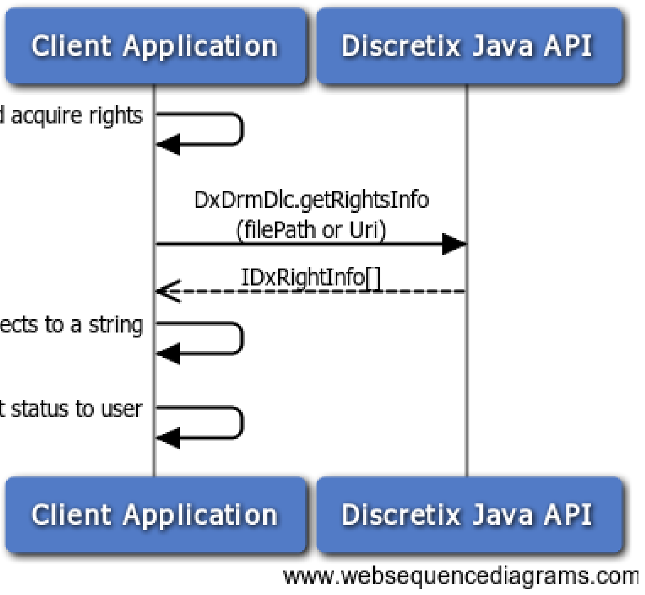

Figure 4: Retrieving License Details

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.

2.2.2.3 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:

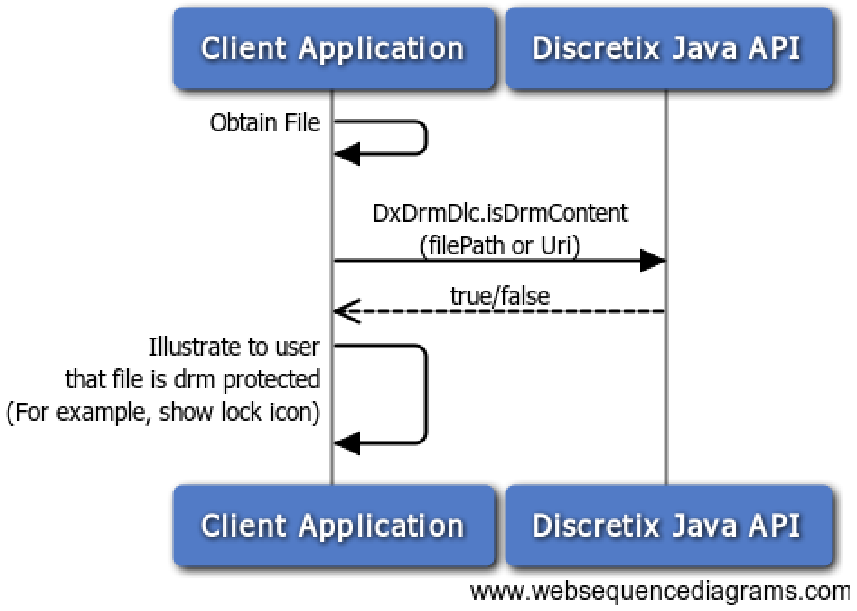
Client Application
Discretix Java API
Obtain File
DxDrmDlc.isDrmContent (filePath or Uri)
true/false
Illustrate to user that file is drm protected (For example, show lock icon)
Client Application
Discretix Java API
www.websequencediagrams.com

Figure 5: Determining if Content is Protected

An alternative method to identify whether a specific media file is content-DRM protected 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.

2.2.2.4 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:

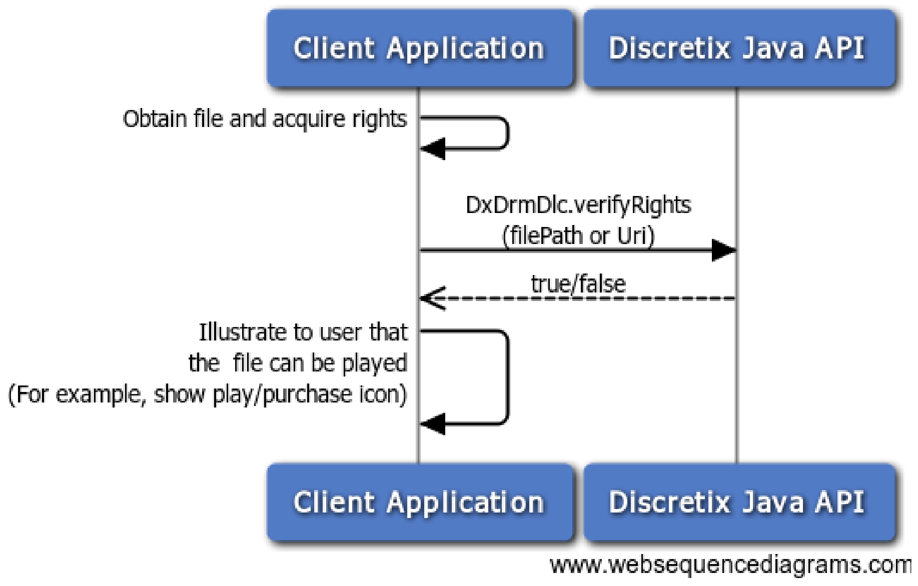
Client Application
Discretix Java API
Obtain file and acquire rights
DxDrmDlc.verifyRights (filePath or Uri)
true/false
Illustrate to user that the file can be played (For example, show play/purchase icon)
Client Application
Discretix Java API
www.websequencediagrams.com

Figure 6: Determine 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.

2.2.2.5 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.

2.2.2.6 Retrieving DRM Version Details

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

2.2.3 SOAP Error Handling

The SP-SDK propagates 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 see referenced document **[SP_SDK_COMM_IG]** – *SOAP Error Handling.*

2.2.4 Content Playback

Discretix 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 SP-SDK Client Application must supply Android's `SurfaceView` to `VODXPlayer` for rendering the video.

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

`VODXPlayer` requires to be tied with an activity that is playing the video in the following manners:

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

For the full playback scenarios see [VO_ANDR_IG].

The supported DRM file formats are described in **[SP_SDK_COMM_IG]** - *Content Playback*.

A valid license must be available on the device for successful playback of a protected content file. To verify that the license is available, the application can call the `DxDrmDlc.verifyRights()` method. This method returns `True` if license is available, or `False` otherwise.

When no valid license exists, the application may suggest that the user purchases a new content license, as described in referenced document **[SP_SDK_COMM_IG]** - *Obtaining Rights Information*, or display an appropriate error message.

2.2.4.1 Precondition for Protected Content Playback

A valid license must be available on the device for successful playback of a protected content file. To verify that the license is available, the application can call the `DxDrmDlc.verifyRights()` method. This method returns `True` if there is 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 referenced document **[SP_SDK_COMM_IG]** - *Obtaining Rights Information*, or display an appropriate error message.

2.2.4.2 Displaying Video using VODXPlayer

To add `VODXPlayer` to your application perform the following steps:

1. Create an instance of a class implementing the `VODXPlayer` interface, e.g. `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.

2.2.5 Progressive Download Playback

The following DRM formats are supported in progressive download:

- .eny – Envelope PlayReady file.
- .ismv* - IIS Smooth Streaming Video

*On ISMV file MUST NOT perform a "seek" operation until the file is fully downloaded.

For the full Progressive Download playback operation see [VO_ANDR_IG].

2.2.6 Extra playback options

To enable extra playback options such as subtitle and multi audio channel.

See [VO_ANDR_IG].

2.3 Developing the Client Application

This section explains how to integrate and test the SP-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 referenced document **[SP_PERS_SDK_API]**).

Discretix 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 additional information on the Microsoft® PlayReady® test server see http://playready.directtaps.net/.

2.3.1 Local Personalization

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

The SP-SDK solution package includes built-in required credentials for working with the Microsoft® PlayReady® test server. Prior to performing the Personalization Verification and Initiation (p. 10) use case, call `DxDrmDlcDebug.setClientSideTestPersonalization(True)` (see Set Client-Side Test Personalization Mode (p. 21)) to enter the client-side test personalization mode. A call to `DxDrmDlc.performPersonalization()` will provision the local credentials without contacting the personalization server.

2.3.2 Python Test Personalization Server

Discretix 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 install and configure the test server see referenced document **[SP_PERS_PROJ_STP]**. Follow the personalization initiation and verification use-case (described in Personalization Verification and Initiation (p. 10)) as if working with a regular personalization server.

2.4 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.

2.4.1 Deleting Personalization Credentials

Normally, personalization is executed once in a 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.

2.4.2 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 configuring the log level, the path for the logs, and which of the modules will print logs.

For exact details on input parameters see referenced document **[SP_ANDR_SDK_API]**.

2.4.3 Set Client-Side Test Personalization Mode

Developing and testing with the SecurePlayer client-side SDK can be done without requiring a personalization server. The SP-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 additional information see Developing the Client Application (p. 19).