Integrating CSP with iOS®

Connected Sentinel Player 1.x

viaccess·orca

Document reference number: 21818

Document version number: 1.0 Release

Contents

Chapter 1: Introduction

This document provides guidelines on how to set up and configure an iOS® project using the Viaccess-Orca Connected Sentinel Player SDK, and on how to integrate the Connected Sentinel Player SDK within an application. The interface described in this document is specific to iOS®-based platforms.

Target Audience

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

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 19 at the end of the manual for a complete definition.

Chapter 2: iOS Project Setup

To set up an iOS project for the Connected Sentinel Player, you need the following:

Pre-requisites

The following are prerequisites to setting up an iOS project:

- OSX Version - The minimal OS version required is OS X Mountain Lion (10.8.x).
- xCode Version - The xCode required is at least version 5.0.

Package Content

The operator receives a ZIP file containing documents (including Doxygen), framework of the Connected Sentinel Player SDK and a Connected Sentinel Player Demo application (DxApiDemos) including source code.

Item name	Description
SecurePlayer.framework	The framework contains the Connected Sentinel Player SDK.
DxApiDemos	The folder contains an xcode project including source of an application with basic usage of the Connected Sentinel Player framework.
Documentation	The documents include the Doxygen and .pdf files

Setup Procedure

Extracting the package

To extract the package:

1. Extract the `SecurePlayer.framework` folder from the provided ZIP.
2. Place the `SecurePlayer.framework` folder somewhere in your file system.

Caution: The `SecurePlayer.framework`, like any other framework, contains soft links that cannot be addressed properly in Windows OS, and might be changed by it. Therefore, you must unzip the `SecurePlayer.framework` on an OSX machine only.

Modifying iOS Projects

The developer should insert the `SecurePlayer.framework` in the xCode project that will be using the Connected Sentinel Player functionality.

To modify the xCode project:

1. iOS simulator is not supported - Select iOS Device destination only.
2. Open the xCode project.
3. Add the following frameworks into your project:
   `Libz.dylib` (part of the iOS libraries).
   `libresolv.dylib` (part of the iOS libraries).
   `MediaPlayer.framework` (part of the iOS libraries).
   `Security.framework` (part of the iOS libraries).

`CoreText.framework` (part of the iOS libraries).

`AVFoundation.framework` (part of the iOS libraries).

`AudioToolbox.framework` (part of the iOS libraries).

`CFNetwork.framework` (part of the iOS libraries).

`QuartzCore.framework` (part of the iOS libraries).

`UIKit.framework` (part of the iOS libraries).

`CoreGraphics.framework` (part of the iOS libraries).

`OpenGLES.framework` (part of the iOS libraries).

`CoreMedia.framework` (part of the iOS libraries).

`SecurePlayer.framework` - for location refer to *Extracting the package* on page 7.

4. Valid architectures: armv7, armv7s
5. Other linker flags: -lstdc++
6. C++ Standard Library: libc++ (LLVM C++ standard library with C++11 support).
7. For every file that should use the Connected Sentinel Player SDK API, add the import of the `SecurePlayer.h` file:

```
#import <SecurePlayer/SecurePlayer.h>
```

Local personalization

To enable local personalization the project should include within the Resource folder the PlayReady® test credentials wrapped in a single package file, specifically: provisioning.dat.

note

See the DxApiDemos project as a reference.

VisualOn Player resources

To enable the VisualOn player to play correctly a video player license and a CAP file should be included in the Resource folder. The files required are:

- iOS-cap.xml
- voVidDec.dat

note

See DxApiDemos project as a reference.

Chapter 3: Client SDK

Client Side Architecture

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

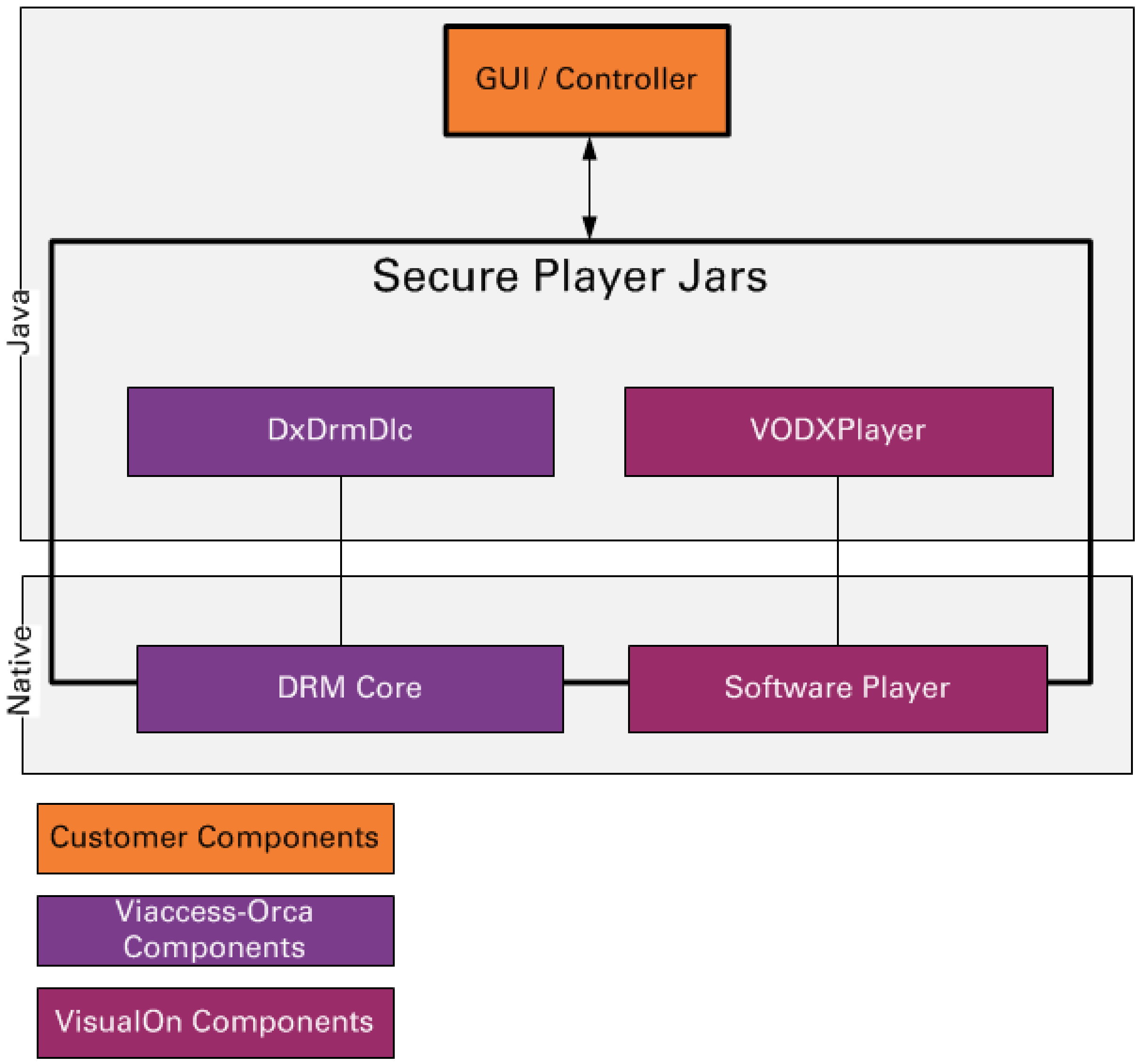

figure 1. Typical client application architecture

The diagram components are explained below:

- **GUI/Controller**: GUI front-end and logic for the client application, locally stored content.
- **SecurePlayer Framework**: A framework easily incorporated into xCode projects, wrapping the entire CSP-SDK APIs.
- **DxDrmManager**: An Objective-C interface for downloading and managing content licenses for integration with 'UI controller' (part of customer application for iOS devices).
- **VODXPlayer**: Player capable of interfacing with the DRM client for content decryption and playback operations.
- **DRM Core**: implements the DRM core functionality.

- **VisualOn® Player**: The actual video player displaying the content

Connected Sentinel Player SDK Integration

This section provides a guide to integrate the Viaccess-Orca CSP-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 CSP-SDK components and interface.

Types of Use Cases	Use Case
Personalization	*Personalization, Verification and Initiation on page 10*
DRM Management	*Rights Acquisition* on page 12
	Retrieving License Details on page 14
	Determining if Content is DRM-Protected on page 15
	Determining if Content can be Played on page 16
	Deleting Content License on page 16
	Retrieving DRM Version Details on page 16
Content Playback	Displaying video using `VODXPlayer` with *Content Playback* on page 17
Debug API	*Deleting Personalization Credentials* on page 18
	Configuring Logs on page 18
	Set Client-Side Test Personalization Mode on page 18

Personalization

Personalization, Verification and Initiation

Upon invocation, the application must verify the personalization status by calling the `[personlizationVerify]` function; this is a short operation, and can be performed from any context.

Successful personalization (`[personlizationVerify]` returns `YES`) is a pre-condition for performing any other DRM operation with CSP-SDK. If `NO` is returned, personalization was not yet performed, and the application should call the `[performPersonalizationWithSessionID:withServerURL:withAppVersion:]` function to accomplish it. The `[performPersonalizationWithSessionID:withServerURL:withAppVersion:]` function 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 CSP-SDK, and the Personalization server in a fresh activation of the client (e.g., the device was not personalized yet).

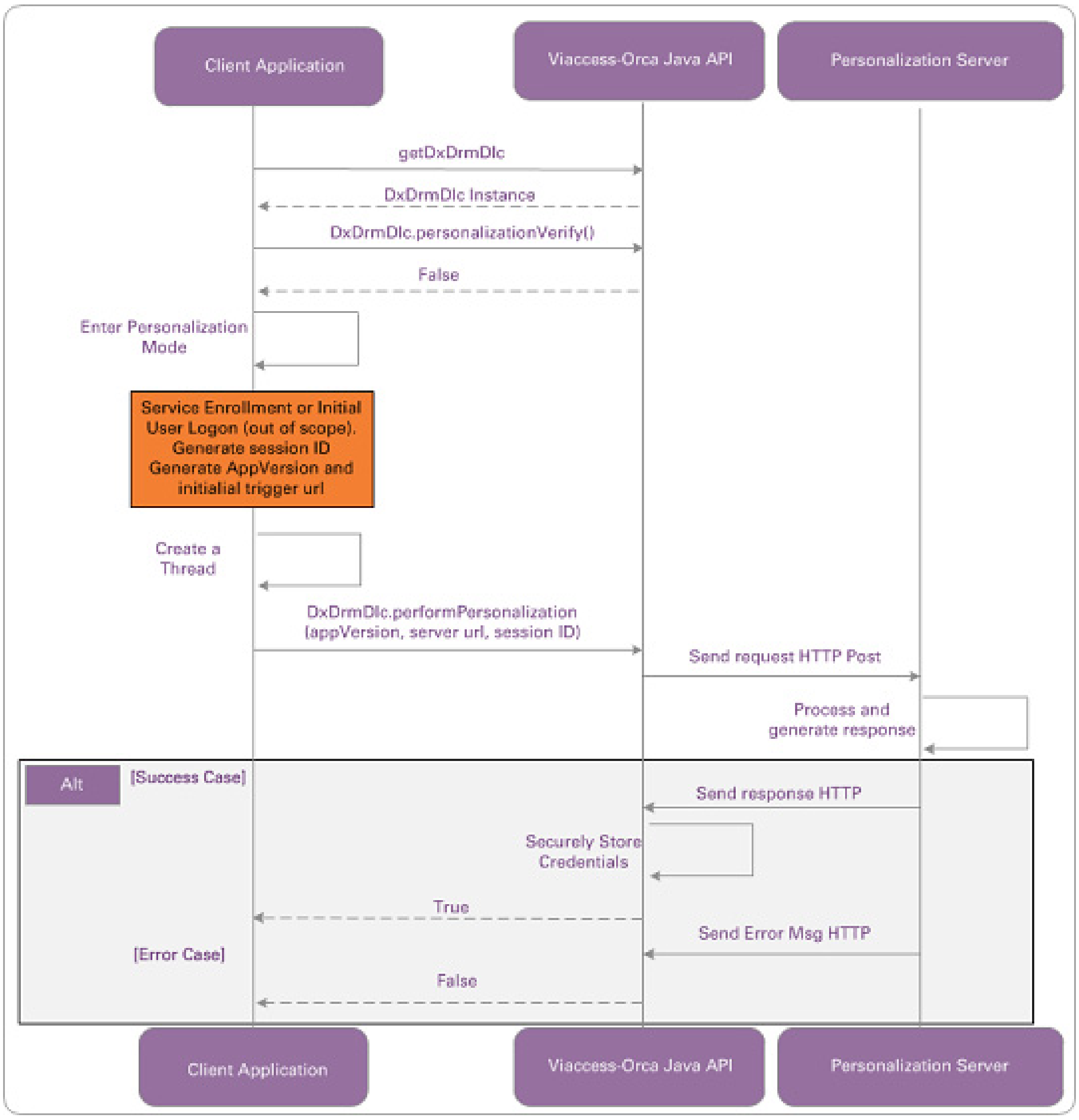
Client Application
Viaccess-Orca Java API
Personalization Server
getDxDrmDlc
DxDrmDlc Instance
DxDrmDlc.personalizationVerify()
False
Enter Personalization Mode
Service Enrollment or Initial User Logon (out of scope). Generate session ID Generate AppVersion and initialial trigger url
Create a Thread
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
[Error Case]
Send Error Msg HTTP
False
Client Application
Viaccess-Orca Java API
Personalization Server

figure 2. Personalization sequence diagram

The personalization workflow is as follows:

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 `DxDrmManager` object by calling `[DxDrmManager sharedManager]`. To enable logging, a `[DEBUG_setLoggingLevel]` object should be supplied.
3. The player application calls `[personlizationVerify]` to query if it needs to perform personalization.
4. As the device does not contain personalization information, the CSP-SDK returns `NO`.
5. Based on the returned value, the master state machine of the player application enters **Personalization Mode**, in which the user is presented with a limited set of screens and functions.
6. The player application notifies the user about entry to **Personalization Mode** (a "Please Wait" spinner animation or something alike).

7. The application should invoke a thread before calling `[performPersonalizationWithSessionID:withServerURL:withAppVersion:]` to begin the personalization operation.
8. The personalization request is sent to the Personalization server.
9. 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 details in *Integrating CSP Server-Side*, section *Personalization Response*.
10. The personalization server sends an HTTP response using the message as the response body.
11. The CSP-SDK finishes the personalization process and returns control to the player application.
12. The client application updates the user on completion of the personalization procedure, and moves on to normal operation mode.

DRM Management

note

When using Harmonic HLS format only the alternative method (using the initiator) will work. In sections *Retrieving License Details* on page 14, *Determining if Content is DRM-Protected* on page 15, *Determining if Content can be Played* on page 16 and *Deleting Content License* on page 16. A solution to this limitation is described in *Integrating CSP with Harmonic PlayReady Encryption*, chapter 4.

Rights Acquisition

There are two ways to acquire rights:

- Content-based rights acquisition: The application should call the `[acquireRightsForFile:withCustomData:withRightsUrl:error:]` function (This way is not suitable for HLS in Harmonic format).
- Initiator-based rights acquisition: The application should call the `[executeInitiatorWithFile:error:]`, `[executeInitiatorWithData:error:]` or `[executeInitiatorWithURL:error:]` functions.

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

note

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

For additional information, refer to the *Common Integration Guide*, section *Rights Acquisition (PlayReady)*.

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

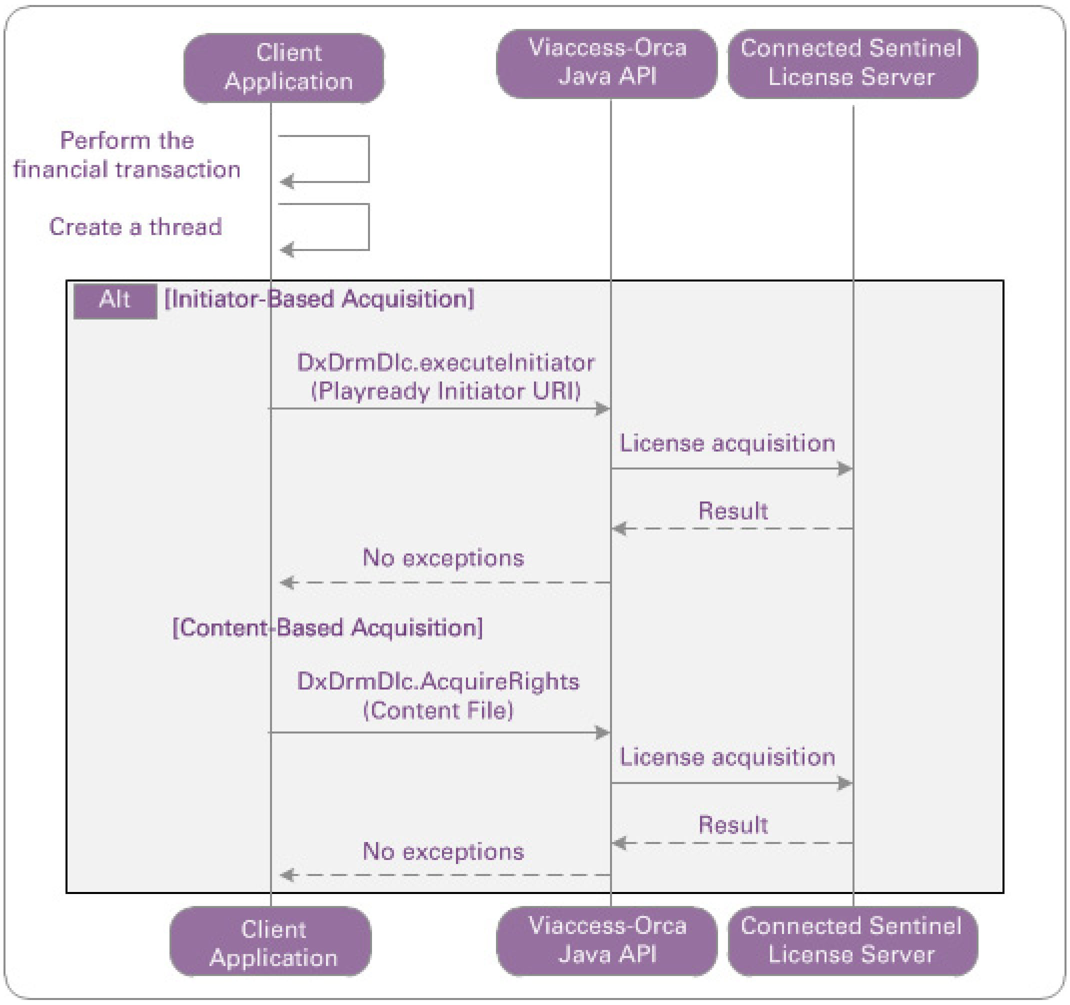
Client Application
Viaccess-Orca Java API
Connected Sentinel License Server
Perform the financial transaction
Create a thread
Alt
[Initiator-Based Acquisition]
DxDrmDlc.executeInitiator (Playready Initiator URI)
License acquisition
Result
No exceptions
[Content-Based Acquisition]
DxDrmDlc.AcquireRights (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 workflow is 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 the acquisition operations. Invoking a thread is highly recommended as acquisition takes time, and the application should stay responsive.
4. The client application instructs the Viaccess-Orca CSP-SDK Framework to acquire the license by invoking `[executeInitiatorWithFile:error:]`, `[executeInitiatorWithData:error:]` or `[executeInitiatorWithURL:error:]` functions with the PlayReady initiator.

   Alternatively, the client application instructs the Viaccess-Orca CSP-SDK framework to acquire the license by invoking the `[acquireRightsForFile:withCustomData:withRightsUrl:error:]` function with the content filename.
5. The license acquisition completes successfully.
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.

Acquiring Rights for Different Content Types

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

- Envelope
- HLS
- Microsoft Smooth Streaming

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.

- To acquire rights for Envelope:

1. The Envelope file must reside on the local file system.
2. The application should call `[acquireRightsForFile:withCustomData:withRightsUrl:error:]` with the Envelope file path as the first parameter.

- To acquire rights for HLS:

1. The application should download the play list file (`.m3u8`). It is important that the play list contains the `#EXT-X-DXPLAYREADY` attribute. This is usually part of the leaf play list.
2. The application should call `[acquireRightsForFile:withCustomData:withRightsUrl:error:]` with the play list as the first parameter.

- To acquire rights for Microsoft Smooth Streaming:

1. The application should download the manifest file (`.ism/Manifest`). It is important that the manifest contains the `Protection` element.
2. The application should call `[acquireRightsForFile:withCustomData:withRightsUrl:error:]` with the manifest as the first parameter.

Adding HTTP Cookies to the Acquiring Rights Request

To add HTTP cookies to the request from the license server, proceed as follows:

1. Call `[setCookies:<Cookies Array>]`.
2. Perform the HTTP request using the `[acquireRights…]` or the `[executeInitiator…]`.
3. Clear the cookies by calling `[setCookies:nil]`.

Retrieving License Details

To retrieve license information for a specific content, the CSP-SDK provides a function called `[getRightObjectsForFile:result:]`. This function accepts a filename/URI and a context object and returns an (array of) `IDxRightsInfo` object(s).

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 `[getRightObjectsForFile:result:]` was called. These objects should not be cached or reused at later stages.

The following diagram illustrates this process:

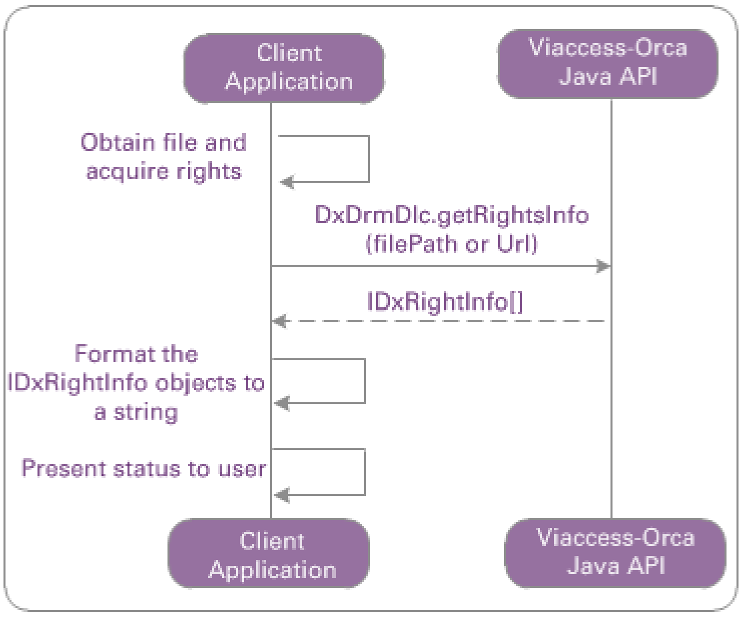

figure 4. Retrieving license details

An alternative method to retrieve license information is to download the Initiator (XML-based CMS file) used to acquire rights for the same content and apply the `[getRightObjectsForFile:result:]` function on it.

Determining 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 `[isDrmContent:result:]` function should be used. It is passed a path or URI of the queried DRM content file and returns `YES` if the file is DRM protected, or `NO` otherwise.

The following diagram illustrates this process:

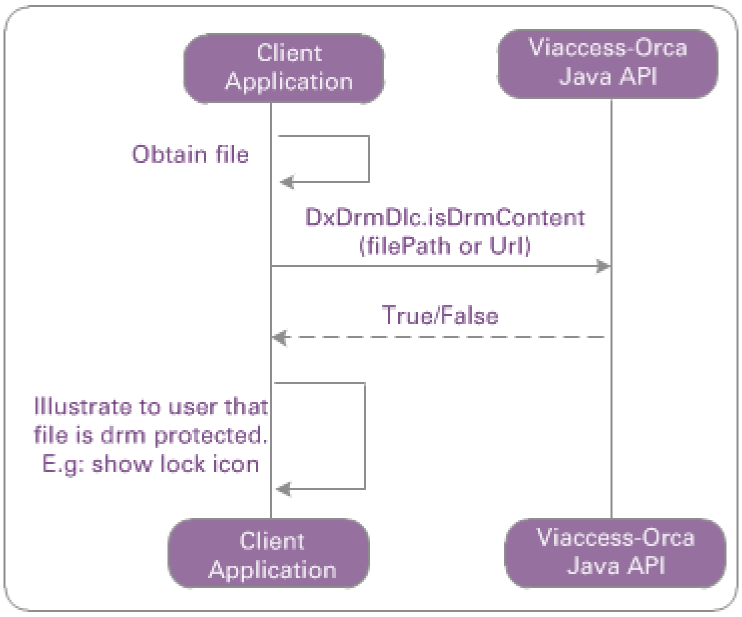
Client Application
Viaccess-Orca Java API
Obtain file
DxDrmDlc.isDrmContent (filePath or Url)
True/False
Illustrate to user that file is drm protected. E.g: show lock icon
Client Application
Viaccess-Orca Java API

figure 5. Determining if content is protected

An alternative method to identify whether a specific media file is content-DRM protected is to download the Initiator (XML-based CMS file) used to acquire rights for the same content and apply the `[isDrmContent:result:]` function on it.

Determining 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 `[verifyRightsForFile:]` function should be used. It is passed a path or URI of the queried DRM content file and returns `YES` if there is a valid license installed for this file, or `NO` otherwise.

The following diagram illustrates this process:

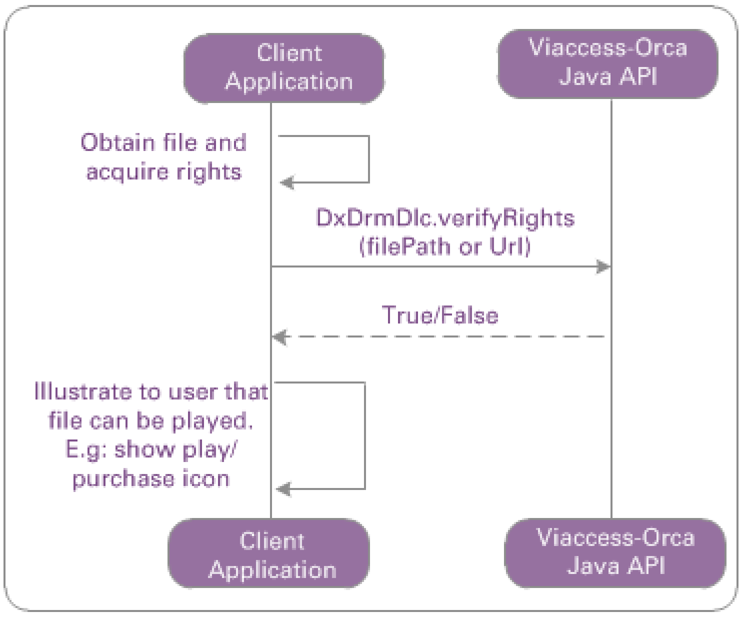
Client Application
Viaccess-Orca Java API
Obtain file and acquire rights
DxDrmDlc.verifyRights (filePath or Url)
True/False
Illustrate to user that file can be played. E.g: show play/ purchase icon
Client Application
Viaccess-Orca Java API

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 `[verifyRightsForFile:]` function on it.

Deleting Content License

If the application needs to remove a previously-purchased license it can call the `[deleteRightsForFile:]` function with either a URI or a filename. The function 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 `[deleteRightsForFile:]` function on it.

Retrieving DRM Version Details

The CSP-SDK provides a function called `[getDrmVersion]` which returns a string that represents the DRM version defined in the build process.

SOAP Error Handling

The CSP-SDK propagates SOAP errors to the Customer Application. The following functions propagate SOAP errors:

- `[executeInitiatorWithFile:error:]`
- `[executeInitiatorWithData:error:]`
- `[executeInitiatorWithURL:error:]`
- `[acquireRightsForFile:withCustomData:withRightsUrl:error:]`

The error parameter is an NSError object containing a dictionary of key/value pairs:

- `SOAP_ERROR_BODY`
- `SOAP_ERROR_REDIRECT_URL`
- `SOAP_ERROR_CUSTOM_DATA`

The functions have to be called within an AutoRelease Pool to prevent any memory leaks.

An NSError object will be created only if a SOAP error exists. The Customer Application should inspect the existence of the NSObject output parameter when the functions return a value other than `DX_MANAGER_SUCCESS`.

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

Content Playback

To play back the content, a valid license should 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 `[verifyRightsForFile:]` function. This function returns `YES` if a license is available, or `NO` 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.

To play the content refer to the *VisualOn Player SDK Integration Guide for iOS.*

Developing the Client Application

This section explains how to integrate and test the Connected Sentinel Player Client SDK 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 Server-Side Personalization SDK API Reference*).

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 additional 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 `[DEBUG_setLocalPersonalization:YES]` (see *Set Client-Side Test Personalization Mode* on page 18) to enter the client-side test personalization mode. A call to `[performPersonalizationWithSessionID:withServerURL:withAppVersion:]` 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 the *Integrating CSP Server-Side* manual. Follow the personalization initiation and verification use-case (described in *Personalization, Verification and Initiation* on page 10) as if working with a regular personalization server.

Debug API

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

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 `[DEBUG_deletePersonalization]`.

Configuring Logs

The logging levels can be configured by calling the `[DEBUG_setLoggingLevel:withLogFile:useStdOutLogs:withDisabledModules:]` static function before the `DxDrmManager` singleton is created. Logs are disabled by default.

For exact details on input parameters refer to the *Connected Sentinel Player SDK iOS API*.

Set Client-Side Test Personalization Mode

The Connected Sentinel Player client-side SDK can be developed and tested 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 `[DEBUG_setLocalPersonalization:YES]` prior to `[performPersonalizationWithSessionID:withServerURL:withAppVersion:]`. For additional information refer to *Developing the Client Application* on page 17.

Appendix

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

Glossary

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

Reference Documentation

- Common Integration Guide
  Reference number: 21816
- Integrating CSP Server-Side
  Reference number: 21826
- Connected Sentinel Player Personalization SDK API
  Contact your VO Technical Engineer for the appropriate version of this document.
- API Reference for CSP Integration with iOS
  Reference number: 21949
- VisualOn Player SDK Integration Guide for iOS (Version: 1.2)
  Provided as part of the CSP 1.x Documentation Package
- Integrating CSP with Harmonic PlayReady Encryption
  Reference number: 21714