SECUREPLAYER SDK

COMMON INTEGRATION GUIDE

VERSION 2.1.1

External Revision History

Rev	Date	Description
0.96	06-Nov-11	First official release.
1.0	20-Mar-12	• New section: *SOAP Error Handling*. • New section: *Delete Rights*. • Modified section: *Identify DRM Protected Media Files*. Alternative method using initiators. • Modified section: *Determine if Valid Rights Exist* Alternative method using initiators. • Modified section: *Supported Playback Scenarios*. Adding progressive download.
1.1	24-May-12	• Modified section: *SecurePlayer Client Application*. Adding notice of HTTPS • Modified section: *Rights Acquisition (PlayReady®)*. Adding notice of HTTPS • Modified section: *SOAP Error Handling*. Rephrasing.
1.2	16-Aug-12	• Modified section: *Identify DRM Protected Media Files*. Reformatting. • Modified section: *Determine if Valid Rights Exist*. Reformatting. • Modified section: *Obtain Rights Information*. Alternative method using initiators.
2.1		• Modified section: *Related Documents*. Adding reference to the documentation by VisualOn.

Contents

List of Figures

List of Tables

1 Introduction

The Discretix SecurePlayer Software Development Kit (SDK) package provides components and tools that allow extending a downloadable application with Secure Media Player playback and Microsoft® PlayReady® DRM capabilities.

This kit targets customers that already have an application that supports purchasing and viewing non-DRM content.

1.1 Intended Audience

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

1.2 Related Documents

Table 1: Reference Documentation

Reference	Document
[PRWP]	PlayReady® Overview White Paper
[SP_ANDR_SDK_IG]	SecurePlayer SDK Android Integration Guide
[SP_ANDR_PROJ_STP]	SecurePlayer SDK Android Project Setup
[SP_ANDR_SDK_API]	SecurePlayer SDK Android API Reference
[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 Server-Side Personalization SDK API Reference
[SP_iOS_PROJ_STP]	SecurePlayer SDK iOS Project Setup
[SP_iOS_SDK_API]	SecurePlayer SDK iOS API Reference
[SP_iOS_IG]	SecurePlayer SDK iOS Integration Guide
[SP_SIG]	SecurePlayer Signing Process Guide
[VO_iOS_IG]	VisualOn Player SDK Integration Guide for iOS (Version: 1.2)
[VO_ANDR_IG]	VisualOn Player SDK Integration Guide for Android (Version: 1.2)
[DX_HAR_PROHLS]	Discretix –Harmonic PlayReady Encryption Integration Guide

1.3 Terms and Abbreviations

Table 2: Glossary

Term	Description
DRM	Digital Rights Management
CA	Certificate Authority
CID	Content ID
API	Application Programming Interface
AV	Audio Video
GUI	Graphical User Interface
NDK	Native Development Kit
SDP	Session Description Protocol
TLV	Type-Length-Value Encoding
SP-SDK	Discretix Secure Player SDK
KID	PlayReady® Key Identifier associates between content and license
CMS	Content Management System

2 Solution Overview

2.1 System Architecture

Figure 1: Eco-System/System Architecture shows a Device-Personalization SDK integrated with a customer server to support over-the-air personalization, and a client-side SDK integrated into the customer application running on devices.

The Discretix solution provides two major components:

- **Discretix SecurePlayer Client SDK**

  This is the client SDK for DRM content playback and license management. The APIs exposed by this SDK are tuned to the relevant device platform. It is intended to be integrated as part of the customer's downloadable application.

- **Discretix Personalization Server SDK**

  This is the server-side component for managing the personalization-protocol messages. This component should be integrated into the customer's user-management HTTP server.

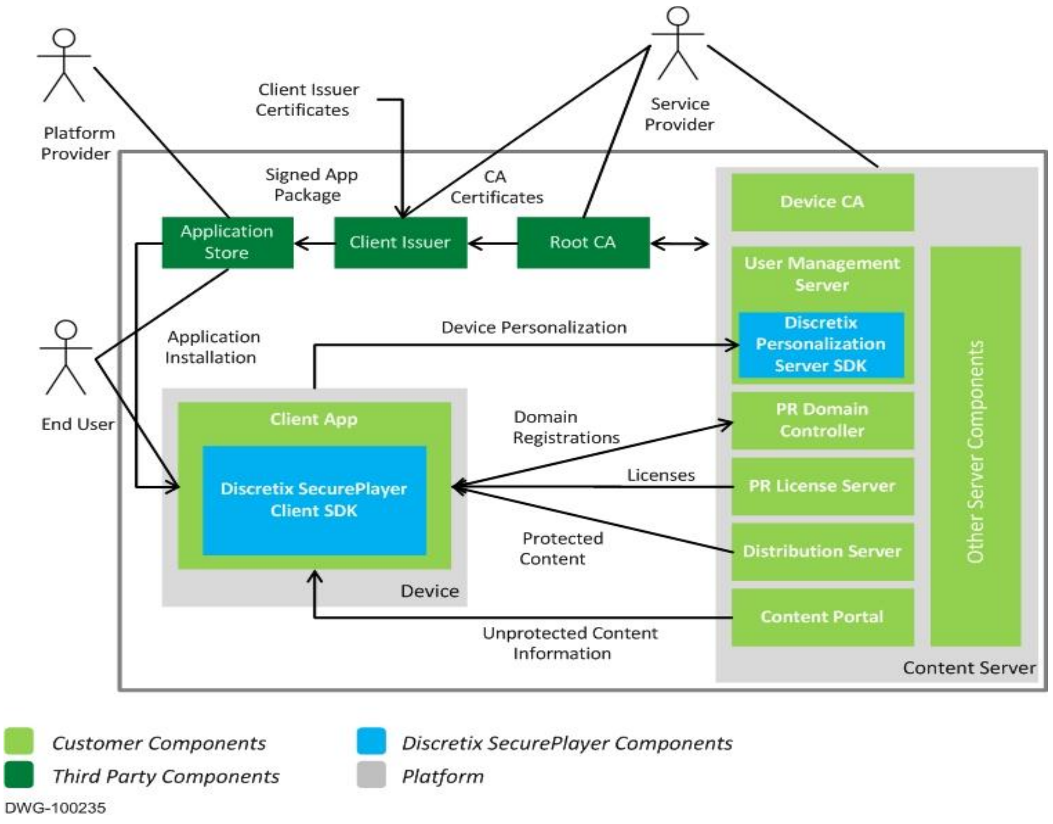

Figure 1: Eco-System/System Architecture

2.1.1 Certificate Authority

The Certificate Authority (CA) is an external component responsible for issuing CA root certificates, rights issuer CA certificates and device CA certificates.

CA root certificates are used for establishing a trust chain between the client and the server, when performing mutual authentication. The CA root certificates are provided to the Client Issuer for inclusion of the certificate in the application package.

The content server certificate is installed on the content server, and used for validating server identity when performing mutual authentication with the client.

The device certificate is used for claiming device identity when performing mutual authentication with the content server. The device certificate is installed on the device as part of the personalization process.

For additional information see referenced documents [PRWP] and [SP_PERS_SDK_IG].

2.1.2 Client Issuer

The Client Issuer is an external component that creates a signed application package containing a downloadable client application. This package is provided to the Application Store. The application package contains a CA root certificate as one of its non-code resources.

The package is signed with the Client Issuer certificate obtained according the Application Store rules, thereby identifying the Client Issuer to the Application Store and the device operating system.

2.1.3 Application Store

The Application Store is an external component that allows discovery of the downloadable client by users, as well as client download and installation.

2.1.4 Content Server

Details of Content-Server implementation are out of the scope of this document.

At the logical level the server comprises of the following functional components:

- **Device Manager** - tracks device certificates, maps devices to user accounts, performs device personalization procedures, and provides device credentials to the Rights Issuer component.
- **PlayReady® License Server** - generates content rights objects for specific devices and performs protocol procedures.
- **PlayReady® Domain Controller** - issues domain certificates and performs registration of devices in the domain.

- **Distribution Server** - serves protected content to the clients in a downloadable file or streaming forms.
- **Content Portal** - provides users with unprotected content information, such as previews, reviews, and ratings, allows selection and acquisition of specific content objects.

  Once the user purchases specific content rights, this information is communicated to the Rights Issuer for generation of rights objects for the user devices.
- **Device CA** - generates device certificates and private keys for the personalization process.

2.1.5 Client Application

The downloadable Client Application is built according to superlative platform procedures, and is installed by the user on the device. The client application usually provides the users with the following functionality:

- An application requiring user-level permissions in order to operate on the device.
- A GUI for the Content Portal and locally-stored content.
- A player capable of DRM content playback.
- DRM management capabilities.
- A financial-transaction backend.

The SecurePlayer SDK is responsible for the DRM management and content playback. All other features are in the responsibility of the customer – application developer.. Nevertheless, there is a direct correlation between application features and the DRM operations provided by the SecurePlayer SDK (e.g., purchasing media is directly associated with rights acquisition). These correlations are out of the scope of this document, and should be discussed directly with Discretix.

3 Typical Integration Process

The following diagram provides an overview of a complete typical integration process. Reading System Architecture first may be helpful for understanding the overall system and the location of the Discretix components within it.

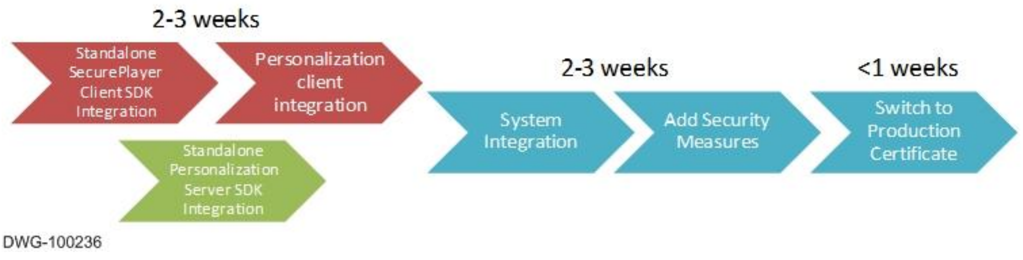
2-3 weeks
Standalone SecurePlayer Client SDK Integration
Personalization client integration
Standalone Personalization Server SDK Integration
2-3 weeks
System Integration
Add Security Measures
<1 weeks
Switch to Production Certificate
DWG-100236

Figure 2: SecurePlayer SDK Integration Flow

Discretix recommends developing and testing the Client Application and the Personalization Server standalone before performing System Integration. These steps include a standard API integration process.

After System Integration, the SDK component is to be replaced by an SDK component version containing production level Personalization secrets.

Final system tests should be performed to ensure meeting the service end-to-end compliancy to the specification.

Prior to deployment the test certificate are switched to Production Certificates.

3.1 Standalone SecurePlayer SDK Client Application Integration

Integration of SP-SDK with the application is done in 3 stages:

1. Signing SO files (Android only)
2. Personalization
3. License Acquisition
4. Content Playback

Android only - A valid signature file supplied by Discretix MUST be present in the application project. If the application requires an additional SO files it MUST go over the procedure described in [SP_SIG].

To develop the SP-SDK Client without requiring a personalization server, use Local Personalization (see the corresponding platform integration guides [SP_ANDR_SDK_IG]/[SP_IOS_SDK_IG]).

Once development of License Acquisition and Content Playback is completed, perform the personalization process with the Discretix Test Personalization Server (see the corresponding platform integration guides [SP_ANDR_SDK_IG]/[SP_IOS_SDK_IG]).

Remember to update the server's database and add a record matching your client's SDK version and HASH code (see [SP_PERS_SDK_IG] - *SecurePlayer SDK version table*).

3.2 Standalone Personalization SDK Integration

Integrating the Personalization SDK into the customer's user-management HTTP server is described in detail in referenced documents [SP_PERS_SDK_IG] and [SP_PERS_SDK_API].

3.3 System Integration

3.3.1 SecurePlayer Client Application

Verify that first parameter of your `performPersonalization` method contains yours personalization server's URL, the `appVersion`, and any additional customer related parameters necessary in the `sessionId`.

NOTE! The personalization server URL can be either HTTP or HTTPS one.

3.3.2 Personalization HTTP Server

Update your server's database and add a record matching your client's SDK version and HASH (see referenced document [SP_PERS_PROJ_STP]).

4 Common Use Cases

The following section describes how to integrate the most common use cases that are usually required by an Application using the SP-SDK.

Use case integration is common to all platforms, and is therefore; described in high–level, without going into specific platform details.

Along with this Common integration guide, the SP-SDK includes the following platform-specific documents:

- Project setup [SP_ANDR_PROJ_STP]/[SP_iOS_PROJ_STP]
- Integration guide [SP_ANDR_SDK_IG]/[SP_iOS_IG]
- API Reference guide[SP_ANDR_SDK_API]/[SP_iOS_SDK_API]

NOTE! Reviewing the platform-specific integration guide is required for a full understanding of the client integration.

4.1 Personalization Process

The goal of the personalization process is to provision the Client Application with credentials (certificates and keys) in a secure manner. These credentials are used to securely deliver licenses from the Microsoft® PlayReady® License Server to the Client Application. Typically, personalization occurs once in the lifecycle of the Client software, unless the software version is updated.

As an additional effect of the protocol, the Personalization Server gathers information about the client's user, device, and OS.

When the client software is activated for the first time, the client and the service provider's server perform the client personalization process. The following provides a brief overview of this process:

1. The client software generates a challenge that includes user information, and sends it to the server.

   NOTE! The server URL is already known by the client application.

2. The Personalization Server processes the challenge, and validates some information.
3. The Personalization Server then generates a set of device-specific credentials, uniquely identifying the user account, and containing a device private key and certificates. For additional information see referenced document [SP_PERS_SDK_IG].
4. The device-specific credentials are securely transferred to the user's device and securely provisioned in the device.
5. Following the completion of the personalization procedure, the client enters normal operation mode, which allows users to browse, purchase and playback video content.

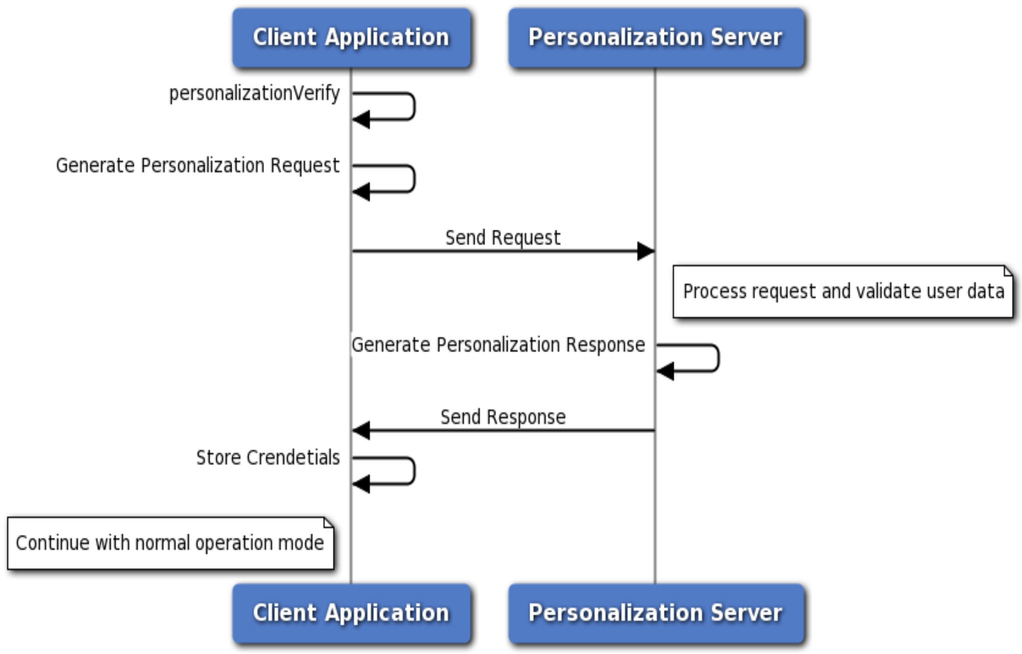

Figure 3: General Personalization Flow

4.2 SecurePlayer SDK Application Update

An application software update is sometimes required. This may be due to a security breach, compatibility issues, or another reason.

NOTE! The actual software update mechanism varies per platform, and is the customer application's responsibility (out of the scope of this document).

4.2.1 Update Required Notification

The SecurePlayer SDK defines only one way by which to notify that an update is required. This occurs during the `performPersonalization` process, when an `updateRequired` exception is thrown.

NOTE! The customer application can define additional ways to require software updates.

The device assets are not provided until the SecurePlayer software version is updated and `performPersonalization` is performed again.

4.2.2 Keeping Licenses through Software Updates

The application developer should keep in mind that if application data is erased during an update, all stored licenses will be lost. In other words, in order for the user to maintain his or her acquired licenses over a software update, the application data must not be erased.

4.3 Rights Acquisition (PlayReady®)

The SecurePlayer performs rights acquisition according to the protocol defined by Microsoft® PlayReady®, and is fully compliant with it.

To acquire rights for protected content, the SecurePlayer requires the PlayReady® DRM header object, which is usually embedded within the PlayReady® protected content (depends on the content format). The header object contains the rights acquisition URL (`LA_URL`) and the KID for which to acquire the rights.

NOTE! The rights acquisition URL can be either HTTP or HTTPS one.

In most cases, the SecurePlayer can extract the header object from the protected content, and the application is not required to do any parsing.

There are two ways to conduct the rights acquisition: content-based and initiator-based.

4.3.1 Content-Based Rights Acquisition

NOTE! This way is not suitable for HLS in Harmonic format. A solution to this limitation is described in [DX_HAR_PROHLS] chapter 8.

In this case, the SP-SDK retrieves the PlayReady® DRM header object from the protected content. Based on the header, it conducts the rights acquisition. This use case is suitable mainly for local protected content. However, it is also applicable for protected streams (see 4.3.3 Rights Acquisition for a Protected Stream).

Rights acquisition is carried out by calling the `AcquireRights` method. This method receives the local content path as a parameter, extracts the PlayReady® header and conducts the license acquisition process.

The `AcquireRights` method additionally accepts the following optional parameters:

- **Custom data** – This data will be sent to the license server through the license challenge, as defined by the PlayReady® license acquisition protocol.
- **Rights URL** – Setting this URL will override the rights URL (`LA_URL`) defined within the header object.

4.3.2 Initiator-Based Rights Acquisition

In this case, rights are acquired by processing a PlayReady® license acquisition initiator. This use case is suitable for both local protected content and protected streams.

The initiator XML contains a PlayReady® DRM header object. Rights are acquired based on it.

The rights are associated with a protected content by the KID in the header object. It is up to the application to make sure that the initiator correlates with the protected content it is intended for.

To learn more about Initiators, see 4.4 Execution of PlayReady® Initiators.

NOTE! It is possible to process other types of initiators (e.g., join/leave domain, metering and combined initiators).

4.3.3 Rights Acquisition for a Protected Stream

NOTE! This way is not suitable for HLS in Harmonic format. A solution to this limitation is described in [DX_HAR_PROHLS] chapter 8.

Acquiring rights for a protected stream is slightly more complicated than for local content, as the stream is not available locally on the device, and therefore there is no PlayReady® header object to acquire rights for. The easiest way to overcome this is by using a license acquisition initiator (see 4.3.2 Initiator-Based Rights Acquisition). However, this may not always fit the customers' needs.

An alternative is to use a protected stream identifier. This identifier can be used as the local content path for the `AcquireRights` method (see 4.3.1 Content-Based Rights Acquisition).

The protected stream identifier needs to be retrieved by the application, usually by downloading it from the content server.

There are several types of identifiers. Each stream type defines its own identifier:

- **Protected Smooth Streaming**: the Manifest file should be used as a stream identifier.
- **Protected HLS** (as defined by Discretix PlayReady® over HLS format ver. 2.1 or ver. 3.0): the playlist file (`.m3u8`) should be used as a stream identifier.

  NOTE! The playlist must contain either the `#EXT-X-DXPLAYREADY` (ver. 2.1) or `#EXT-X-DXDRM` (ver. 3.0) attribute. This is usually part of the leaf playlist.
- **ISMV Progressive Download:** There are two options for stream identifier:
  - Manifest file (if exists).
  - ISMV file prefix – must contain the "moov" box.
- **Envelope Progressive Download:** the Envelope header should be used as a stream identifier. Header size can be up to 20KB.

4.4 Execution of PlayReady® Initiators

The PlayReady® Initiator mechanism is an inherent part of the Microsoft® PlayReady® DRM. Initiators are used to initiate and carry out DRM operations.

There are several types of initiators:

- **License Acquisition** – acquire rights for a content referenced by the PlayReady® DRM header object within the initiator. This operation conducts a license acquisition process with the license server.
- **Join/Leave a Domain** – Join (or leave) the domain defined within the initiator.
- **Metering** - Send content usage information to the server.

- **Combined** – An initiator containing one or more of the initiators defined above. The DRM operations defined by the initiators are carried out synchronously.

The application can execute an initiator by calling the `ExecuteInitiator` method. This method receives the initiator path as a network address, or as a local file.

4.5 Obtaining Rights Information

The application may be required to query about available content files and licenses.

NOTE! When using Harmonic HLS format only the alternative method (using the initiator) will work (referring to sections 4.5.1, 4.5.2 and 4.5.3). A solution to this limitation is described in [DX_HAR_PROHLS] chapter 8.

4.5.1 Identify DRM Protected Media Files

To ascertain whether a specific media file is a DRM protected content, call the `IsDrmContent` method.

An alternative method to identify whether a specific media file is a DRM protected content is to download the initiator (XML-based CMS file) used to acquire rights for the same content and apply the `IsDrmContent` method to it.

4.5.2 Determine if Valid Rights Exist

To ascertain whether a protected content has valid rights for playback, call the `VerifyRights` method.

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 `VerifyRights` method to it.

4.5.3 Obtain Rights Information

It is possible to obtain the rights information associated with a protected content. A protected content can have several licenses with various restrictions.

NOTE! A protected content may have a license with no restrictions.

The restrictions are:

- **Time constraint**: the media can be played from a start date until an end date. Before the start date and after the end date the media has no valid rights.
- **Interval constraint**: the media may be played for a time interval starting from the time it is first played. When the interval license is consumed for the first time, it transforms to a time-constraint license.
- **Count constraint**: the media may be played a given number of times.

NOTE! In PlayReady®, there is no real count constraint. Only an abstraction of a short interval which will appear as 1 count before consumed. After consumption it may transform into a short time constraint.

Retrieving the rights information is done by calling the `GetRightsInfo` method. This method receives the protected content path and returns an array of rights info objects.

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

4.6 Delete Rights

NOTE! When using Harmonic HLS format only the alternative method (using the initiator) will work. A solution to this limitation is described in [DX_HAR_PROHLS] chapter 8.

The SP-API supplies a `DeleteRights` method to delete rights of a specific content.

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 `DeleteRights` method to it.

4.7 Content Playback

Playback of content with the SecurePlayer is done by the VisualOn Software Player. For more information see the relevant referenced document [VO_iOS_IG] / [VO_ANDR_IG].

A precondition for playing protected content is that valid rights must exist for it (see 4.3 Rights Acquisition (PlayReady®)). It is possible to determine if valid rights are available for protected content (see 4.5 Obtaining Rights Information). If rights do not exist, playback will fail.

The Discretix SecurePlayer supports several playback scenarios, as described in the following sections.

4.7.1 Supported Playback Scenarios

The following table describes the supported playback scenarios:

Table 3: Supported Playback Scenarios

Playback Scenario	Description	Supported formats
Local playback	Playback of local file stored on the device.	• Envelope • ISMV
Adaptive Streaming	Playback of content hosted on a remote streaming server. Dynamically monitoring local bandwidth and CPU	• Smooth Streaming • HLS

Playback Scenario	Description	Supported formats
	utilization allows playback optimization by switching video quality in real-time.	
Progressive Download	Playback of local file currently being downloaded by the Client Application to the device. *ISMV limitation – cannot perform seek until the file is fully downloaded.	• Envelope • ISMV

4.7.2 Supported V/C Formats

The following table shows a complete list of supported content containers:

Table 4: Supported Container Formats

Content Container	Description	Platform availability
Envelope	Play Ready® content container wrapping any type of content file (e.g., 3GP with H.264). Not optimal for streaming, except MMS/HTTP.	Android, iOS
Smooth Streaming	Content container based on PIFF (ISMV). Used for smooth streaming.	Android
HLS	HTTP-based streaming protocol (defined by Apple). Discretix extends this protocol to support PlayReady® content protection.	Android, iOS

NOTE! Supported content formats may vary between different flavors of the SecurePlayer SDK.

4.8 SOAP Error Handling

The PlayReady® Server may send SOAP error messages upon errors, such as failure to acquire rights. The SOAP error may include custom data including service specific information on the error for the use of the application. Configuring the PlayReady® Server to respond with the SOAP errors is out of scope of this document.

The entire SOAP error is available to the customer application. For convenience the redirect URL and custom data parameters which are part of the SOAP error are also available separately.

The customer application may handle the SOAP error in any way required.