Integrating CSP Server-Side

Connected Sentinel Player 1.x

viaccess·orca

copyright

trademarks

Document reference number: 21826

Document version number: 1.0 Release

Contents

Introduction

This document explains how to utilize the Personalization SDK Dynamic Link Library as part of a User-Management Server and describes the necessary components to be added. It also describes how to setup the Viaccess-Orca Personalization Package.

The Personalization Package includes:

- the Personalization SDK.
- the Personalization Test Utilities:
  - *Personalization Test Client (used by server-side developers to simulate proper client behavior).*
  - *Personalization Test Server (used by client-side developers to simulate proper server behavior).*

Overview of the Integration Process

A client application using the Connected Sentinel Player SDK requires unique device assets in order to implement the DRM functionality. These assets include (PlayReady®) certificates and private keys.

The personalization process has two roles in the Connected Sentinel Player ecosystem:

- Provide the client application with the device assets through a secure protocol.
- Monitor the activation of the different applications using the Connected Sentinel Player.

The device assets are unique per device and are always protected in the device memory. When they are stored in persistent device memory, they are encrypted with a unique device key.

The actors in the Personalization mechanism are:

- The Connected Sentinel Player SDK
- The client application running on a mobile device (e.g., Android® and iPhone®)
- The User-Management Server.

The Connected Sentinel Player SDK Client API supplies two functions related to Personalization:

- `VerifyPersonalization`

  The function checks whether the encryption keys are already stored on the device and accessible for the Connected Sentinel Player Client application. This operation is local on the mobile device and does not require any communication with the User-Management Server.

- `PerformPersonalization`

  The function initiates communication between the Connected Sentinel Player Client application and the User-Management Server in order to obtain the encryption keys.

> **note**
>
> If `VerifyPersonalization` indicates that the device assets are available then the `PerformPersonalization` call is not required. The `VerifyPersonalization` should be used to ensure that the Personalization mechanism is activated only once after first launch.

Target Audience

The document is intended for software engineers and architects who are required to design and set up a User-Management Server.

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 *Glossary* on page 39 at the end of the manual for a complete definition.

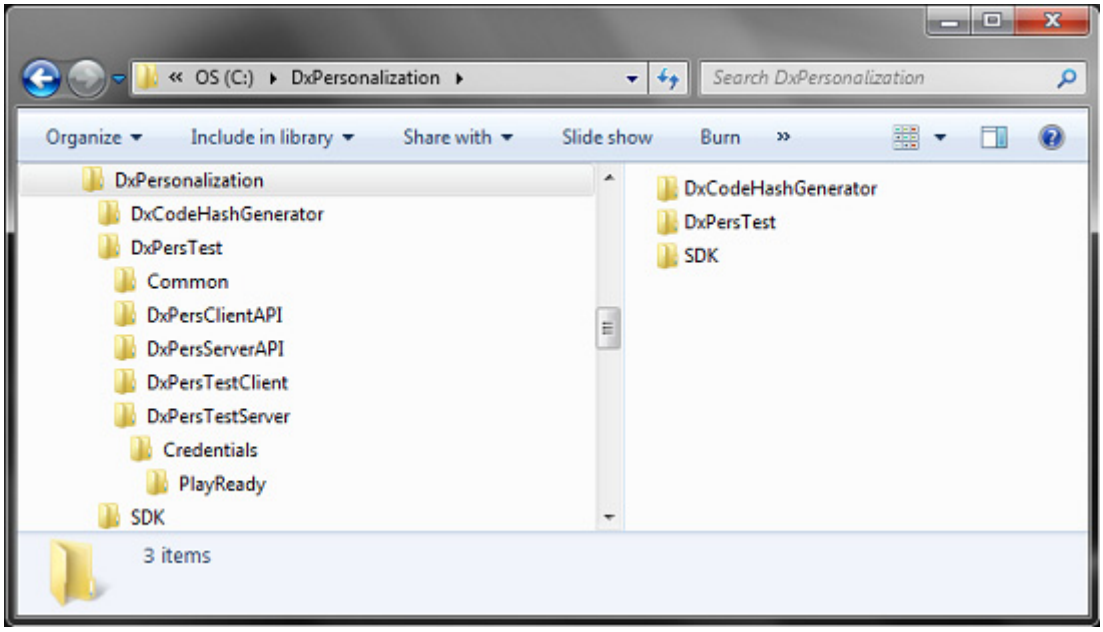
part 1: Project Setup

Chapter 1: Personalization Package Content

The Personalization Package supplied is a DxPersPkg.zip file to be unzipped on the computer hard drive. Once unzipped, this file contains the following structure:

- `Setup.exe` file.
- `DxPersonalizationSetup.msi` file.
- Folder `vcredist_x86` containing a single file `vcredist_x86.exe`.

Running the `setup.exe` (Vanilla setup) will install the Personalization Package components as follows:



Personalization SDK

The Personalization SDK is located under `c:/DxPersonalization/SDK` and contains the following files:

- `DxPersServer.dll`
- `DxPersServer.lib`
- `DxTypes.h`
- `DxPersDefines.h`
- `DxServer.h`
- `DxSecretInfo.h`
- `DxClientInfo.h`

Test Utilities: Personalization Test

The Personalization Test is a utility containing a Personalization Test Server and a Personalization Test Client.

The Personalization Test is located at `c:/DxPersonalization/DxPersTest` and contains the following:

- `DxPersTestServer.bat` batch file to run the Personalization Test Server (the installation places a shortcut to the batch from the desktop).
- `DxPersTestClient.bat` batch file to run the Personalization Test Client (the installation places a shortcut to the batch from the desktop).
- Folder `c:/DxPersonalization/DxPersTest/DxPersTestServer` containing:
    - `configFile.cfg`
    - `DeviceID.cfg`
    - `ServerDB.cfg`
    - *Folder* `c:/DxPersonalization/DxPersTest/DxPersTestServer/Credentials` *contains internal files.*
    - *Other internal files.*
- Folder `c:/DxPersonalization/DxPersTest/DxPersTestClient` containing:
    - `configFile.cfg`
    - `DeviceID.cfg`
    - *Other internal files.*
- Folder `c:/DxPersonalization/DxPersTest/Common` containing internal files.
- Folder `c:/DxPersonalization/DxPersTest/DxPersServerAPI` containing internal files.
- Folder `c:/DxPersonalization/DxPersTest/DxPersClientAPI` containing internal files.

Chapter 2: Personalization Package Setup Steps

Prerequisites

- Fully checked on Windows XP, Windows 7.
- Should be supported on every win32 environment.

Install

1. Unzip the `DxPersPkg.zip` somewhere on the computer hard drive.
2. Run `setup.exe`.

During the installation you might be required to restart the computer. After it restarts, run `setup.exe` again.

To avoid installing Visual Studio Runtime libraries only in systems where these libraries already exist, run `DxPersonalizationSetup.msi`.

Uninstall

1. Go to the **Add and Remove Programs** via the Control Panel.
2. Select **Personalization Package**.
3. Click the **Remove** button.

Chapter 3: Setup Personalization SDK

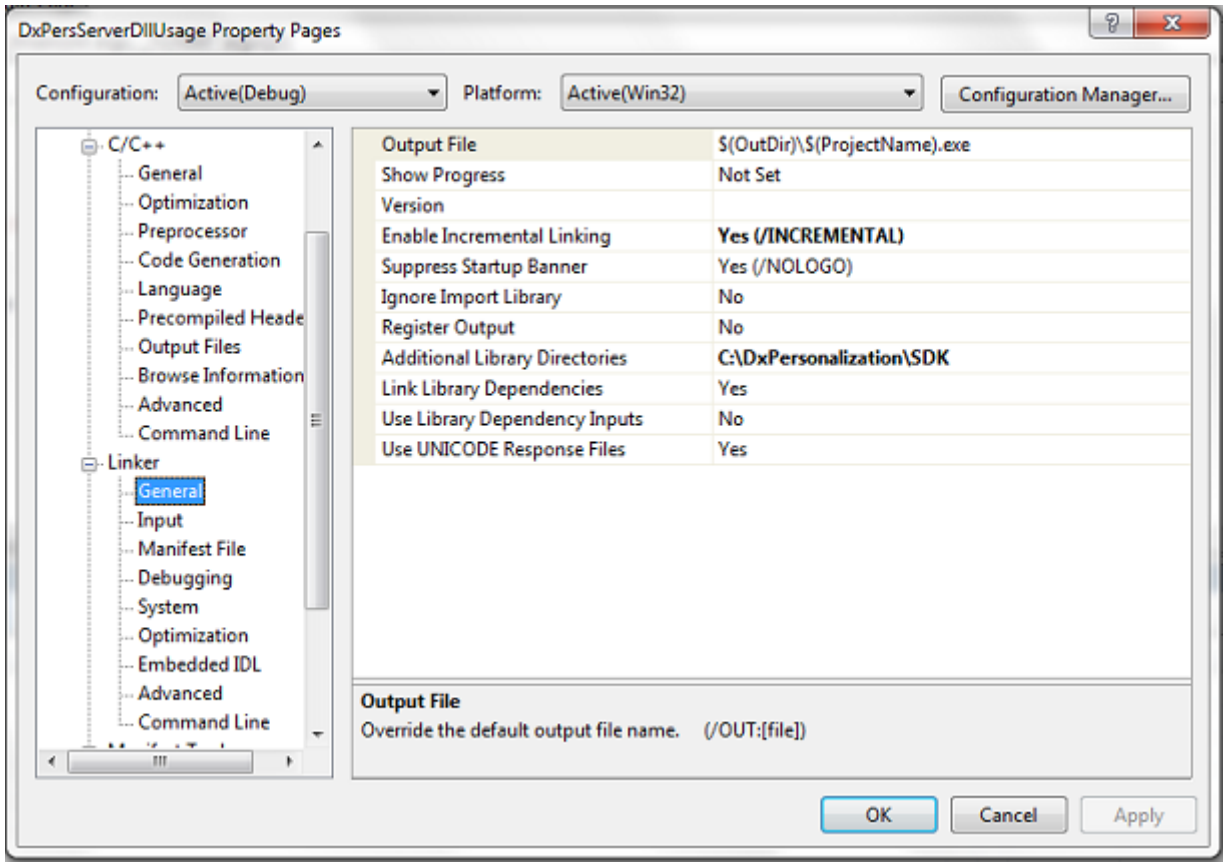
Prerequisites

- Fully checked on Windows XP, Windows 7.
- Should be supported on every win32 environment.

Setup steps

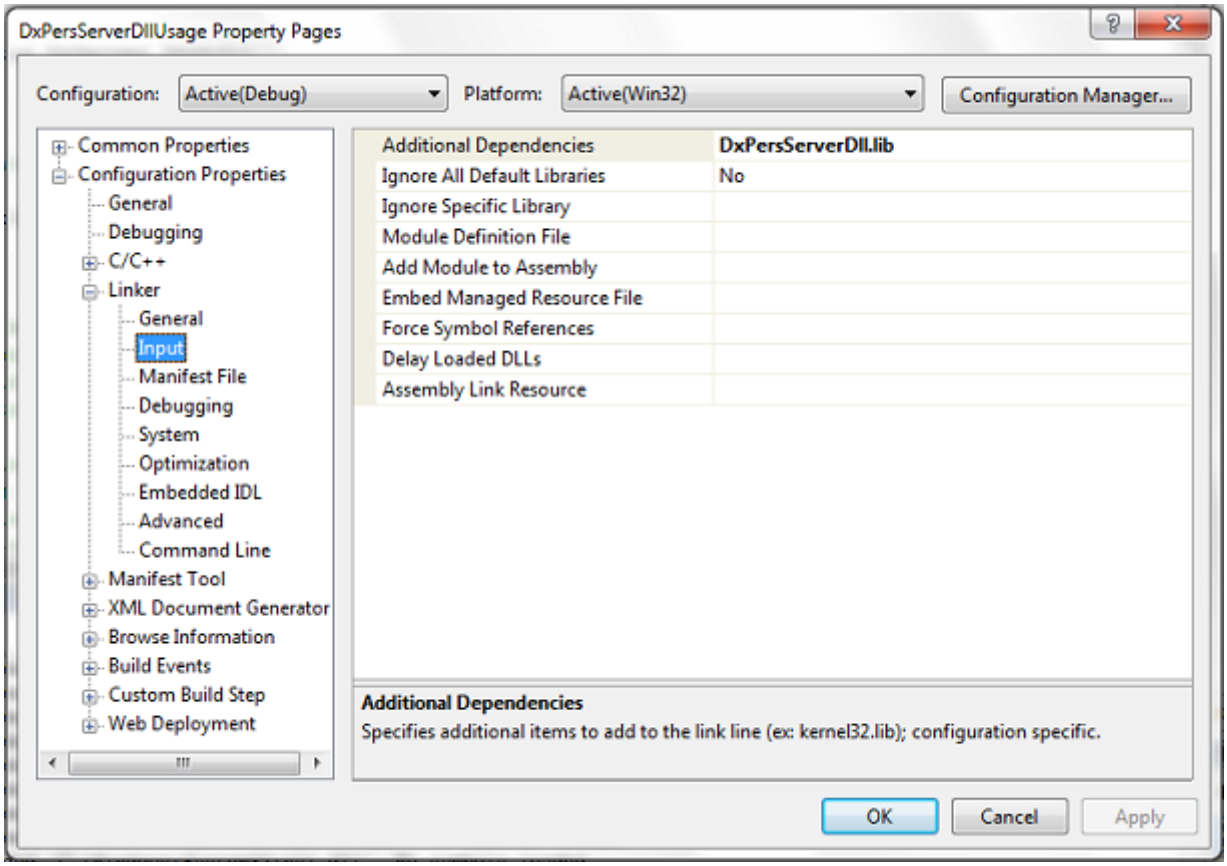
This section describes how to set up a sample project that uses the `DxPersServer` DLL. Windows 7 with Visual Studio 2005 are recommended, but not mandatory.

1. Open your project in Visual Studio (project intended to use the `DexPersServer` DLL).
2. Open project properties.
3. Select: **Configuration Properties > Linker > General**.



4. In **Additional Library Directories**, add the DLL location: `C:\DxPersonalization\SDK`.

5. Select **Configuration Properties > Linker > Input.**



6. In **Additional Dependencies**, add the library `DxPersServerDll.lib`.
7. From the source code include the `DxServer.h` file.
8. In the Environment Variables add to the system PATH the location of the `DxPersServer.dll`: `C:\DxPersonalization\SDK`

Enable Logs

1. In the folder of your project, create a text file named `DxDrmConfig.txt`.
2. Edit the file with `ServerLogPath = <relative path>/PersServerLog`

   For example:

   ```
   ServerLogPath = Logs/PersServerLog
   ```
3. Run the application.

   Following the example, after the application calls `DxServer_Init()`, you should see the in the folder of your project a newly created `Logs` folder with a log file: `PersServerLog_20110427_123421_0622B7BA_376.DxLog` (the additional strings after `PersServerLog_` are merely a timestamp).

Chapter 4: Test Utilities

There are three Python®-based utilities associated with personalization that should be used as part of the development and integration processes.

Personalization Test Server

Complete the Python source code of the personalization test server. The Python server receives client personalization requests, processes them and generates personalization responses. For more information about the Personalization Process, refer to *Integrating CSP with Android*.

The Personalization Test Server is used by the Client Developer Team to verify the client behavior with a Personalization server.

Prerequisites

- WIN32 Environment
- Python 2.5.2

Setup Steps

1. Install Python 2.5.2 **32Bit** – target to `C:\Python25`.
2. Configure the `configFile.cfg` text file located on `C:\DxPersonalization\DxPersTest\DxPersTestServer`:

```
[IP Address]
IP-server=localhost
Port=8000

[URL server path]
UrlServerPath=/Personalization

[Log File Path]
LogFilePath=C:\DxPersonalization\DxPersTest\DxPersTestServer\DxLog\
```

   - `IP-Server` should contain the current machine IP.
   - `Port` should contain the port on which the client and server communicate. Make sure IT opens the ports on the relevant IP addresses.
   - `UrlServerPath` contains name representing the personalization request.
   - `LogFilePath` should contain the path where logs will be placed in.

3. Run the application by running the batch file `DxPersTestServer.bat` located at `C:\DxPersonalization\DxPersTest\DxPersTestServer`. The batch file has a shortcut placed on the desktop after installation.

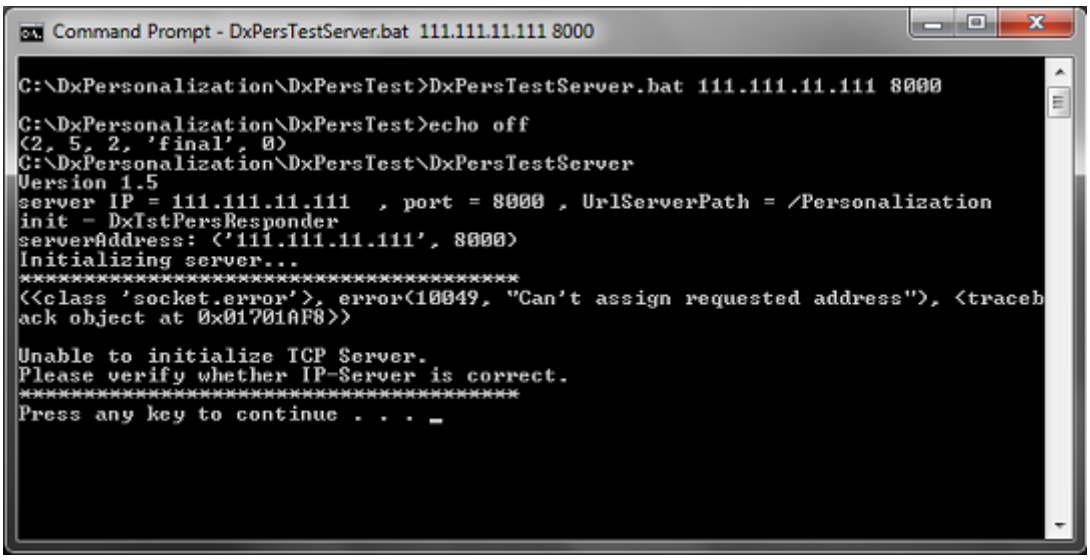
   You may overwrite the IP-Server and Port by adding them as arguments to the `DxPersTestServer` batch file. For example:

```
DxPersTestServer 172.16.8.137:8000

DxPersTestServer 172.16.8.137:8000
```

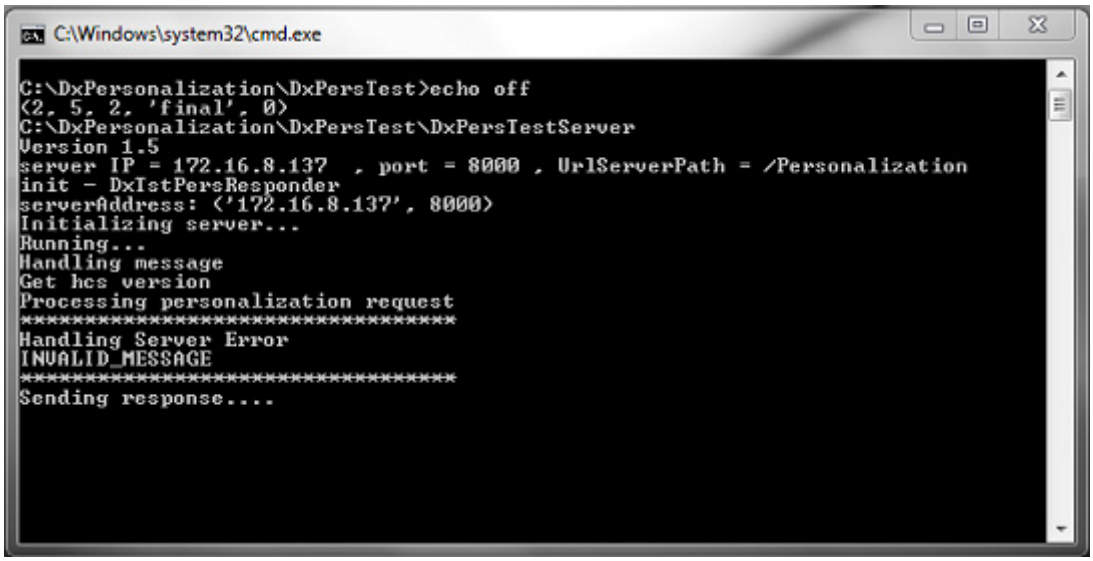
Troubleshooting

In case you get the following error:



Check whether the IP-Server in the `configFile.cfg` file has the IP address of your machine.

In case you get the following error:



Check whether the mapping in the `ServerDB.cfg` file is valid.

Personalization Test Client

Complete the source code of the Python personalization test client.

The personalization test client is able to generate and send personalization requests periodically, and process the server response.

For more information refer to *Integrating CSP with Android*.

The Personalization Test Client is used by the Server Developer Team to verify the server behavior with a Personalization request sent by clients.

Prerequisites

- WIN32 Environment
- Python 2.5.2

Setup Steps

1. Install Python 2.5.2 – target to `C:\Python25`.
2. Configure the `configFile.cfg` text file located at `C:\DxPersonalization\DxPersTest\DxPersTestClient`:

```
[IP Address]
IP-server=localhost

Port=8000

[URL server path]
UrlServerPath=/Personalization

[Log File Path]
LogFilePath= C:\DxPersonalization\DxPersTest\DxPersTestClient\DxLog\
```

   - `IP-Server` should contain the Personalization Server IP address.
   - `Port` should contain the port on which the client and the server communicate. Make sure the IT opens the ports on the relevant IP addresses.
   - `UrlServerPath` contains the URL of the personalization request name. It needs to be identical as defined in the configuration file of the personalization test server.
   - `LogFilePath` should contain the path where logs will be placed in.
3. Run the application by running the batch file `DxPersTestClient.bat` located at `C:\DxPersonalization\DxPersTest\DxPersTestClient`. The batch file has a shortcut placed on the desktop after installation.

   You may overwrite the IP-Server and Port by adding them as arguments to the `DxPersTestClient` batch file. For example:

   - `DxPersTestClient 172.16.8.137:8000`
   - `DxPersTestClient 172.16.8.137 8000`

part 2: Integration

Chapter 5: User-Management Server Internals

User-Management Server Internals

The User Management server is the service entity that manages the business logic including the user registration, subscription and user rights. It holds a persistent storage for user information.

The User-Management Server is in essence an HTTP Server using among other things the Viaccess-Orca Personalization SDK. The Personalization SDK is a Dynamic Link Library offered as part of the Personalization Package.

For more information, refer to the Personalization Package Setup to learn how to set up the Personalization SDK within a Microsoft Development Project.

> **note**
>
> The Personalization SDK is the only component within the User-Management Server provided by Viaccess-Orca.

Block Diagram

The following diagram illustrates the user management server and the personalization library

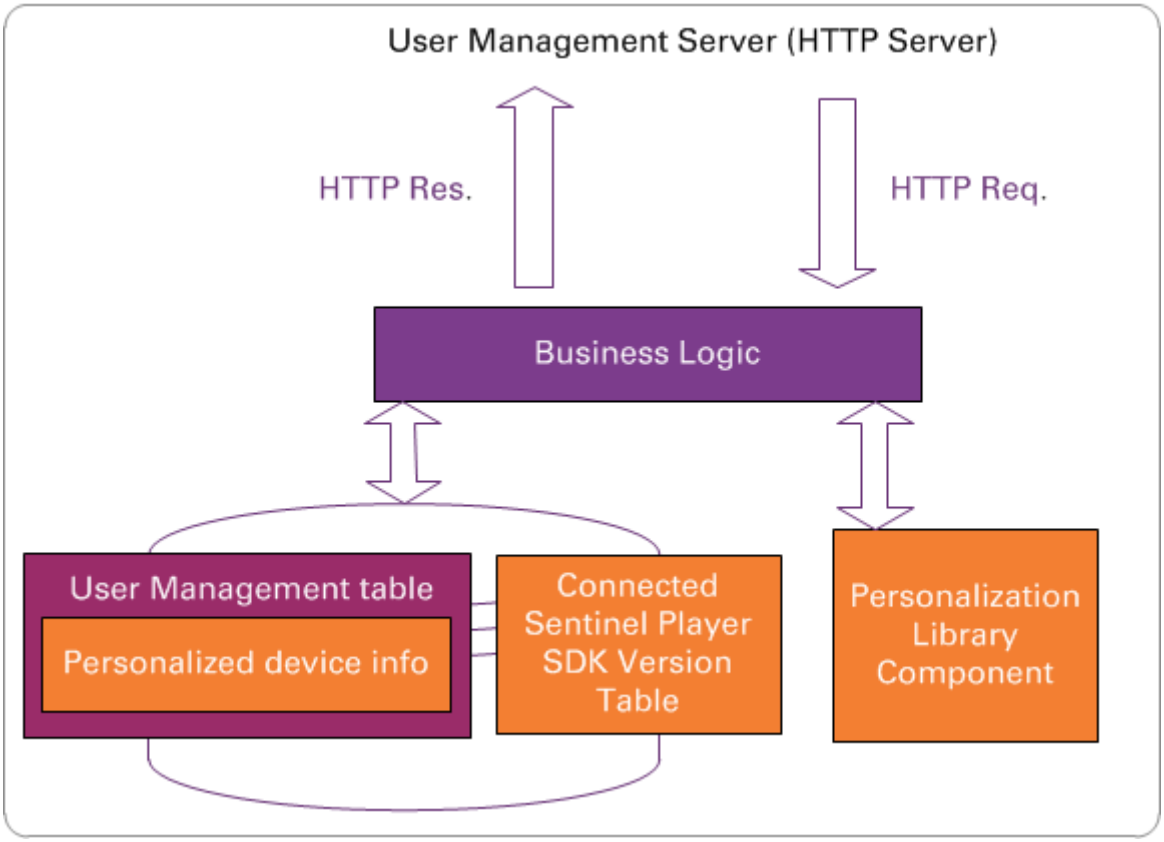


figure 1. User-Management Server and Personalization Library

Personalization Protocol

The Personalization has two protocol versions.

Personalization Protocol Version 1

The Personalization Protocol Version 1 dictates that the Personalization Server-Side SDK will generate, upon a valid client request, a response containing the Model Certificate. This version also dictates that when the client receives the response it will generate the PlayReady® and WMDRM® device certificates and sign and encrypt keys.

This version is referred to as Client Side Certificate Generation (CSCG) since the device certificates are generated on the client side.

Personalization Protocol Version 2

The Personalization Protocol Version 2 dictates that the Personalization Server-Side SDK will generate PlayReady and WMDRM Device certificates and sign and encrypt keys and send them in the response to the client. The client will be required only to store these assets.

> **note**
> The aim is to have all clients work with the Personalization Protocol Version 2 which is more secure.

This version is referred to as Server Side Certificate Generation (SSCG) since the device certificates are generated on the server side.

Persistent Storage

The Persistent Storage is any type of persistent storage that is accessible by the User-Management Server code.

The Personalization SDK requires the following information to be stored in the persistent storage:

- Connected Sentinel Player SDK Version Table – storing version specific information on the supported client versions.
- PlayReady Assets.
- Personalized device information.

Connected Sentinel Player SDK Version Table

The Connected Sentinel Player SDK Version Table is a table maintained in any way (file, database, etc.) by the customer storing information about the various Connected Sentinel Player SDK clients to be handled.

The minimal Version Table structure requirements and examples are available on table – Minimal Connected Sentinel Player SDK Version Table.

DRM Core Version [String]	Platform Name [String]	Platform Version [String]	Code Hash [16 Byte Hex buffer]	Secret Personalization Key [16 Byte Hex buffer]
GENRAL_ANDR_-CLP_PR_D_1_1_51_0	android	9	de6c3f8ae984d2e1a4b79055f2134e75	0404040404040404 0404040404040404
GENRAL_ANDR_-CLP_PR_D_1_1_51_0	android	10	de6c3f8ae984d2e1a4b79055f2134e75	0404040404040404 0404040404040404

GENRAL_IOS_N-P_PR_D_1_1_52_0	iOS	4.3.3	de6c3f812123d2e1a4b79055f2134e75	0404040404040404 0404040404040404

Every new Connected Sentinel Player SDK supplied to a customer will include a new row in the Connected Sentinel Player SDK version table (CSPV table).

The customer can extend the table by adding other columns such as Application Version.

As a minimal requirement, every version must contain the CSPV table configuration information for the Viaccess-Orca Connected Sentinel Player:

- Drm Core Version
- Platform: the given Connected Sentinel Player supports: Platform name + Platform Version.
- Secret Personalization Key is provided per customer.

Platform Name and Version

The platform name and version are mandatory values within the Connected Sentinel Player SDK Version table. For each platform name and version supported, an entry must be added to the version table.

Field name	iOS	Android
Platform name	iOS	android
Platform version	Obtained via: `[[UIDevice currentDevice] model]`	Obtained via: `Build.VERSION.SDK`

PlayReady Assets

Viaccess-Orca provides the customer with secret PlayReady Assets to enable the personalization process and thus enable the end user to acquire licenses and play encrypted content.

Asset name	Also referred as	Description
Certificate Template	`bgroupcert.dat`	PlayReady model certificate used to generate the final device certificate.
Model Key	`zgpriv.dat`	PlayReady private key of the model certificate. It is used to sign the final device certificate.
WM certificate template	`devcerttemplate.dat`	WMDRM certificate template used to generate the WMDRM device certificate.
WM key	`priv.dat`	WMDRM Group key and fallback key.

note

The WMDRM assets are an integral part of the PlayReady ecosystem. The Connected Sentinel Player SDK cannot perform without them.

note

Previous PlayReady Assets included priv_group.dat and priv_fallback.dat – These files combined are in essence the priv.dat.

Personalized device information

During the Personalization process there are two blobs of information that should be acknowledged:

DxClientInfo

The DxClientInfo is a blob containing information about the client – It includes Personalization protocol version, Drm Core Version, Platform name, platform version and other optional parameters.

These parameters are stored in the DxClientInfo in a key-value pair and may be extracted easily.

The DxclientInfo must be stored by the business logic for the Royalty Report. For more information refer to *Royalty Report* on page 29.

DxSecretInfo

The DxSecretInfo is a blob containing the secret PlayReady and WMDRM Device Certificates, Sign and Encrypt keys. This blob is generated during the `GeneratePersResponse` function API.

The DxSecretInfo is relevant only in Personalization Protocol Version 2.

The DxSecretInfo will not be generated by the `GeneratePersResponse` if the client is using Personalization Protocol version 1.

It is optional to store the DxSecretInfo and thus to enable an option of backup in the Business Logic in the future.

note

Currently the Personalization API does not support any usage of the stored SecretInfo.

Business Logic

The Business Logic is the module that receives incoming HTTP POST requests, calls the Personalization SDK functions and looks up columns within the Persistent Storage to generate valid responses.

note

The Business Logic is not provided by Viaccess-Orca and it is up to the customer to implement it.

The HTTP POST request is sent following the Client Application call for `PerformPersonalization`.

The following table describes the `PerformPersonalization` function arguments.

Argument	Mandatory /optional	Description
ServerUrl	Mandatory	The DNS name or IP Address of the Personalization Server.
SessionId	Optional	Customer additional data. The `SessionId` passed via the protocol is encrypted.
ApplicationVersion	Optional	Customer Application version using the Connected Sentinel Player SDK. The `ApplicationVersion` passed via the protocol is not encrypted.

The following figure illustrates the business logic personalization process as a sequence diagram and *Business Logic Flowchart* on page 24 illustrates it as a flowchart.

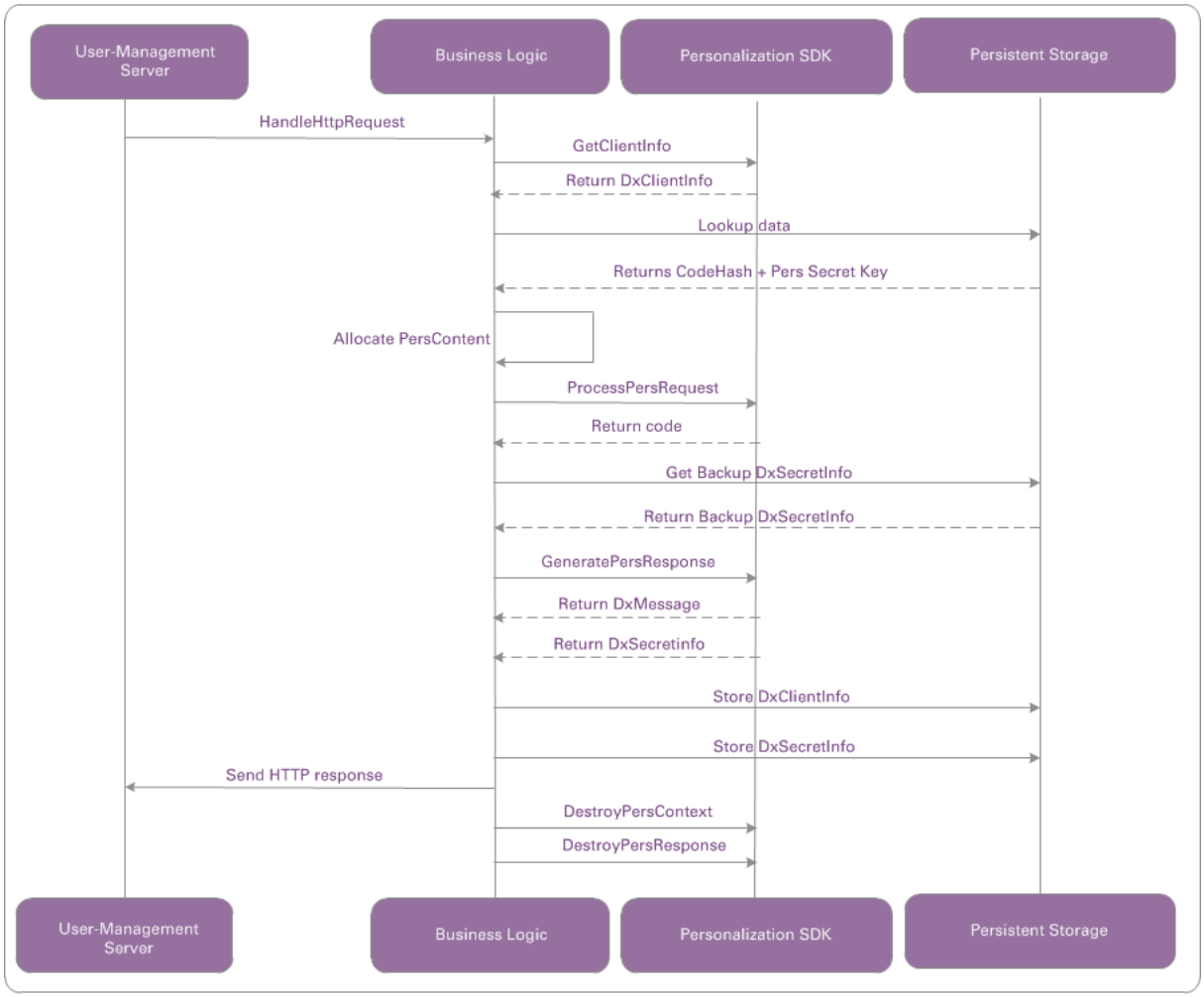


figure 2. Business Logic Personalization Sequence

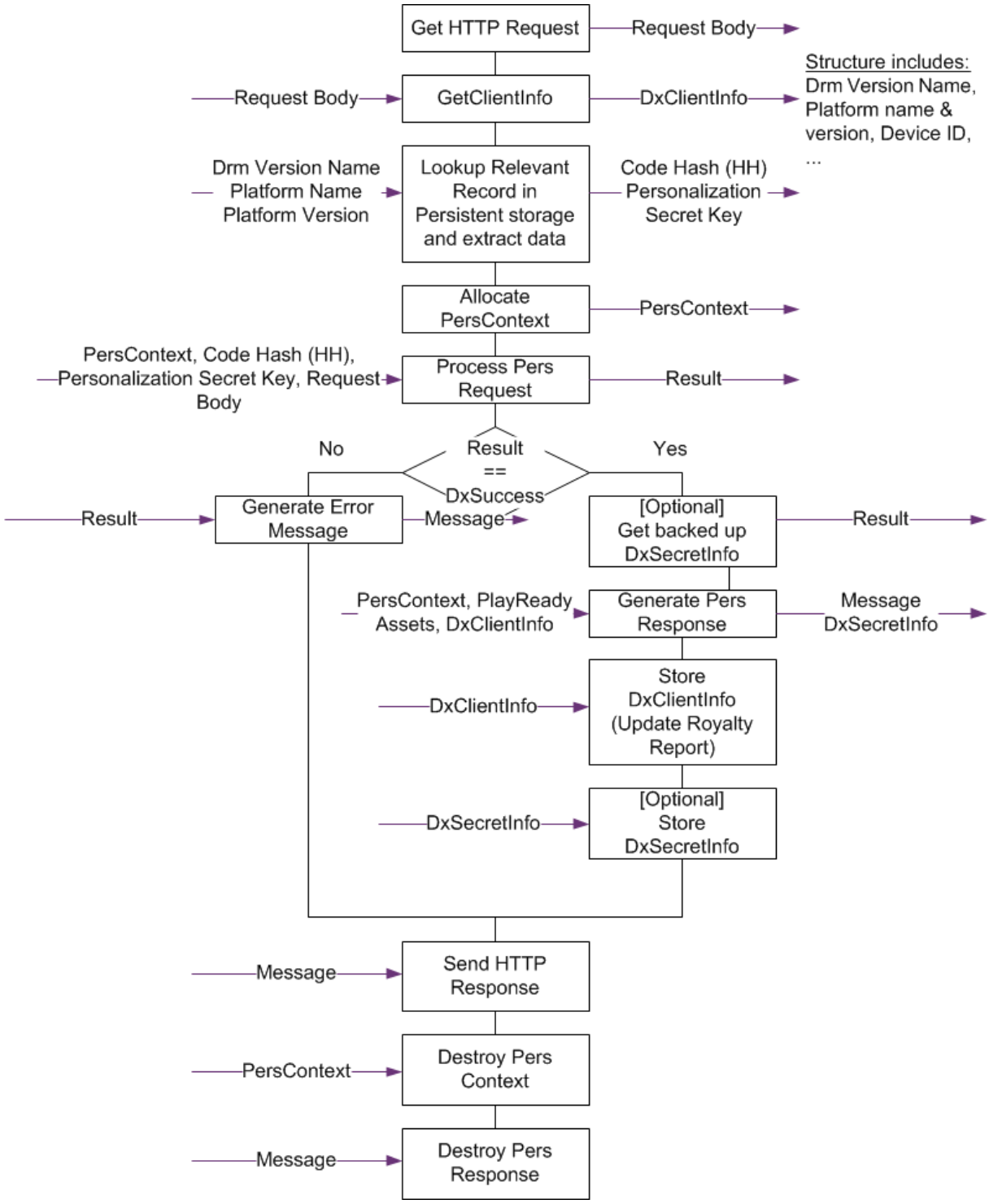


figure 3. Business Logic Flowchart

The following table provides a detailed description of the personalization steps followed by a pseudo-code of the Business logic portion of the personalization procedure:

Operation	Input	Output
`GetClientInfo` Extracts information from the content for preprocessing.	Request Body - The content of the HTTP POST request sent by client.	`DxClientInfo`: • DrmCore Version • Platform Name • Platform Version • Device Id
Lookup Relevant record in Persistent Storage according to some or all of the input variables.	• DrmCore Version • Platform Name • Platform Version (Additional parameters according to Customer requirements.)	• Code Hash • Personalization Secret Key
Allocate PersContext over Heap or Stack (depends on internal development)		`PersContext`
`ProcessPersRequest`	• Request Body • `PersContext` • Code Hash • Personalization Secret Key	Result
`GeneratePersResponse`	• `PersContext` • `PlayReady® Assets` • `ClientInfo`	Response - The data to be sent via HTTP as response to the client. `SecretInfo` - The blob containing the generated device certificates.
`GenerateErrorMessage`	Result - The error code.	Response Body - The data to be sent via HTTP as response to the client.
Update Royalty Report	Device Id	Update the Royalty Report. Insert Device Id to list of devices that performed personalization.
Send HTTP Response	Response Body	
`DestroyPersContext` Free memory allocation of Context internals.	`PersContext`	
`DestroyPersResponse` Free memory allocation of Response allocated by GeneratePersResponse or GenerateErrorMessage.	Response Body	
`DxClientInfo_Destroy`	`ClientInfo`	
`DxSecretInfo_Destroy`	`SecretInfo`	

Business Logic Pseudocode

```
// Receive HTTP request body into RequestBody.
...

EDxStatus status;
DxClientInfo ClientInfo;
DxMessage message;

// Extract Client Info from the Request body.
status = DxServer_GetClientInfo(
     RequestBody,
     RequestBodySize,
     &ClientInfo);
if (status != EDX_SUCCESS)
     goto end;

// Extract data from the ClientInfo
const DxChar* pDrmCoreVersion = DxClientInfo_GetStringItem (
     ClientInfo,
     CLIENT_INFO__DRM_CORE_VERSION);

const DxChar* pPlatformName = DxClientInfo_GetStringItem (
     ClientInfo,
     CLIENT_INFO__PLATFORM_NAME);

const DxChar* pPlatformVersion = DxClientInfo_GetStringItem (
     ClientInfo,
     CLIENT_INFO__PLATFORM_VERSION);

const DxChar* pDeviceID = DxClientInfo_GetStringItem (
     ClientInfo,
     PERS_CLIENT_INFO__DEVICE_ID);

// Access the SPV table with the ClientInfo extracted data.
// Read from SPV table the relevant CodeHash, and PersonalizationSecretKey.
...

// Optional step:
// Read from SPV table the boolean value UpdateRequired.
if (UpdateRequired == TRUE)
{
     // In case Update is required.
     status = EDX_ERROR_VERSION_TOO_OLD;
     goto end;
}
```

```
// Create Personalization Context object
DxServer_PersContext PersContext;

// Create output parameter DxUserData to extract SessionID.
DxUserData outUserData;

// Process Personalization Request
status = DxServer_ProcessPersRequest(
      &PersContext,
      RequestBody,
      StrLen(RequestBody),
      PersonalizationSecretKey,
      CodeHash,
      &outUserData);
if (status != EDX_SUCCESS)
      goto end;

// Customer internal handling of outUserData.m_SessionID
...




// Generate response
status = DxServer_GeneratePersResponse(
      PersContext,
      wmCertTemplate, wmCertTemplateSize,
      wmGroupPrivateKey, wmGroupPrivateKeySize,
      wmFallbackPrivateKey, wmFallbackPrivateKeySize,
      prdyCertTemplate, prdyCertTemplateSize,
      prdyModel, prdyModelSize,
      ClientInfo, outSecretInfo
      &message);

If (status == EDX_SUCCESS)
{
      // Check whether the device identified by pDeviceID (received from above)
      // did perform personalization or not.
      ...

      // Update the Royalty Report accordingly.
      ...

      // Store pDeviceID internally as a device that performed
      // personalization.
      ...}
if (status != EDX_SUCCESS)
```

```
        goto end;

    end:

    if (status != EDX_SUCCESS)
    {
        // failed somewhere along the way
        DxServer_GenerateErrorMessage(status, &message);
    }

    // Send message data via HTTP as response.
    // This may be the error message or the actual generated response.
    ...

    // Clear Message
    DxServer_DestroyPersResponse(&message);

    // Clear PersContext
    DxServer_DestroyPersContext(&PersContext);

    // Clear ClientInfo
    DxClientInfo_Destroy(ClientInfo);

    // Clear SecretInfo
    DxSecretInfo_Destroy(outSecretInfo);
```

Chapter 6: Royalty Report

The Business logic should maintain in its persistent records information on the devices performed for personalization. On a regular basis royalties report must be created as required per license.

The records should include the information collected from the device during the personalization process and specifically the device unique identifier.

According to this unique identifier the Business Logic should verify whether it is a new device or an upgrade of the existing one. For this, the Business Logic should maintain a list of unique identities of all devices that interacted with the server.

The Business Logic shall use the DxClientInfo `PERS_CLIENT_INFO__DEVICE_ID` to get a unique identity for the device requesting personalization.

Chapter 7: Handling Upgrades

There are certain times when a client application upgrade might be required. Those include a security breach, a compatibility issue or a requirement of the service provider.

The enforcer of the upgrade is the Customer Business Logic. The customer changes the CSPV Table to support new version and to disable version due to internal business reasons or security breach.

In cases of security breaches the PlayReady Assets will be changed and client application not upgraded yet will not be able to perform DRM operations with a given SOAP error: `DRM_E_DEVCERT_REVOKED`.

Customer Application Upgrade

The customer may at some point redistribute a newer version of the application. The customer may decide to inform the end-user to upgrade the application. For that matter the following implementation is suggested:

1. The customer should add another row in the CSPV table.
2. Add a new field in the table referred to as `UpdateRequired`. The field value can be Boolean.

DRM Core Version	Platform Name	Platform Version	Code Hash (HH)	Secret Personalization Key	Update Required
`MAIN_DLC_PR_D_1_1_51_0` (Old ver.)	android	7	`de6c3f8ae984d2e1a4b79055f2134e75`	`0404040404040404040404040404040404`	`TRUE`
`MAIN_DLC_PR_D_1_1_52_0` (New ver.)	android	8	`de6c3f8ae984d2e1a4b79055f2134e77`	`0404040404040404040404040404040404`	`FALSE`

3. When a client older version connects and attempts to perform personalization, the relevant information from the CSPV table is extracted including the `UpdateRequired` value, `TRUE`.
4. The User-Management server code should call the `GenerateErrorMessage` with error code `EDX_ERROR_VERSION_TOO_OLD`.

   On the application side the `PerformPersonalization` function will raise an exception. It is up to the Customer application to handle the exception appropriately.

note

The Persistent storage structure including the usage of `UpdateRequired` is a suggestion. The Customer may build the Persistent Storage in other configurations as long as the functionality is not impaired.

Client Application Upgrade Handling

When the User-Management server decides that an upgrade is required, a response is sent back to the client. This is true no matter what component requires the upgrade.

The response is generated in the server by calling `DxServer_GenerateErrorMessage` with error code `EDX_ERROR_VERSION_TOO_OLD` and is sent back to the client.

The error is processed by the Connected Sentinel Player SDK and propagated to the client application.

More about the client application upgrade process can be found in the *Common Integration Guide*.

Chapter 8: Personalization HTTP Protocol

The entities at work in the Personalization HTTP Protocol are the Connected Sentinel Player running within the Customer Application and the User-Management Server.

The protocol used is HTTP/1.1.

Personalization Request

The Personalization HTTP Request is initiated by the Connected Sentinel Player following a call to `PerformPersonalization`.

The HTTP Request specifics are as follows:

```
POST /Personalization HTTP/1.1
Accept-Encoding: identity
Content-Length: 463
Host: xxx.xxx.xxx.xxx
<<REQUEST-DATA>>
```

The `/Personalization` following the POST keyword can be changed and configured differently by the customer simply by sending a different URL string to the `PerformPersonalization` function.

The port number may be inserted into the URL as well.

Personalization Response

All responses of the Personalization Server are as follows:

```
HTTP/1.1 200 OK
Content-Length: 8219
<<RESPONSE-DATA>>
```

Chapter 9: PlayReady Assets

The DRM scheme supported by the Connected Sentinel Player is Microsoft PlayReady®. Hence, the assets that need to be provided via the personalization protocol are PlayReady® certificates and keys.

The way these assets are provided to the User-Management Server is out of the scope of this document.

Assets required by the User-Management Server

The assets required by the User-Management server are:

- Certificate Template
- Model Key
- WM certificate template
- WM keys (Group and fallback)

Device Certificate Chains

A device certificate chain is a digitally-signed binary (or XML document). Part of it is provided by the service provider and the rest is generated by the device at run time.

The certificate chain is persistently stored on each device.

Certificate chains are used by other components of the PlayReady ecosystem, including the device itself or computers that the device interacts with, in order to authenticate and verify the DRM capabilities of the device.

> **note**
>
> For detailed explanation on device certificate chains refer to the PlayReady documentation which is provided by Microsoft®. More precisely refer to document *PlayReady® Device Porting Kit v1.2*, section *Getting Started -> Obtaining Certificates -> About Device Certificate Chains*

All device certificates consist of a chain that is comprised of 4 certificates:

- **Device root CA certificate**: this is the topmost certificate in the chain. It contains information that validates the certificate chain to a known root. This certificate also includes the public key for the device root CA certificate.
- **Device company CA certificate:** this is the authorization certificate for a specific OEM to create model certificates for its device models. This certificate also includes the public key for the device company CA certificate.
- **Model certificate**: this certificate contains information about the device model, such as the model name and manufacturer, as well as the device characteristics, such as the ability to support a secure clock. This certificate also includes the model certificate public key.
- **Device certificate**: this certificate contains the unique device certificate, including a device certificate public key.

Chapter 10: Integration Steps

This section describes the suggested integration process steps:

Step 1: Environment Setup

Step 2: Building your own HTTP Server

Step 3: Business Logic implementation

Step 4: Integrate with Actual Client Application

Following these steps assists in the implementation of the User-Management server.

Step 1: Environment Setup

Setting up the environment includes the following operations:

1. Install the personalization package.
2. Install all required dependencies.
3. Build a simple Visual Studio project linked with the personalization SDK.
4. Set up and activate the provided Python® tools.

Detailed instructions to perform the steps mentioned above can be found in the Personalization Package Setup application.

Step 2: Building your own HTTP Server

The aim of building your own HTTP Server is to:

- Have a functional HTTP/1.1 server.
- Validate proper communication between your server and the Python test client.

> **note**
> It is not recommended to use the Python server provided by Viaccess-Orca as a commercial server.

The current Python client-server implementation expects the HTTP POST request to be sent to `http://xx.xx.xx.xx:yy/Personalization`.

The URL syntax sent by the test client can be easily changed in order to suit your HTTP server needs. How to modify the client URL is described in detail in the Personalization Package Setup. For additional information regarding the HTTP protocol, refer to *Personalization Response* on page 33.

At this point all we want is for the test client to be able to contact the HTTP server successfully. Processing the request is done in the next step.

Step 3: Business Logic implementation

The aim of Business Logic implementation is to:

- Implement the Business Logic
- Incorporate the Business Logic within your own HTTP/1.1 User-Management Server.

It can be helpful to follow these guidelines for the implementation:

1. Create a Visual Studio project for the Business-Logic according to Personalization Package Setup.
2. Implement according to **Business Logic**
3. Add the persistent storage component.

4. Build and incorporate your Persistent Storage solution. It should maintain the Connected Sentinel Player SDK version table (CSPV table).
5. Each entry in the CSPV table should include (at least) the following Connected Sentinel Player SDK specifics:
   - DRMCoreVersion,
   - Platform name,
   - Platform Version,
   - Code Hash,
   - Personalization Secret Key.

   In order to work with the Python test client you need to add the following entry to your CSPV table:
   - **Code Hash**: `de6c3f8ae984d2e1a4b79055f2134e75`
   - **Personalization Secret Key**: `04040404040404040404040404040404`
6. Add the PlayReady Device Assets.

   These assets are composed of:
   - CertTemplate,
   - GroupPrivateKey,
   - FallbackPrivateKey,
   - PlayReady CertTemplate,
   - PlayReady Model.

   They should be accessible by the Business Logic component and provided to the `DxServer_GeneratePersResponse` function.

During development it is recommended to use the Microsoft test certificates that are provided by Viaccess-Orca. The test certificates will not work with a production license server that requires security level of 2000.

> **note**
> The production certificates and keys are highly confidential and should be stored in a secure environment with limited access.

Capturing such a request can be done by using, for example, Wireshark or just modifying the Test Client code to test your Business Logic. However, to perform this action before integration with your actual HTTP User-Management Server you need a valid personalization request sent by the Python client. Capturing such a request can be done by using Wireshark for sniffing or just modifying the client Python code to dump the request to a file.

Incorporate the Business Logic within your own HTTP/1.1 User-Management Server. Once this is done, you may test your implementation with requests sent by the Python test client.

Step 4: Integrate with Actual Client Application

End-to-end solution of the User-Management server and the customer client application using the Connected Sentinel Player SDK.

It can be helpful to follow these guidelines for the implementation:

- Update the CSPV table with the values matching the client application. These should be supplied by Viaccess-Orca.
- On the client application call the `performPersonalization` function with the URL expected by the User-Management Server. It is possible to provide also the application version and `sessionID`.

Appendix

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

Glossary

Term	Description
CA	Certificate Authority
CSPV table	Connected Sentinel Player version table, maintained in persistent storage.
HH	Pers Code Hash is a hash of the client code used to protect from code tampering. Pers Code Hash is a 16-byte binary data provided by Viaccess-Orca for each release
VS	Visual Studio
WM	WMDRM, i.e. Windows Media DRM.

Reference Documentation

- Common Integration Guide
  Reference number: 21816
- Integrating CSP with Android
  Reference number: 21828
- PlayReady Device Porting Kit v.1.2
  Available to the PlayReady technology licensee only.
- Personalization Server SDK API
  Contact your VO Technical Engineer for the appropriate version of this document.