



WIDEVINE

Widevine License SDK

WvPL version 4.5.4

Contents

Overview	7
Dependencies	7
Using boringSSL	8
Working with Certificates	8
One-time Initialization	8
Periodic Updates	9
The Generate License Flow	10
Create Session	12
Service Certificate Request	12
Setup Session	13
Generate License	13
Destroy Session	14
UsageStats API	14
Initializing the Widevine environment	14
GetStatsAsBytes	15
GetStatsAsString	15
Verified Media Path (VMP) API	15
C++ Data Structures	16
Playback Policy (WvPLPlaybackPolicy)	16
Video Resolution Constraints (WvPLVideoResolutionConstraint)	17
Output Protection (WvPLOutputProtection)	19
Content Key or Entitlement Key (WvPLKey)	21
Session Initialization (WvPLSessionInit)	22
License Identification (LicenseIdentification)	24
Session State (WvPLSessionState)	24
Status (WvPLStatus)	26
Device Information (WvPLDeviceInfo)	27
Client Information (WvPLClientInfo)	28
Client Capabilities (WvPLClientCapabilities)	29
Entitled key (WvPLEntitledKey)	30
Pssh Data (WvPLWidevinePsshData)	30
Request Type (WvPLRequestType)	31
C++ Classes	32

Session (WvPLSession)	32
Environment (WvPLEnvironment)	33
PSSH Generator (WvPLPsshDataGenerator)	35
PSSH Information (WvPLPSSHParser)	35
Appendix	36
Device Certificates	36
allow_unknown_device	36
device_certificate_expiration	36
Fetching Certificate Status List	37
Service Certificates	38
Service Certificate installed on the Server	38
Service Certificate installed on the Client	38
Requesting a Service Certificate	38
Initialization Issues	39
License Renewals	40
Offline License	41
Offline License Release	41
Durations	42
Device Security Level	43
Robustness Requirements	43
Platform Verification Status (VMP Status)	44
UsageStats destination	45
Sample Usage Log	45
UsageStats workflow	47
Group License	49
For VOD	49
For Live Streaming	49
Packaging workflow	49
ARM Support	50
Requirements	50
Common Issues	50
Contact Us	50

© 2019 Google, Inc. All Rights Reserved. No express or implied warranties are provided for herein. All specifications are subject to change and any expected future products, features or functionality will be provided on an if and when available basis. Note that the descriptions of Google's patents and other intellectual property herein are intended to provide illustrative, non-exhaustive examples of some of the areas to which the patents and applications are currently believed to pertain, and is not intended for use in a legal proceeding to interpret or limit the scope or meaning of the patents or their claims, or indicate that a Google patent claim(s) is materially required to perform or implement any of the listed items.

Revision History

Version	Date	Description
1.3		Add API for setting the service certificate.
1.4		Add API for getting the PSSH data. Add description of SDK files and dependencies.
1.5		Add to can_renew, can_persist and renewal intervals to WvPLPlaybackPolicy.
1.6		Add rental_duration_seconds to WvPLPlaybackPolicy.
1.7		Add requested_output_ptotection to WvPLKey.
1.8		Add security_level to WvPLOutputProtection. Add video_resolution_constraint to WvPLKey.
1.9		Add WvPLDeviceInfo structure. Add WvPLSession::GetProvisionedDeviceInfo().
1.10		Deprecated (removed) renewal_recovery_duration_seconds.
1.11		Add track_type to WvPLKey Changed WvPLOutputProtection::security_level setting for the case where WvPLOutputProtection::secure_data_path is false. Previously, when secure_data_path was false, the security_level was always set to SW_SECURE_DECODE. Now, security_level is set to SW_SECURE_CRYPTO for all platforms that are not Chrome or if the track type is AUDIO. Otherwise, the default security level is set to SW_SECURE_CRYPTO
1.12		Updated updateDeviceCertificates to updateWithCertificates Marked preprovisioning keys as deprecated Removed PSSH deprecated methods
1.13		Corrected typo in key value in curl command for device certificate status list.
1.14		Updated key value in curl command for device certificate list.
1.15		Deprecated rental duration . Note: This change will be reverted in the near future.
1.16 WvPL 1.12.0		Added UsageStats API. Document version now references WvPL software version.
WvPL 1.12.1		Fix for licensing error due to low device capabilities.
WvPL 1.12.2		Fix for device security level being reported as "UNKNOWN" (Java API only). Added license flow diagrams.

WvPL 1.13.1		Updated Key Control Block (KCB) to return OEMCrypto version supported by the client.
WvPL 1.14.1		Minor bug fix to usage stats.
WvPL 1.14.1	2/15/2017	Updated usage reporting URL and sample usage in Appendix
WvPL 1.15.0	3/29/2017	Add PSSH Generator class.
WvPL 2.0.1	5/12/2017	SDK now requires C++ 11 (CentOS 7 64-bit, Ubuntu 14.04 / 16.04 64-bit) Replaced OpenSSL with boringSSL . Added Verified Media Path (VMP) API . Expanded Service Certificates section.
WvPL 3.0.0	8/15/2017	Add support for group license .
WvPL 3.1.0	8/25/2017	Add documentation for SessionInit .
WvPL 3.2.0	9/14/2017	Add GetParametersFromPSSH API. Add License Renewals .
WvPL 3.3.0	10/16/2017	Add ClientInfo structure and Session::GetClientInfo() API.
WvPL 3.4.2	10/25/2017	Add analog_output and CGMS to WvPLOutputProtection . Update Key Control Block (KCB) to return OEMCrypto version of the client for V13 or greater.
	11/20/2017	Added ARM requirements for Raspberry Pi.
WvPL 4.1.1	01/31/2018	Protobuf (which is a dependency for this SDK) version updated to 3.5.0 from 2.6.1. Build process change (MOE). Added device certificate serial number and device certificate service id to WvPLDeviceInfo. A call to GetProvisionedDeviceInfo() would return these additional fields as well, if available.
WvPL 4.1.2	02/12/2018	Fixed "WvPLEnvironment.generateErrorResponse returns empty string" Fixed "Return null when WvPLSession::generateErrorResponse() fails, instead of empty bytearray". Added "Replace state-wrapping POJOs for WvPL objects, with one that returns the String representing the state". Added "SessionInit.oem_crypto_api_version" to allow for setting in license request. This is available only in native SDK API and not in WvPL SDK API.
WvPL 4.3.2	4/3/2018	Add WvPLWidevinePsshData and Session::GetPsshData API. SessionTester updates to exercise "native SDK" API better for a valid service certificate and a valid device certificate status list. Fixed license counter status bug to avoid overriding GenerateLicense status in WvPL.
WvPL 4.3.4	5/15/2018	Added GetPsshData API to WvPLSession, which returns WvPLWidevinePsshData. Fixed GetSessionUsage API in Session.

WvPL 4.4.1	7/1/2018	Added support for Entitlement licensing. WvPLKeyType allows WvPLKey API users to set the KeyType.
WvPL 4.4.2	8/21/2018	Fixes for (1) allowing 'devices with revoked systemIDs' to license in WvDRM SDK only (2) javadocs for WvPLKeyType (3) WvPLWidevinePsshData includes 'content id'. Add Appendix section on License Release.
WvPL 4.5.0	10/01/2018	A content provider using the WvPL License SDK must load a content-provider-specific Service Certificate before processing license requests and generating licenses. The Content Provider must have at least one valid service certificate. Protos renaming. Use libexported_wvpl_license_server_sdk_proto-speed.jar and libexported_wvdrm_license_server_sdk_proto-speed.jar. Fix for counter data API not working with "flushData" setting. The data flush was not being performed.
WvPL 4.5.1	11/15/2018	Add support for HDCP 2.3. Add WvPLOutputProtection::disable_digital_output().
WvPL 4.5.2	01/15/2019	Fix for setting of required_output_protection and requested_output_protection. Expose WvPLSessionState to WvPL SDK for release and renewal request.
WvPL 4.5.3	02/15/2019	Add new function getRequestType to get a wrapper of MessageType, LicenseType and LicenseRequestType. Fix to retrieve WvPLSessionState from WvPLSession. This functionality will work only upon successful license generation. Fix to retrieve LicenseIdentification from WvPLSessionState.
WvPL 4.5.4	03/30/2019	The service certificate contains a new type field. <i>A new service certificate must be re-issued to work with 4.5.4 (or later).</i> Added or Updated ClientCapabilities , EntitledKey , LicenseIdentification , SessionState . Add new function getKeys to get WvPLKey. GetClientInfo() returns client identification information for license requests from any Widevine client.

Overview

The Widevine Partner License SDK is a set of libraries to allow for the integration and deployment of a stand-alone license fulfillment mechanism for modular DRM requests, from a Widevine client platform.

The license SDK will enable the following:

- Ability to fulfill license requests
 - Generate a valid license response with basic parameters
 - license duration
 - playback duration
 - toggle HDCP
- Ability to validate a legitimate device
- [Generate aggregated statistics](#)
 - Widevine requires periodic submissions for device and product quality improvements.

This SDK is shipped with two libraries:

- libwvpl.so
 - *libwvpl.so* is the library application developers will use directly. It implements the APIs in this document.
- libwvdrm.so
 - *libwvdrm.so* is a lower level API used by *libwvpl.so*.

Note: This document describes the C++ APIs. Equivalent **Java APIs** are described in the SDK download tarball under javadocs.

Dependencies

- C++ 11
 - Ubuntu 14.04 64-bit
 - CentOS 7 64-bit
- libprotobuf.so.9 => /usr/local/lib/libprotobuf.so.9
 - Requires libprotobuf version 3.5.0
 - If this varies from the version installed in /usr/local/lib, will need to override the version by adding the path to the 3.5.0 version in LD_LIBRARY_PATH similar to boringSSL below.
 - Most underlying data structures used in *libwvdrm.so* are google protocol buffers. See: <https://developers.google.com/protocol-buffers/>
- libcrypto.so.1.0.0 => /lib/x86_64-linux-gnu/libcrypto.so

Using boringSSL

Download and install boringSSL from [github](#).

After unpacking the downloaded zip file, run:

```
cd boringssl-master
sed -i s/-Werror//g CMakeLists.txt
mkdir build
cd build
cmake -DBUILD_SHARED_LIBS=1 -DCMAKE_CXX_FLAGS="-lrt" ..
make
```

To run the Java API, set the LD_LIBRARY_PATH as:

```
export LD_LIBRARY_PATH=<the location of your
boringSSL>/boringssl-master/build/crypto:<location of license service SDK
libs>:$LD_LIBRARY_PATH
```

Working with Certificates

Widevine-enabled devices contain device certificates used to individualize the licenses. The license server needs to authenticate the device certificate included in the license request. For this reason, the license server must hold a set of intermediate certificates and the root certificate that are used to authenticate the device's certificate. The intermediate and root certificates are retrieved from a Google server and should be periodically updated to pick up newly supported devices. The root certificate remains unchanged.

Starting from version 4.5.4, a new service certificate must be issued. Old service certificates used with earlier versions of the SDK are no longer compatible.

One-time Initialization

Generate a Widevine environment (using the `WvPLEnvironment` class) object one time on server or service startup. The *WvPLEnvironment* object is used to create widevine sessions. `WvPLEnvironment` is used to hold data and spans the lifetime of a license session, therefore it should be retained by the caller. Sessions will not get generated if the initial calls to *UpdateWithCertificates()* or *SetServiceCertificate()* fail.


```

map<string, string> config_values;
    // Configure the values.
    SomeFunctionToSetupConfiguration(&values);
WvPLEnvironment* wv_env = new WvPLEnvironment(config);
WvPLStatus status = wv_env->SetServiceCertificate(...);
std::string cert_status_list;
    // Fetch the intermediate certificates.
    SomeFunctionToFetchTheCertificates(&cert_status_list);
status = wv_env->UpdateWithCertificates(cert_status_list);

```

Periodic Updates

In order to stay up-to-date with new devices, it is recommended to fetch new certificates on a regular basis. The update interval is recommended to be once a day. This means new devices will be supported within 24 hours after added to Widevine's database of supported devices. If `UpdateWithCertificates()` fails, the existing certificates are still valid, but are subject to [expiration](#). The expiration is controlled by the content provider and configured as a parameter when creating `wv_environment`. For details, see [Appendix: Device Certificates](#).

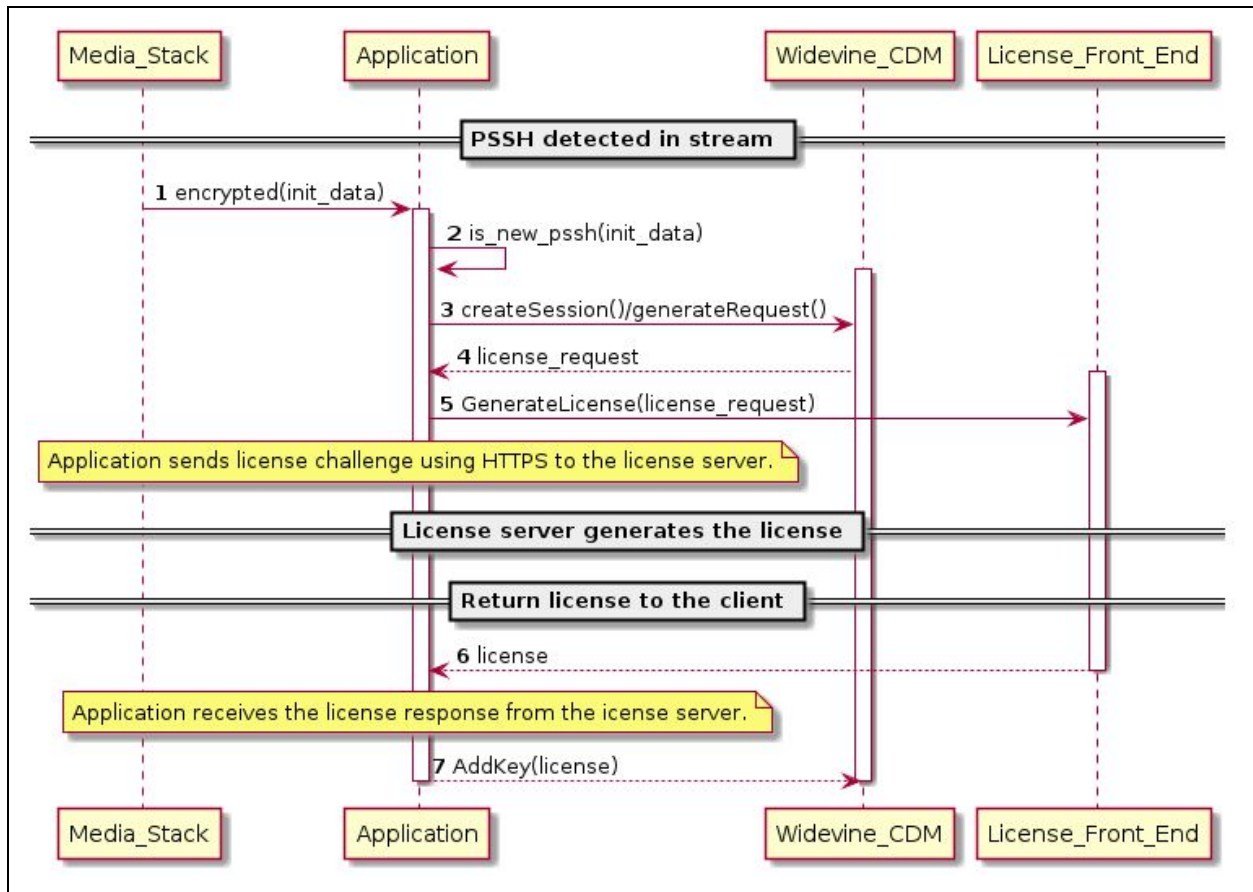
```

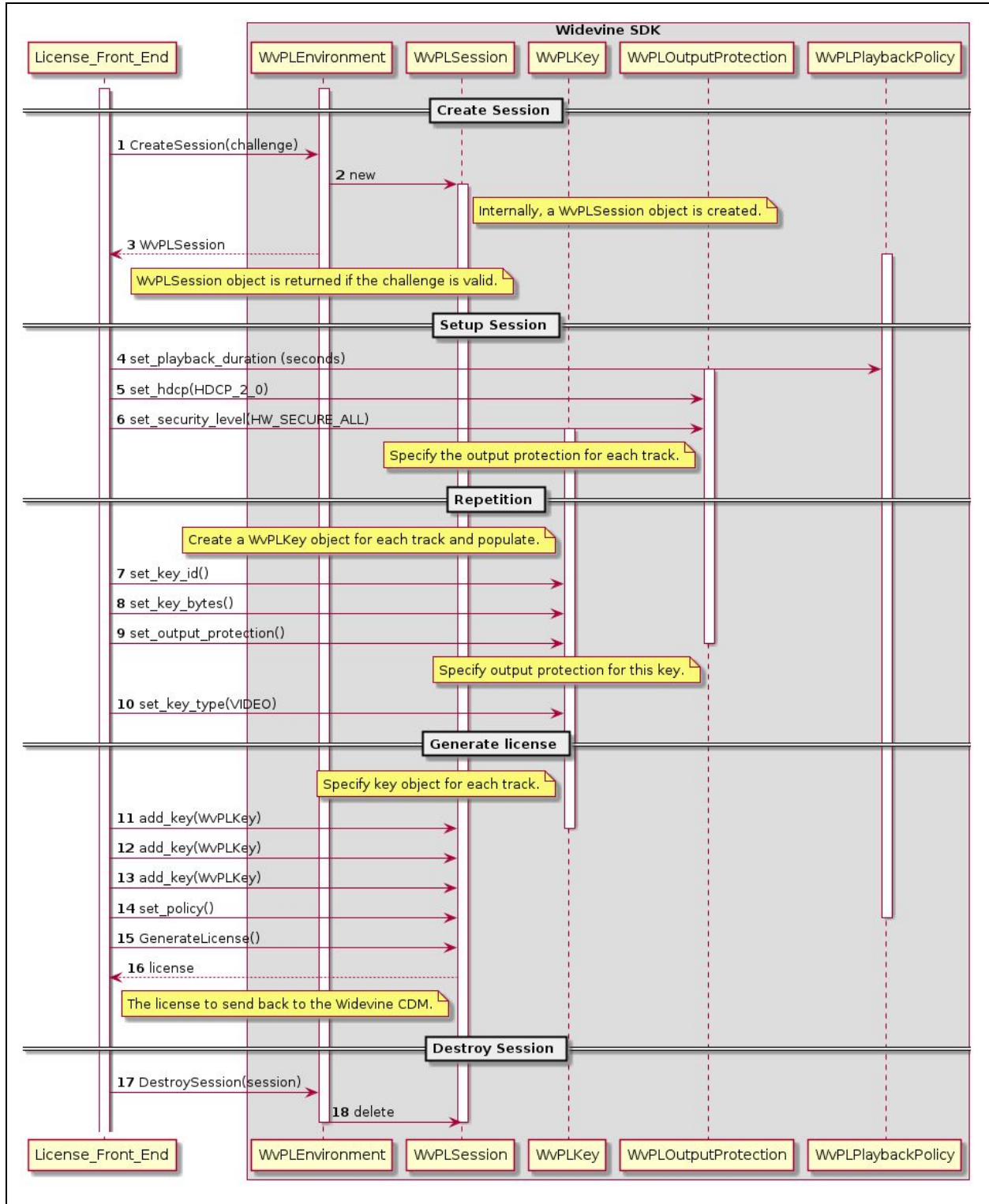
status = wv_env->UpdateWithCertificates(cert_status_list,);

```

The Generate License Flow

The sequence diagrams below shows the interactions between the application, the content provider's license server, referred to as the License_Front_End, and Widevine SDK.





Create Session

Once a Widevine environment object is successfully initialized, generate a Widevine session object for each license request. `CreateSession()` parses the request and validates the request by verifying the device signature. If successful, a session object is created and OK is returned.

```
WvPLSession* session;
status = wv_env->CreateSession(request_from_cdm, &session);
if (!status.ok()) {
    string error_license;
    if (WvPLEnvironment::GenerateErrorResponse(status, &error_license) {
        // Send "error_license" to the client.
    } else {
        // Handle error.
    }
}
```

Service Certificate Request

`WvPLEnvironment::CreateSession()` may fail with error 139 (SERVICE_CERTIFICATE_REQUEST_MESSAGE) indicating the client is requesting a service certificate (see [Appendix: Service Certificates](#)). In this case, `WvPLEnvironment::GenerateErrorResponse()` will handle this error by generating a license response with the service certificate.

Setup Session

Once a Widevine session object is successfully created, setup the session object with the policy and keys. *AddKey()* is called once for each key used to encrypt the asset. For example, if HD is not authorized, then call *AddKey()* twice, once for audio and once for SD. Once all keys are added, a license can be generated.

```
WvPLPlaybackPolicy policy;
// Allow 7 days to start playback
policy.set_license_duration_seconds(604800);

// Allow 24 hours once playback starts.
policy.set_playback_duration_seconds(86400);
status = session->set_policy(policy);

// Require HDCP 2.0 for HD key.
WvPLOutputProtection hd_output_prot;
hd_output_prot.set_hdcp(HDCP_2);
hd_output_prot.set_security_level(HW_SECURE_ALL);

WvPLKey audio_key;
WvPLKey sd_key;
WvPLKey hd_key;

hd_key.set_output_protection(hd_output_protection);

status = session->AddKey(audio_key);
status = session->AddKey(sd_key);
Status = session->AddKey(hd_key);
```

Generate License

Once a Widevine session object is successfully created and configured. Use *GenerateLicense()* to construct a license. If successful, OK is returned.

```
std::string output_license;
status = session->GenerateLicense(&output_license);
```

Destroy Session

Once a license is generated, the session should get destroyed.

```
wv_env->DestroySession(&session);
```

UsageStats API

The purpose of this API is to send aggregated license statistics to Google. This will allow us to identify errors and assist in resolving end-user playback problems. The aggregate statistics are collected within the SDK and the API will retrieve the data.

Please review the [UsageStats workflow](#) in the Appendix.

Initializing the Widevine environment

The Widevine environment constructor requires 3 configuration parameters to support UsageStats:

1. provider
2. provider_iv
3. provider_key

Note: kProvider, kProviderIv and kProviderKey are already defined in wvpl_environment.h.

```
const char kWidevineTestProvider[] = "widevine_test";
const char kWidevineTestProviderIv[] = "d58ce954203b7c9a9a9d467f59839249";
const char kWidevineTestProviderKey[] =
    "1ae8ccd0e7985cc0b6203a55855a1034afc252980e970ca90e5202689f947ab9";

std::map<std::string, std::string> config;
config[kProvider] = kWidevineTestProvider;
config[kProviderIv] = kWidevineTestProviderIv;
config[kProviderKey] = kWidevineTestProviderKey;
WvPLEnvironment wvpl_env(config);
```

GetStatsAsBytes

After successful initialization of the Widevine environment object, this call is designed to generate data to be sent to the Google service.

This call returns the signed usage stats data in serialized format and should be sent as a POST data payload. The aggregated stats data is optionally flushed via the `flush_data` parameter. If the data is not flushed, the aggregate stats will continue to accumulate.

Note: `flush_data` will purge the data for both [GetStatsAsBytes](#) and [GetStatsAsString](#).

Please review the [usage stats workflow](#) in the Appendix.

```
std::string signed_license_stats;  
WvPLStatus status = wvpl_env.GetStatsAsBytes(true, &signed_license_stats);
```

GetStatsAsString

This call is designed to be only used to view the data that is being collected in a string format. *Do not submit this data to the Google service.*

As with [GetStatsAsBytes](#), this call has an optional `flush_data` parameter. If the data is not flushed, the aggregate stats will continue to accumulate.

Note: `flush_data` will purge the data for both [GetStatsAsBytes](#) and [GetStatsAsString](#).

Please review the [usage stats workflow](#) in the Appendix.

```
std::string actual_license_stats;  
WvPLStatus status = wvpl_env.GetStatsAsString(true, &actual_license_stats);
```

Verified Media Path (VMP) API

The VMP is a means of establishing trust with the host that receives the decrypted media from the Widevine Client Decryption Module (CDM).

This API retrieves the [VMP Status](#) for a generated license.

```
PlatformVerificationStatus VerifyPlatform()
```

C++ Data Structures

Playback Policy (WvPLPlaybackPolicy)

Indicates playback attributes for the license. The policy attributes apply to the entire license and affects all keys. Most providers only need to set the first two values in the table below.

Name	Type	Description
license_duration_seconds_	uint32	The license window. Once a license is granted, the number of seconds to use this license. Default = 0, indicating unlimited. Recommend using <i>set_license_duration_seconds(value)</i>
playback_duration_seconds_	uint32	The viewing window. Once playback starts, the number of seconds to use the license. Default = 0, indicating unlimited but not to exceed the license_duration . Recommend using <i>set_playback_duration_seconds(value)</i>
rental_duration_seconds_	uint32	The rental window. Once a license is granted, the number of seconds to use the license. Once playback starts, the viewing window (playback duration) is activated. Recommend using <i>set_rental_duration_seconds(value)</i>
can_play_	bool	Playback of the content is allowed. Default = true Recommend using <i>set_can_play(flag)</i>
can_persist_	bool	License may be persisted for offline use. Default = false Recommend using <i>set_can_persist(flag)</i>
can_renew_	bool	Renewal of the license is allowed. Default = false Recommend using <i>set_can_renew(flag)</i>

renewal_interval_seconds_	int64	How many seconds after start time before renewal is first attempted. Default = 0 Recommend using <i>set_renewal_interval_seconds(secs)</i>
renewal_retry_interval_seconds_	int64	Delay in seconds between renewals, in case of failure. Default = 0 Recommend using <i>set_renewal_retry_interval_seconds(secs)</i>
renewal_recovery_duration_seconds_	int64	Window of time in which playback is allowed to continue while renewal is attempted. Default = 0 Recommend using <i>set_renewal_recovery_duration_seconds(secs)</i>
renewal_url_	string	Renewal requests are sent to this URL. Recommend using <i>set_renewal_url(url)</i>

Video Resolution Constraints (WvPLVideoResolutionConstraint)

Specify output protection based on minimum and maximum video resolutions settings. If not specified, the output protection for the key is used for all resolutions.

Name	Type	Description
min_resolution_pixels_	uint32	Minimum video resolutions in the range (height x width). Recd using <i>set_ommenmin_resolution_pixels(value)</i>
max_resolution_pixels_	uint32	Maximum video resolutions in the range (height x width). Recommend using <i>set_max_resolution_pixels(value)</i>
hdcp_	enum	Available options: HDCP_NONE HDCP_V1 HDCP_V2 HDCP_V2_1 HDCP_V2_2 HDCP_V2_3

		<p>HDCP_NO_DIGITAL_OUTPUT</p> <p>Default = Use the HDCP setting from WvPLOutputProtection.</p> <p>Recommend using <i>set_hdcp(value)</i>.</p>
--	--	---

Output Protection (WvPLOutputProtection)

Indicates the output protection settings. Output protection is specified per key. For example, an SD or Audio key may not require HDCP, however an HD key may require HDCP.

Name	Type	Description
hdcp_	enum	<p>Available options:</p> <p>HDCP_NONE HDCP_V1 HDCP_V2 HDCP_V2_1 HDCP_V2_2 HDCP_V2_3 HDCP_NO_DIGITAL_OUTPUT</p> <p>Default = HDCP_NONE</p> <p>Recommend using <i>set_hdcp(value)</i>.</p>
hdcp_srm_rule_	enum	<p>Describes the System Renewability message (SRM) settings.</p> <p>Available options:</p> <p>HDCP_SRM_RULE_NONE CURRENT_SRM</p> <p>Default = HDCP_SRM_RULE_NONE</p> <p>Recommend using <i>set_hdcp_srm_rule(value)</i>.</p>
secure_data_path_	bool	<p>If true, requires a secure data path on the device. This is the same as setting the security_level to HW_SECURE_ALL.</p> <p>Default = false</p> <p>Note: Setting the security_level explicitly will override this setting</p> <p>In the case where secure_data_path is false:</p> <ul style="list-style-type: none">• For Chrome (except AUDIO), security_level=SW_SECURE_DECODE.• For all other platforms and AUDIO, security_level=SW_SECURE_CRYPT. <p>Recommend using <i>set_secure_data_path(flag)</i>.</p>
security_level_	enum	<p>Available options (in increasing order of security):</p> <p>SW_SECURE_CRYPT SW_SECURE_DECODE</p>

		HW_SECURE_CRYPT0 HW_SECURE_DECODE HW_SECURE_ALL If set, this will supersede “secure_data_path”. Default = Use secure_data_path settings Recommend using <i>set_security_level(level)</i> . See here for more information.
disable_analog_output_	bool	Indicates analog output is not allowed. Default = false. Recommend using <i>set_disable_analog_output(flag)</i> .
disable_digital_output_	bool	Indicates digital output is not allowed. Default = false. Recommend using <i>set_disable_digital_output(flag)</i> .
cgms_	enum	Copy Generation Management System - Analog Available options: CGMS_NONE - Copy No more. COPY_FREE - Unlimited copies. COPY_ONCE - One copy. COPY_NEVER - No copies may be made. Default = CGMS_NONE Recommend using <i>set_cgms_(value)</i>

Content Key or Entitlement Key (WvPLKey)

The content key or entitlement key used to encrypt/decrypt the content including the output protection to apply.

Name	Type	Description
key_id_	std::string	REQUIRED Unique identifier for this key. Recommend using <i>set_key_id(id)</i> .
key_bytes_	std::string	REQUIRED Key used for encrypt/decrypt. Must be 16 bytes for content key and 32 bytes for entitlement key. Recommend using <i>set_key_bytes(key)</i> .
output_protection_	WvPLOutputProtection	Required output protection for this key. Recommend using <i>set_output_protection()</i> .
requested_output_protection_	WvPLOutputProtection	Requested output protection for this key. Recommend using <i>set_requested_output_protection()</i> .
video_resolution_constraint_	std::vector	std::vector of WvPLVideoResolutionConstraint. Use the AddVideoResolutionConstraint() to add new constraint objects to the vector. Recommend using <i>AddVideoResolutionConstraint(const WvPLVideoResolutionConstraint& constraint)</i> .
track_type_	enum	Track types: AUDIO VIDEO_SD VIDEO_HD Default = TRACK_TYPE_UNSPECIFIED Recommend using <i>set_track_type(track_type)</i> .
key_type_	enum	Key types: UNKNOWN_KEY CONTENT ENTITLEMENT

		<p>Default = CONTENT</p> <p>Recommend using <i>set_key_type(key_type)</i>.</p>
--	--	--

Session Initialization (WvPLSessionInit)

Optional data used for generating a license.

Name	Type	Description
session_id_	std::string	<p>Identifier for this session. This value is returned in the license and will exist in subsequent license request used during renewals.</p> <p>Recommend using <i>set_session_id(id)</i>;</p>
purchase_id_	std::string	<p>Identifier for this session. This value is returned in the license and will exist in subsequent license request used during renewals.</p> <p>Recommend using <i>set_purchase_id(id)</i>;</p>
master_signing_key_	std::string	<p>128-bit key that can be optionally set if RENEWAL license requests are to be issued by the client app. If specified, this License SDK will derive a “Renewal Signing Key”, which is returned in the generated license. This “Renewal Signing Key” will be used by the Widevine Client (CDM) when sending RENEWAL license requests.</p>
signing_key_	std::string	<p>512-bit key to be split into two HMAC-SHA256 keys, for server and client, if RENEWAL license requests are to be issued by the client app. If this field is specified, the License SDK will use this key as is for “Renewal Signing Key”, which is returned in the generated license. This “Renewal Signing Key” will be used by the Widevine Client (CDM) when sending RENEWAL license requests.</p>
license_start_time_	int64	<p>Time of the request in seconds (UTC). If specified, this value is used to indicate license start time. If not specified, the license server will use the current time.</p> <p>Recommend using <i>set_license_start_time(start_time)</i>;</p>
provider_client_token_	std::string	<p>Token that will persist on the client. The client token is sent in all subsequent license request.</p>

		Recommend using <i>set_provider_client_token(token)</i> ;
override_provider_client_token	bool	<p>If true and the license request includes a previous token, the token specified by <i>provider_client_token</i> above will replace it, otherwise the existing token is not replaced.</p> <p>Default = false</p> <p>Recommend using <i>set_override_provider_client_token(flag)</i>;</p>
provider_session_token_	std::string	A token that is set by the content provider and is used by the client to index the session identifier.
exclude_group_key	bool	<p>Set true if group master license should not be included in the license.</p> <p>Default = false</p> <p>Recommend using <i>set_exclude_group_key(flag)</i>;</p>

License Identification (LicenseIdentification)

Optional data used for generating a license.

Name	Type	Description
request_id_	std::string	License request id.
session_id_	std::string	Session identifier for the license request.
provider_session_token_	std::string	A token that is set by the content provider and is used by the client to index the session identifier.
purchase_id_	std::string	Identifier for this session. This value is returned in the license and will exist in subsequent license request used during renewals.
license_type_	LicenseType	An enumeration of license type, enum LicenseType { STREAMING = 1, OFFLINE = 2, };
version_	uint32	License request version.

Session State (WvPLSessionState)

Optional data used for generating a license.

Name	Type	Description
license_id_	LicenseIdentification	License information propagated from the license request to the license. Recommend using <code>set_license_id(license_id);</code>
signing_key_	std::string	512-bit key to be split into two HMAC-SHA256 keys, for server and client, if RENEWAL license requests are to be issued by the client app. If this field is specified, the License SDK will use this key as is for “Renewal Signing Key”, which is returned in the generated license. This “Renewal Signing Key” will be used by the Widevine Client (CDM) when sending RENEWAL license requests.

		Recommend using <code>set_signing_key(signing_key);</code>
provider_client_token_	std::string	Token that will persist on the client. The client token is sent in all subsequent license requests. Recommend using <code>set_provider_client_token(token);</code>
keybox_system_id_	uint32	Widevine system ID for the device that is in the Widevine keybox. Recommend using <code>set_keybox_system_id(keybox_system_id);</code>
license_counter_	uint32	License counter associated with provider_client_token_.

Status (WvPLStatus)

The return status used in many of the APIs.

```
WvPLStatus status = wv_env.CreateSession(...);  
if (status.error_code() != OK) {  
    LOG << status.error_message();  
}
```

Name	Type	Description
status_code_	StatusCode	Numeric status value. Recommend using <i>error_code()</i> ..
msg_	std::string	Human readable string with error details. Recommend using <i>error_message()</i> .

Device Information (WvPLDeviceInfo)

Information for a provisioned device making the license request.

Name	Type	Description
system_id_	uint32	Widevine system ID for the device.
soc_	std::string	Name of system-on-a-chip.
manufacturer_	std::string	Name of manufacturer.
model_	std::string	Manufacturer's model name.
device_type_	std::string	Type of device (Phone, Tablet, TV, etc).
security_level_	enum	Widevine-defined security level. Values: LEVEL_UNSPECIFIED LEVEL_1 LEVEL_2 LEVEL_3
test_device_	bool	True if the certificate corresponds to a test (non-production) device.
serial_number_	std::string	128-bit globally unique serial number of the certificate.
service_id_	std::string	Service Identifier for the provider who owns this certificate.

Client Information (WvPLClientInfo)

Information for a client making the license request.

Name	Type	Description
max_hdcp_version_	enum	HDCEP version. Values: HDCEP_NONE HDCEP_V1 HDCEP_V2 HDCEP_V2_1 HDCEP_V2_2 HDCEP_V2_3 HDCEP_NO_DIGITAL_OUTPUT
oem_crypto_api_version_	uint32	OEM Crypto API version.
cdm_version_	string	CDM Client version.
provider_client_token_	string	Client token generated by the content provider.
names_value_	map<std::string, std::string>	Map of other fields specified as name/value pairs.
analog_output_capabilities_	AnalogOutputCapabilities	Analog capabilities.
can_disable_analog_output_	bool	'true' if the device can disable its analog output, else 'false'.
client_capabilities_	WvPLClientCapabilities	Client capabilities.

Client Capabilities (WvPLClientCapabilities)

Capabilities of the client making the license request.

Name	Type	Description
client_token_	bool	Client token generated by the content provider.
session_token_	bool	Token for the client session.
video_resolution_constraints	bool	'true', if video resolution constraints are supported on the device, else 'false'.
max_hdcp_version_	enum	HDCEP version. Values: HDCEP_NONE HDCEP_V1 HDCEP_V2 HDCEP_V2_1 HDCEP_V2_2 HDCEP_V2_3 HDCEP_NO_DIGITAL_OUTPUT
oem_crypto_api_version_	uint32	OEM Crypto API version.
anti_rollback_usage_table_	bool	'true', if the client supports anti-rollback of the usage table, else 'false'.
srm_version_	uint32	Version of the System Renewability Message (SRM).
can_update_srm_	bool	'true', if the device can update SRM data, else 'false'.
supported_certificate_type_	std::list<CertificateKeyType>	A list of certificate key types. CertificateKeyType is an enum with possible values of RSA_2048 RSA_3072
analog_output_capabilities_	AnalogOutputCapabilities	Analog capabilities.
can_disable_analog_output_	bool	'true' if the device can disable its analog output, else 'false'.

Entitled key (WvPLEntitledKey)

Information about the entitled or wrapped key in the Widevine PSSH.

Name	Type	Description
entitlement_key_id_	std::string	Entitlement or wrapping key identifier.
key_id_	std::string	Entitled or wrapped key identifier.
key_bytes_	std::string	Entitled or wrapped key bytes.
entitlement_key_iv_	std::string	IV used for entitlement or wrapping key.
entitlement_key_size_bytes_	uint32	Size of the entitlement or wrapping key in bytes.

Pssh Data (WvPLWidevinePsshData)

Information about the Widevine Pssh in the license request.

Name	Type	Description
key_ids_	std::list<std::string>	A list of key identifiers, for entitlement keys or content keys.
content_id_	std::string	The content identifier.
entitled_keys_	std::list< WvPLEntitledKey >	A list of wrapped (entitled) keys.
video_feature_	std::string	Video feature identifier.

Request Type (WvPLRequestType)

Information about the type of request sent to this SDK.

Name	Type	Description
message_type_	MessageType	An enumeration of message type, enum MessageType { UNKNOWN = 0, LICENSE_REQUEST = 1, SERVICE_CERTIFICATE_REQUEST = 4 };
license_type_	LicenseType	An enumeration of license type, enum LicenseType { STREAMING = 1, OFFLINE = 2, };
license_request_type_	LicenseRequestType	An enumeration of license request type, enum LicenseRequestType { REQUEST_TYPE_UNSPECIFIED = 0, NEW = 1, RENEWAL = 2, RELEASE = 3, };

C++ Classes

Session (WvPLSession)

Represents the session and is used for generating the license. The public methods are:

- *WvPLStatus AddKey (const WvPLKey& key)*
Called once for each key in the license. Subsequent calls with the same key_id will return an ALREADY_EXIST error.
- *WvPLStatus SetPolicy (const WvPLPlaybackPolicy& policy)*
Called once for specifying the playback policy. Subsequent calls will overwrite the previous policy.
- *WvPLStatus SetSessionInit (const WvPLSessionInit& session_init)*
Called once for specifying optional session initialization data.
- *WvPLStatus GenerateLicense (std::string* license, string)*
Generates the actual license.
- *WvPLStatus GetContentId (std::string* content_id)*
Returns the content Id from the PSSH header.
- *bool GetProvisionedDeviceInfo (WvPLDeviceInfo* device_info)*
Returns information about the provisioned device making the license request.
- *bool GetClientInfo (WvPLClientInfo* client_info)*
Returns information about the client making the license request.
- *WvPLStatus GetPsshData(WvPLWidevinePsshData* wvpl_widevine_pssh_data)*
Returns Widevine PSSH data.
- *static std::string GetVersionString ()*
Returns a version string in the form: <major_version>.<minor_version>.<release>
- *WvPLRequestType GetRequestType()*
Returns information about the type of request sent to this SDK.
 - *MessageType message_type()*
Returns the message type as defined by the MessageType enumeration.
 - *LicenseType license_type()*
Returns the license type as defined by the LicenseType enumeration.
 - *LicenseRequestType license_request_type()*

Returns the license request type as defined by the LicenseRequestType enumeration.

Environment (WvPLEnvironment)

Used to generate session objects and manage global data. The public methods are:

- *WvPLEnvironment(const map<std::string, std::string>& config)*

Construct a Widevine environment with the following name/value configuration:

Name	Description
device_certificate_expiration	This value indicates how many seconds before the device certificate status list expires. , The server must call <i>UpdateWithCertificates()</i> before the certificate status list expires. Default = 7 days.
allow_unknown_devices	A device certificate status must exist for a device making the license request. If one is not found, an error is returned during the <i>GenerateLicense()</i> call unless “allow_unknown_device” is set to true. Default = false.

- *WvPLStatus Initialize ()*
Called once. Subsequent calls are ignored and OK is returned.
- *WvPLStatus UpdateWithCertificates (const std::string& cert_status_list)*
Called regularly up update device status list.
- *WvPLStatus SetServiceCertificate (const string& certificate, const string& private_key, const string& private_key_passphrase)* caller must specify the service certificate and private key in binary format.
Called once to set the service certificate.
- *WvPLStatus CreateSession (const std::string& license_from_cdm, WvSession** session)*
Used for creating session objects. If successful, a new session object is created and returned to the caller.
- *WvPLStatus DestroySession (WvPLSession** session)*
Used for destroying sessions objects.

- *bool GenerateErrorResponse (WvPLStatus status_from_create, string* error_license)*
Called to handle errors from CreateSession(). In some cases, CreateSession() may fail because the client is requesting a service certificate or has presented an invalid device certificate. This function is used to generate the appropriate response for the client.
- *WvPLStatus SetPreProvisioningKeys (const map<uint32, string>& keys)*
Set the pre-provisioning keys. This API is provided for situations where the pre-provisioning keys are used instead of the device certificates. **This only required to support legacy devices and is considered deprecated.**
- *WvPLStatus GetStatsAsBytes (bool flush_data, std::string* signed_license_stats)*
Retrieves the collected aggregated license stats as bytes.
The *license_stats* is signed by the provider's IV and key. This data is to be sent to Google via a HTTP POST.
The *flush_data* parameter allows the aggregated license stats to be flushed.
- *WvPLStatus GetStatsAsString (bool flush_data, std::string* license_stats)*
Retrieves the collected aggregated license stats as a string.
This string is in human-readable form and is the license data.
The *flush_data* parameter allows the aggregated license stats to be flushed.
- *WvPLStatus AddGroupMasterKey (string master_key_id, master_key_bytes);*
Add a group master key for use in generating group embedded licenses during PSSH generation and generating group master license. This master group key is used to derive group keys. Multiple master keys are supported to allow the group master key to be periodically rotated or to assign a different master key to each group.
- *WvPLStatus CreatePSSHGenerator (string group_master_key_id, WvPLPSSHGenerator** pssh_generator);*
Creates a PSSH generator object. The *group_master_key_id* must match a *group_master_key_id* previously specified using *AddGroupMasterKey()*. The group master key associated with the specified *group_master_key_id* is use by the newly constructed *WvPLPSSHGenerator* object when generating PSSHs.
- *WvPLStatus CreatePsshParser(const std::string& pssh_bytes, WvPLPSSHParser** parser);*
Creates a *WvPLPSSHParser* from *pssh_bytes*. The resulting parser can be used to extract information from the Widevine PSSH.

PSSH Generator (WvPLPsshDataGenerator)

Used during the packaging process to generate Widevine DRM specific PSSH data. The public methods are:

- *WvPLStatus GeneratePsshData (const WvPLPsshParameters& pssh_params, std::map<string, string>* pssh) pssh_data_map)*
Generate the PSSH data as bytes. The bytes returned in *pssh_data* can be added to the Widevine PSSH box. The *pssh_data_map* is a key/value container where the key is the track type and the value is the PSSH data.

PSSH Information (WvPLPSSHParser)

Used to extract information from a Widevine PSSH. The pssh data is provided when the parser object is created. The public methods are:

- *const std::string& group_master_key_id()*
Extract and return the group master key id from the pssh data.

Appendix

Device Certificates

A device certificate status list is a list of intermediate certificates representing a device family. A device certificate status must exist for a device making the license request. If one is not found, `SERVICE_CERTIFICATE_NOT_FOUND` error is returned during the *GenerateLicense()* call unless “**allow_unknown_device**” is set to true in `WvPLEnvironment`. The server should hold onto the last good certificate status list and use this certificate status list to initialize the `WvPL` library on startup.

`allow_unknown_device`

Devices present their device certificate for verification. The root certificate is required to authenticate the device. The status list is required to determine if the device is still valid or has been revoked. `WvPLEnvironment` can be initialized with “allow-unknown-device” set to true. Setting “`allow_unknown_device`” means the status list is not used to verify the device’s certificate.

`device_certificate_expiration`

One of the configuration values specified during the construction of the Widevine environment object is “**device_certificate_expiration**”. This value indicates how many seconds before the device certificate status list expires. The server must call *WvPLEnvironment::UpdateWithCertificates()* before the current certificate list expires. For example, consider if the server is configured to call *UpdateWithCertificates()* every 24 hours and “`device_certificate_expiration`” is set to 72 hours. If the server is not able to update the certificates for 48 hours (no retries, 2 failures), the current certificate status list will remain valid for another 24 hours.

Fetching Certificate Status List

The URL (supports POST only) for fetching the certificate status list is:

```
https://www.googleapis.com/certificateprovisioning/v1/devicecertificatetestatus/list?key=AIzaSyDMLcEltgmHw8Eg5rUvrdPFgXT6VQl-rHQ
```

The entire JSON response can be provided as input to *WvPLEnvironment::UpdateWithCertificates()* method.

Example:

```
curl -d ""
"https://www.googleapis.com/certificateprovisioning/v1/devicecertificatestatus/list?key=AIzaSyDMLcE1tgmHw8Eg5rUvrdPFgXT6VQl-rHQ"

{
  "kind": "certificateprovisioning#certificateStatusListResponse",
  "signedList":
    "Cu1ICNGAp50FEkcKEIN2Jv3zB6GIvU0NznUu5tMQACIxCIAgEg1USSBPTUFQUyA0NDYwGgdTYW1zdW5nIgt0ZXh1cyBQcm1tZSoCdHYw2w84AxJACHCn68St4SaLcKwnTFRSA9SfEAAiKgiBIBIGVGvncmEyGgZudm1kaWEiB3NhbxN1bmcqB2FuZHJvaWQw2w84ARJLChDDA9vv1UD3qYKmf_V0tqa7EAAiNQiCIBIGVGvncmEyGglmYXV4I
  ...
}
```

Service Certificates

A service provider hosting a license server integrated with Widevine **must obtain a service certificate from Google**. The service certificate is generated by Google and used to authenticate the service provider. The provider should generate a public/private key pair and send the public key to Google. Google will generate the appropriate service certificate and return it back to the provider.

Service Certificate installed on the Server

The service certificate is installed on the hosted license server by calling `WvPLEnvironment::SetServiceCertificate(...)`. Caller must specify the service certificate and private key in binary format.

Service Certificate installed on the Client

If the client does not already have a service certificate, the client will generate a license request indicating it needs a service certificate. The result of calling `WvPLEnvironment::CreateSession()` with this license request will fail with error 139 (SERVICE_CERTIFICATE_REQUEST_MESSAGE). Anytime `CreateSession()` fails, `WvPLEnvironment::GenerateErrorResponse()` should be called. In this case, `GenerateErrorResponse()` will put the service certificate in the license response and return true. The license response should be sent back to the client.

Requesting a Service Certificate

Generate a public/private key pair using OpenSSL as follows:

- `openssl genrsa -out private.pem 2048`. This generates a 2048 bit private RSA key. 1024 and 3072 bit keys are ok as well, but 2048 is recommended.
- `openssl rsa -in private.pem -out public.der -outform DER -RSAPublicKey_out` . This exports the public key portion in RSAPublicKey DER format. On Mac, replace `-RSAPublicKey_out` option with `-pubout`
- `openssl pkcs8 -v2 des3 -topk8 -in private.pem -out private.der -outform DER` . This exports the private key in PKCS#8 EncryptedPrivateKey format. The user will be prompted for a password, which must be saved separately.
- `shred -u private.pem` . Removes the unencrypted copy of the private key.
- Send the public key (public.der) to Widevine for fulfillment [via the contact form](#).

Initialization Issues

Early License SDK users (prior to 2016) may encounter issues with their service certificates due to a format change.

If your SDK initialization is producing an error similar to below, please re-format your private key.

```
Add Service Certificate  
[libprotobuf ERROR internal/rsa_util.cc:356] d2i_PKCS8PrivateKey_bio returned NULL.  
INVALID_SERVICE_PRIVATE_KEY
```

Re-format your existing private key.

- Remove the password protection from your private key.
- Export your private key using the same password, but using "-v2 des3".

```
openssl pkcs8 -v2 des3 -topk8 -in private.pem -out private.der -outform DER
```

License Renewals

The use cases for license renewals is documented [here](#). To enable license renewals at the license SDK, the content provider must set the renewal flag on the playback policy.

```
WvPLPlaybackPolicy wvpl_playback_policy;  
wvpl_playback_policy.set_can_renew(true);
```

The generated license will indicate to the Widevine CDM that a RENEWAL request must be issued and license renewals are thus enabled.

Additionally, the content provider can also specify signing keys which allow the Widevine CDM to sign the renewal request using a “renewal signing key”.

This can be enabled by setting a “master signing key”.

If specifying the master signing key, the content provider must generate a 128-bit key and set it in the session initialization.

```
WvPLSessionInit wvpl_session_init;  
wvpl_session_init.set_master_signing_key(<128-bit key>);
```


Offline License

Two types of licenses exist, streaming and offline. A streaming license is used in most cases where the content is streamed to the device and is viewed as it is streamed. This type of license is loaded in memory and is never persisted on the device. Once the session is closed on the device, the license is cleared from the device. The license type (STREAMING or OFFLINE) is specified in the license request on the client.

An offline license is used where the content is stored on the device for later viewing, generally in cases when the device will be offline. This type of license is persisted on the device and can be loaded at a later time without contacting a license server. An offline license remains on the device until it is released.

The license server proxy must indicate that the license can persist on the device by setting 'can_persist=true' in the license policy. Also, license renewals should not be set to true as shown below.

```
License.Policy playback_policy;  
playback_policy.set_can_persist(true);  
playback_policy.set_can_renew(false);    // This is the default.
```

Offline License Release

Once a license is persisted on the device and the client wants to remove the license, the application will initiate a license release. This will generate a license request with the request type (NEW, RENEWAL or RELEASE) set to RELEASE.

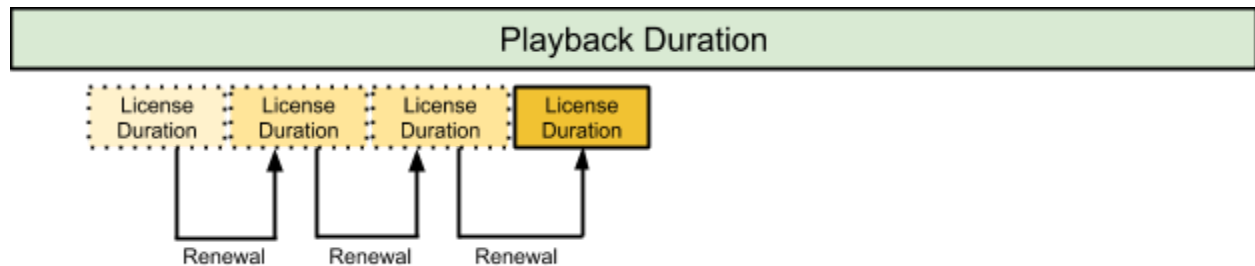
When handling a license where the request type is set to RELEASE, the policy's *can_play* attribute must be set to false as shown below.

```
License.Policy playback_policy;  
playback_policy.set_play(false);
```

The Widevine client will not delete the license from the device until it receives a valid license response indicating the license release was handled by the license server. The proxy must return the license to the client.

Durations

The illustration below shows the different durations in [WvPLPlaybackPolicy](#) and how they are related.



[License duration](#) governs the length of time allowed for the decryption and display of content media. It indicates how long the existing license is valid.

The playback duration value is optional.

If [playback duration](#) is unspecified, it is unlimited.

The additional duration values for playback provide additional business logic:

- To provide a limited time period in which a license is valid.
- To enable [support of license renewals](#).
 - With each successful renewal, the license duration is extended.
 - With continued successful renewals, the license duration can be extended to encompass the length of the media.
 - A renewal failure would result in premature license expiration.

Device Security Level

Security Level	Secure Boot Loader	Widevine Key Provisioning	Security Hardware or Trusted Execution Environment	Widevine Keybox and Video Key Processing	Hardware Video Path
Level 1	Yes	Factory	Yes	Keys never exposed in clear to host CPU	Hardware Protected Video Path
Level 2	Yes	Factory	Yes	Keys never exposed in clear to host CPU	Clear video streams delivered to renderer
Level 3	No	Field	No	Clear keys exposed to host CPU	Clear video streams delivered to decoder

Robustness Requirements

Security Level	Description
SW_SECURE_CRYPTO	Software-based whitebox crypto is required.
SW_SECURE_DECODE	Software crypto and an obfuscated decoder is required.
HW_SECURE_CRYPTO	The key material and crypto operations must be performed within a hardware backed trusted execution environment.
HW_SECURE_DECODE	The crypto and decoding of content must be performed within a hardware backed trusted execution environment.
HW_SECURE_ALL	The crypto, decoding and all handling of the media (compressed and uncompressed) must be handled within a hardware backed trusted execution environment.

Platform Verification Status (VMP Status)

Status	Description
PLATFORM_NO_VERIFICATION	The device platform does not support VMP.
PLATFORM_UNVERIFIED	The device platform status cannot be verified.
PLATFORM_SOFTWARE_VERIFIED	The device platform status was verified at the software level.
PLATFORM_HARDWARE_VERIFIED	The device platform status was verified at the hardware level.
PLATFORM_TAMPERED	The device platform status has been found to be tampered.

UsageStats destination

The statistics collected from using the [UsageStats API](#) would be sent to Google using a HTTP POST command.

Production
https://sdk.widevine.com/stats/setsdkstats

An example command (using curl):

```
curl -d "<serialized signed license stats>" https://sdk.widevine.com/stats/setsdkstats
```

Sample Usage Log

```
provider: "widevine_test"
counter_utc_start_time_usec: 1484553497
counter_utc_end_time_usec: 1484557387
counter_by_systemid {
  device_system_id: 4183
  counter_by_make {
    device_make: "Google"
    counter_by_model {
      device_model: "ChromeCDM-Linux-x86"
      counter_by_status {
        license_status: 0
        count: 3
      }
    }
  }
}
counter_by_systemid {
  device_system_id: 4349
  counter_by_make {
    device_make: "Google"
    counter_by_model {
      device_model: "ChromeCDM-Mac-x86"
      counter_by_status {
        license_status: 0
        count: 26
      }
    }
  }
}
counter_by_systemid {
  device_system_id: 4445
```

```

counter_by_make {
  device_make: "Generic Field Provisioning"
  counter_by_model {
    device_model: "Android KLP ARM"
    counter_by_status {
      license_status: 0
      count: 19
    }
  }
}

counter_by_systemid {
  device_system_id: 6100
  counter_by_make {
    device_make: "Google"
    counter_by_model {
      device_model: "iOS"
      counter_by_status {
        license_status: 0
        count: 7
      }
    }
  }
}

counter_by_systemid {
  device_system_id: 6205
  counter_by_make {
    device_make: "Samsung"
    counter_by_model {
      device_model: "zerolte"
      counter_by_status {
        license_status: 0
        count: 5
      }
    }
  }
}

counter_by_systemid {
  device_system_id: 6346
  counter_by_make {
    device_make: "LGE"
    counter_by_model {
      device_model: "p1"
      counter_by_status {
        license_status: 0
        count: 5
      }
    }
  }
}

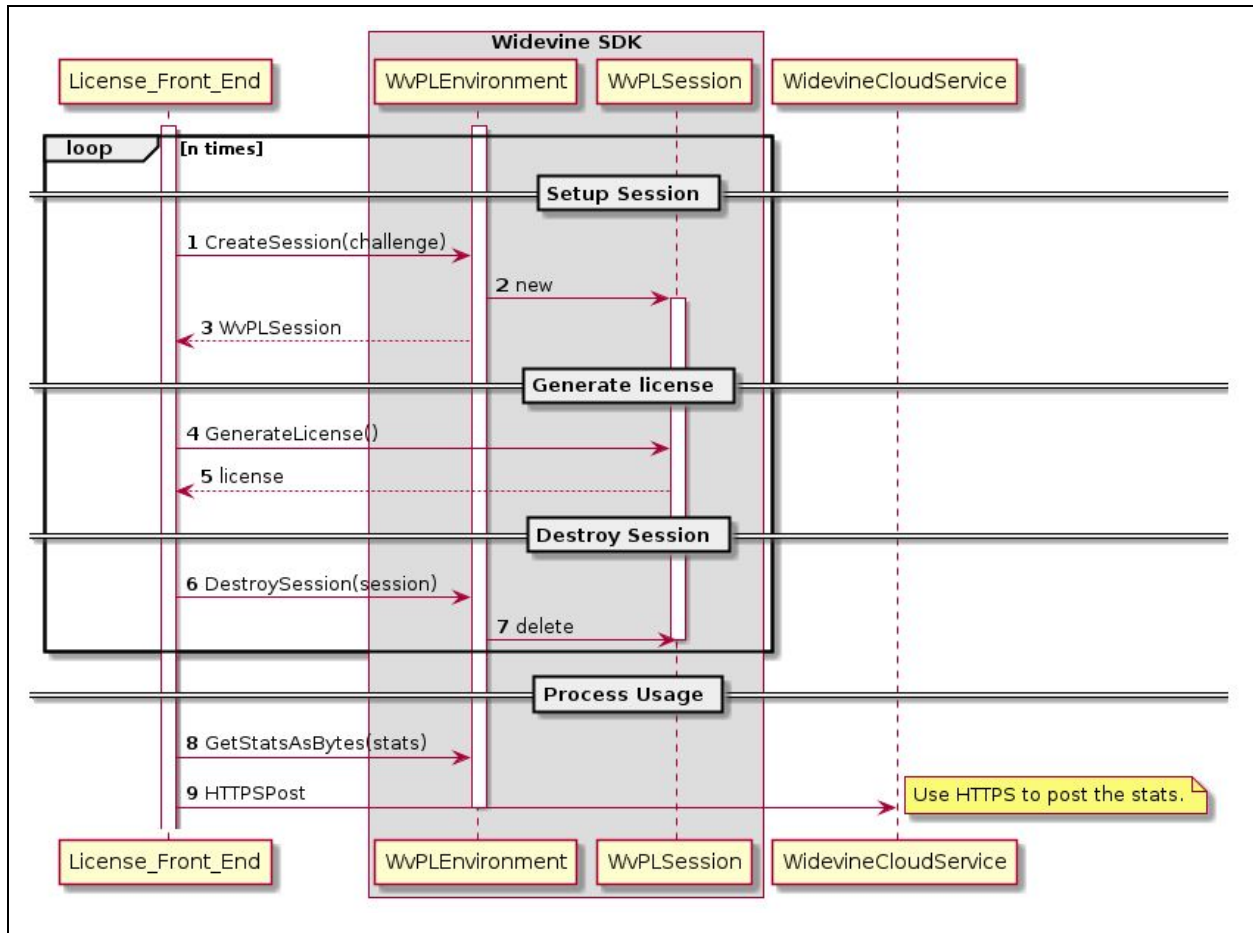
```

```
}  
}
```

UsageStats workflow

This workflow describes the expected operation of UsageStats.

- Initialize the environment with your IV and Key pair.
 - [Contact Widevine](#) if you need your credentials.
- At regular intervals, retrieve the data.
 - We recommend executing data collection on a daily basis.
 - Optional step - provides human-readable logs of the stats for your consumption only.
 - Call `GetStatsAsString` and store the data.
 - Set *flush_data* to false.
 - This step is optional. It provides human-readable logs for your consumption only.
 - Call `GetStatsAsBytes` and store the data.
 - Set *flush_data* to true.
- Send (using HTTPS POST) the `GetStatsAsBytes` data to Google when the data is retrieved.
 - The recommended interval for data submission is as soon as possible, daily is recommended.



Group License

For more information, please see the [separate Group License document](#).

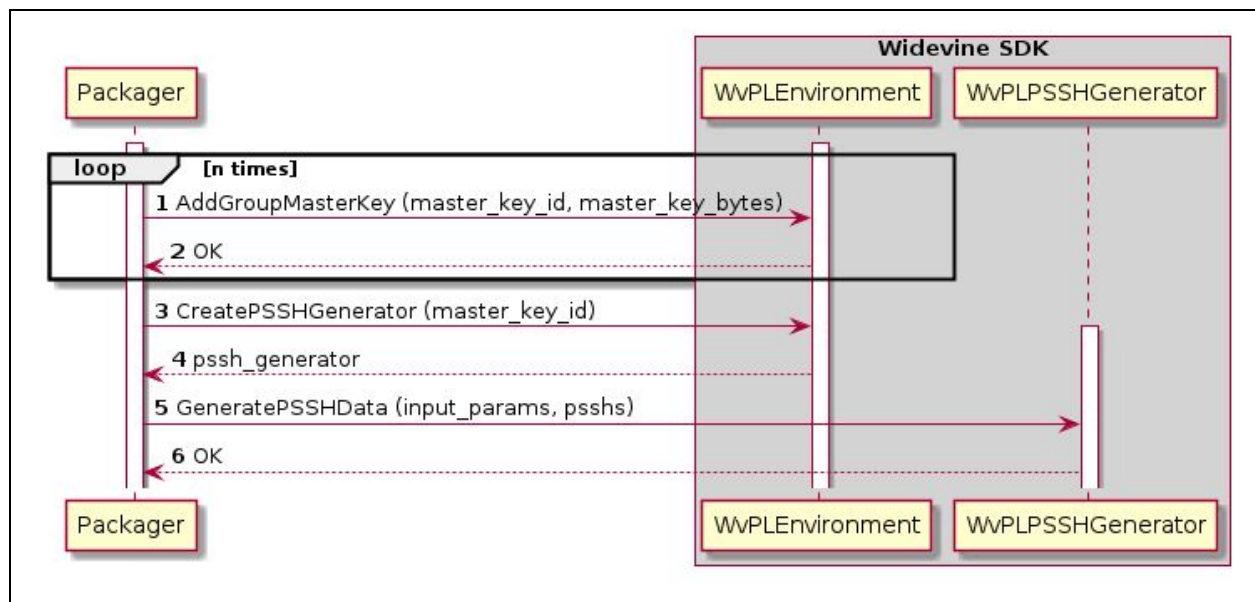
For VOD

Group license allows content providers to group multiple pieces of content into single group identified by a group Id. Each content (and track) in the group is encrypted with a unique content key. A single license (from the license server) for the group can be used to decrypt each content in the group.

For Live Streaming

The group license can also be used for key rotation when streaming live content. The DRM client will fetch a license for the stream if it does not have a key for the content. As keys are rotated within the stream, the same license can be used to decrypt the rotated stream. This removes the need to fetch a new license for each crypto period.

Packaging workflow



ARM Support

We successfully built and tested an ARM version of the Widevine License Server SDK on a Raspberry Pi device. This has seen limited deployment and development efforts are more involved. Below is a list of requirements and common issues that we have encountered during integration with partners.

Requirements

- Raspberry Pi running the platform recommended Raspbian 9
- Protobuf-2.6.1 to build libprotobuf.so.9
- Java Development Kit - Hard Float ABI
- Boringssl
- If using the Java API verify that the LD_LIBRARY_PATH IS SET

Common Issues

- If gcc and g++ compilers are updated to build libprotobuf, be sure to rebuild boringssl with the updated compilers.
- If this error is encountered `“libprotobuf ERROR internal/rsa_util.cc:356] d2i_PKCS8PrivateKey_bio returned NULL”`, it is recommended to reformat your private key.
 - Re-format your existing private key.
 - Remove the password protection from your private key.
 - Export your private key using the same password, but using `"-v2 des3"`.

```
openssl pkcs8 -v2 des3 -topk8 -in private.pem -out private.der -outform DER
```

Contact Us

For any questions, please [contact Widevine](#) from www.widevine.com.