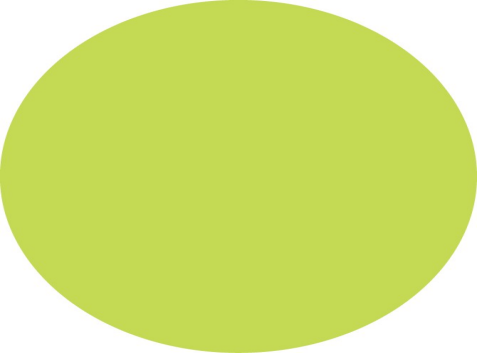

PVMF Return Codes
OHA 2.05, rev. 1
Sep 1, 2009

Table of Contents

1. Introduction

All PV software development kits (SDKs) use the return codes defined in PV multimedia framework (PVMF) architecture for synchronous as well as asynchronous return status for its entire application programming interface (API). The codes for asynchronous informational events or error events are also from with in the set of codes defined in the header `pvmi\pvmf\include\pvmf_return_codes.h`. The error code has been given negative values and informational code has positive values. There are a few "placeholder" codes defined, which are not currently being used for any general, error or informational status. To understand which informational or error codes are valid and expected for a particular API please refer to the specific API documentation.

2. General Return Codes

2.1. PVMFNotSet

This is a "placeholder" code and no API is expected to return this code in normal circumstances. This value can be initialize any return_value variable of type PVMFStatus.

2.2. PVMFPending

This code represents that completion of the particular API is "pending". In an asynchronous framework, there might be some other callback functions defined which should be called when completion of a "pending" API happens.
Please note that its value is 0, but 0 should not be treated as failure.

2.3. PVMFSuccess

This code is for general success.

3. Error Codes

All error codes are defined to have negative values with in the range of –1 to –100 (both included). The macro **`IsPVMFErrCode`** can be used to check if a particular code is with in this range.

3.1. PVMFErrAccessDenied

This error is reported when a resource is accessed without full authorization. A typical example is when rights management does not allow a playback to happen.

3.2. PVMFErrAlreadyExists

This error happens when a resource already exists and another one cannot be created. For example creating or registering some plug-in with specific attributes which is already created or registered.

3.3. PVMFErrArgument
This error is reported if invalid arguments are passed to an API. Please note that validation of argument might not be possible as late as actual processing with “wrong” argument fails.

3.4. PVMFErrBadHandle
This error happens when some invalid resource handle is specified.

3.5. PVMFErrBusy
This error happens when underlying resource is busy and the new request cannot be handled.

3.6. PVMFErrCallbackClockStopped
This error is specific to PVMFMediaClock and means that media clock has stopped.

3.7. PVMFErrCallbackHasBecomeInvalid
This error is specific to PVMFMediaClock and it means that media clock callback has become invalid due to change in direction of NPT clock.

3.8. PVMFErrCancelled
This error code is returned when some old request is cancelled. In that sense this error is actually an expected value and should not be treated as a fatal error.

3.9. PVMFErrContentTooLarge
This error happens when the download content length is larger than the maximum requested size.

3.10. PVMFErrCorrupt
This error occurs due to data corruption being detected. A typical example is when an invalid media stream is encountered in datapath.

3.11. PVMFErrDrmClockError
This error occurs when the DRM clock is not available or cannot be read.

3.12. PVMFErrDrmClockRollback
This error occurs when the DRM clock rollback is detected.

3.13. PVMFErrDrmDeviceDataAccess
This error occurs when access to the DRM device data fails.

3.14. PVMFErrDrmDeviceDataMismatch
This error occurs when DRM data is not matched to the device in use.

3.15. PVMFErrDrmDeviceIDUnavailable
This error occurs when DRM device ID cannot be determined.

3.16. PVMFErrDrmCryptoError
This error occurs when DRM cryptography operation failed.

3.17. PVMFErrDrmDomainNotAMember
This error occurs when a license server reports that the device is not part of the domain.

3.18. PVMFErrDrmDomainRequired
This error occurs when a license server requests registration to a domain.

3.19. PVMFErrDrmDomainRenewRequired
This error occurs when a license server requests renewal of a domain registration.

3.20. PVMFErrDrmInsufficientRights
This error occurs when the requested operation has insufficient DRM rights.

3.21. PVMFErrDrmLicenseExpired
This error occurs when DRM license has expired due to end time or usage count restriction.

3.22. PVMFErrDrmLicenseNotFound
This error occurs due to the lack of a valid license for the content.

3.23. PVMFErrDrmLicenseNotFoundPreviewAvailable
This error happens when a valid license for the content is required but not available, however a preview is available without license. The current playback has to be stopped and playback is to be restarted in preview mode. Please refer to the PVPlayer developer's guide for more details.

3.24. PVMFErrDrmLicenseNotYetValid
This error occurs when the DRM license has a start time restriction and current time is too early.

3.25. PVMFErrDrmLicenseStoreAccess
This error occurs when access to the DRM license store fails.

3.26. PVMFErrDrmLicenseStoreCorrupt
This error occurs when the DRM license store is corrupted.

3.27. PVMFErrDrmLicenseStoreInvalid
This error occurs when the DRM license store is not valid for the device in use.

3.28. PVMFErrDrmNetworkError
This error occurs when DRM network error occurred in server communication.

3.29. PVMFErrDrmOperationFailed
This is a generic DRM operational error not otherwise specified in the list.

3.30. PVMFErrDrmOutputProtectionLevel
This error occurs when DRM rights require higher output protection level than supported by the device.

3.31. PVMFErrDrmServerError
This error occurs when the underlying DRM server fails to respond.

3.32. PVMFErrFirst
This is placeholder and is not an actual error code and not expected to be returned by any API. It is defined as boundary of error codes

3.33. PVMFErrHTTPAuthenticationRequired
This error happened when there is a requirement of user-id and password input from SDK user for HTTP basic/digest authentication.

3.34. PVMFErrInvalidState
This error happens when some request is done to a resource, which is not in the expected “state” to handle that requests. The possible states and state transition of a resource are predefined. A typical example is calling PVPlayer Start without calling PVPlayer Prepare resulting in invalid state.

3.35. PVMFErrLast
This is a “placeholder” code and no api is expected to return this code in normal circumstances.

3.36. PVMFErrMaxReached
This error happens when maximum number of objects in use is reached. A typical example is, when after reaching the limit specified by user, more media items couldn’t be added in the database.

3.37. PVMFErrNoMemory
This error happen when any operation failed due to non-availability of memory.

3.38. PVMFErrNoResources
This error is returned if the resource required in processing of a request is not being available. A typical example is, a socket node connection not available for streaming.

3.39. PVMFErrNotReady

This error happens when a particular resource is not ready (due to some omission) to accept a request. A typical example is failure of PVPlayer Prepare failure if no data-sink has been added.

3.40. PVMFErrNotSupported

This error is reported when some particular request (API, feature, format etc) is not supported in current SDK or in the current configuration of SDK.

3.41. PVMFErrOverflow

This error is reported for case when some sort of overflow has happened for example buffer to hold media data is not big enough. Please note that this error code is different from info code PVMFInfoOverflow that is not treated as fatal error.

3.42. PVMFErrPortProcessing

This error is reported by nodes when they encounter any general error in port processing. This error is not exposed to SDK user.

3.43. PVMFErrProcessing

This error code is used to declare a general data processing error, like data flow could not happen.

3.44. PVMFErrResource

This error code is used for any general error happening in underlying resource. Error message should be checked for more specific info.

3.45. PVMFErrResourceConfiguration

This error happens when some request try to do an invalid configuration of a resource. A typical scenario is, when datapath could not be created using specified data source and sinks.

3.46. PVMFErrTimeout

This error is reported when any request has timed out. In most of the case there are ways provided by PV SDK to set or modify the time-out duration for a particular API. Please refer API specifications for more details.

3.47. PVMFErrUnderflow

This error happens if insufficient data is available for processing. A typical example is, the audio decoder starving due to non-availability of sufficient data.

3.48. PVMFFailure

This return code is for general failure. Please refer API documentation for possible cause of failures. Please note that its value is also (-1)

3.49. PVMFLowDiskSpace

This is an error code returned for low disk space. It is different from PVMFErrNoMemory as this error refers specifically to physical memory space for storage like in download or MTP song transfer session.

3.50. PVMFErrContentInvalidForProgressivePlayback

This is an error code returned when the video container is not valid for progressive playback.

4. Informational codes

All informational codes are defined to have positive values with in the range of 10 to 100 (both included). The macro **`IsPVMFInfoCode`** can be used to check if a particular code is with in this range.

4.1. PVMFInfoBufferCreated

This is a notification that a data buffer has been created. This is an internal event.

4.2. PVMFInfoBufferingComplete

This event indicates the end of the buffering events. For ***RTSP streaming***, it means that the internal data buffer has reached its upper threshold.
For ***PS/PDL***, this event means that the data till the end of the content length has been downloaded. In this case, the total length of the content downloaded accompanies the event.

```
if(aEvent.GetEventType()==PVMFInfoBufferingComplete)
{
    PVExclusivePtr eventData;
    aEvent.GetEventData(eventData);
    uint32 contentSize = (uint32)(eventData);
}
```

4.3. PVMFInfoBufferingStart

Notification that buffering of data has started. In case of streaming, or pseudo-streaming scenarios, this event will also follow the PVMFInfoUnderFlow event. This event indicates that the source nodes, after achieving an underflow, have begun receiving more data.

4.4. PVMFInfoBufferingStatus

This notification is used to tell the data buffering level status.

For ***RTSP streaming***, an internal buffer of X secs is maintained. When an underflow happens, a PVMFInfoUnderflow event Is sent. Then, the first byte of data arriving would signal a PVMFInfoBufferingStart event. Hereafter, further data filling up this buffer would generate the

PVMFInfoBufferingStatus events. The associated data shows the percentage of the X secs filled up in the buffer. In this case, the event is sent up once every 200 msecs.

For ***PS/PDL*** scenarios, this event indicates the percentage of data already downloaded compared to the total size of the content. If the content length is not known, it merely indicates the number of bytes of data currently downloaded. In this case, the event is sent only for integer increments of percentage. As a future enhancement, this event will also be send periodically.

As noted above, for ***PS/PDL*** scenarios, the percentage value can be either time-based or byte-based. By default, byte-based percentages are sent. See section 9.2.1 of pvplayer_developers_guide for more details.

```
if(aEvent.GetEventType()==PVMFInfoBufferingStatus)
{
    int32* percent=(int32 *)aEvent.GetLocalBuffer();
}
```

4.5. PVMFInfoChangePlaybackPositionNotSupported

This is a notification that change position request not supported. This event can be used to indicate that a requested change in playback position by the user of the SDK is not supported. An example where this can be expected is when a user tries to reposition a live RTSP streaming session. This event is currently not being supported.

4.6. PVMFInfoContentLength

This event notifies the user of the length of the content being downloaded in the case of PS or PDL. This event is sent during the Init() call being processed by the pvPlayer.

```
if (aEvent.GetEventType()==PVMFInfoContentLength)
{
    PVExclusivePtr eventData;
    aEvent.GetEventData(eventData);
    uint32 contentSize = (uint32)(eventData);
    fprintf(file,"   PVMFInfoContentLength = %d\n", contentSize);
}
```

4.7. PVMFInfoContentTruncated

This event is to indicate that the data sent by the server is truncated. It can happen when (I) the content length is not known and the downloaded data is greater than the maximum file size set, (ii) the content length is known and the downloaded data is smaller than the content length (happens in the case of server disconnect). The event data provides the amount of data actually downloaded until the point of truncation.

```
if (aEvent.GetEventType()==PVMFInfoContentTruncated)
{
    PVExclusivePtr eventData;
    aEvent.GetEventData(eventData);
```

```
    uint32 downloadSize = (uint32)(eventData);
    fprintf(file,"PVMFInfoContentTruncated!downloadsize=
%d\n",downloadSize);
    PVMFErrorInfoMessageInterface *msg=NULL;

    if(aEvent.GetEventExtensionInterface()&&
aEvent.GetEventExtensionInterface()->queryInterface
(PVMFErrorInfoMessageInterfaceUUID,(PVInterface*&)msg))
    {
        //extract the event code and event UUID.
        int32 eventcode;
        PVUuid eventUuid;
        msg->GetCodeUUID(eventcode,eventUuid);
        if(eventUuid == PVPlayerErrorInfoEventTypesUUID)
        {
            PVMFErrorInfoMessageInterface*sourceNodeInfoIF=
            msg->GetNextMessage();
            if(sourceNodeInfoIF != NULL)
            {
                PVUuid infoUUID;
                int32 srcInfoCode;
                sourceNodeInfoIF->GetCodeUUID(srcInfoCode, infoUUID);
                if (infoUUID==PVMFPROTOCOLENGINENODEInfoEventTypesUUID
                    && srcInfoCode==
 PVMFPROTOCOLENGINENODEInfo_TruncatedContentByServerDisconnect)

                 {
                     fprintf(file," PVMFInfoContentTruncated!
TruncatedContentByServerDisconnect! \n");
                 }
             }
         }
    }
}
```

4.8. PVMFInfoContentType

This event notifies the user the type of content being downloaded in the case of PS or PDL. This provides the user of the SDK an opportunity to make a decision whether or not to continue the playback. This event is sent during the Init() call being processed by the pvPlayer SDK.

```
if (aEvent.GetEventType()==PVMFInfoContentType)
{
    PVExclusivePtr eventData;
    aEvent.GetEventData(eventData);
    char *constentType = (char *)(eventData);
    fprintf(file,"   PVMFInfoContentType   = %s\n", constentType);
}
```

4.9. PVMFInfoDataDiscarded

This event indicates that certain media messages have been discarded during synchronization. A common scenario when this event can be expected is in the case of an Audio-Video playback session, where the platform is not capable of decoding and rendering both audio and video messages in time. In such a case, video frames are discarded to allow a smoother playback of audio. This event is currently not being sent.

4.10. PVMFInfoDataReady

This event indicates that enough data has been buffered for a playback session to start or resume. For http streaming cases, the user of the SDK is expected to signal a Start, only when this event is received for the first time, meaning enough media data has been buffered to start playback smoothly. This event is also generated when the player SDK buffers enough data after auto-pause, and then is about to resume playback. But, under such scenarios, the user of the SDK is expected to ignore the event (i.e., a Start or Resume is not expected).

4.11. PVMFInfoDurationAvailable

Notification that duration is available with source node.
This event is sent to the user of the SDK to indicate that the duration of the content being requested to play is available in PVMFDurationInfoMessageInterfaceUUID event code space. There is no need to query for the duration explicitly. The value sent is in milliseconds like this sample code.

```
if(aEvent.GetType() == PVMFInfoDurationAvailable)
{
PVUuid infomsguuid = PVMFDurationInfoMessageInterfaceUUID;
PVMFDurationInfoMessageInterface* eventMsg = NULL;
    PVInterface* infoExtInterface =
aEvent.GetEventExtensionInterface();
    if (infoExtInterface &&
infoExtInterface->queryInterface(infomsguuid,
(PVInterface*&)eventMsg))
    {
        PVUuid eventuuid;
        int32 infoCode;
  eventMsg->GetCodeUUID(infoCode, eventuuid);

  if (eventuuid == infomsguuid)
        {
            uint32 SourceDurationInMS = eventMsg->GetDuration();
        }
    }
}
```

4.12. PVMFInfoEndOfData

Notification that end of data stream has been reached. This event indicates that a particular node has received its last message for the particular session of the playback. Internally, the source

nodes, decoder nodes, and sink nodes, all send this event. However, to the user of the SDK, only the event originated from the sink nodes is visible, because this is a true indicator of the end of the current playback session.

4.13. PVMFInfoErrorHandlingComplete

Notification that error handling has completed. Once the pvPlayer SDK goes into an error-handling mode, it does not allow any more commands to be queued. If attempted, the command will “leave” with an error of “invalid state”. So, the user of the SDK is expected to check the state of the engine before queuing any command, and if it is in an “Error state”, the user needs to wait for the PVMFInfoErrorHandlingComplete event before attempting to queue any further command.

4.14. PVMFInfoErrorHandlingStart

This is an informational event circulated internal to the pvPlayer SDK. This event indicates that one of the underlying components has erred out, and a cleanup will commence soon. This event is not sent to the user of the SDK because a PVMF error event will be sent, indicating the actual cause of the error. In future, this event may be sent to the user of the SDK.

4.15. PVMFInfoFirst

This is placeholder and is not an actual inof code and not expected to be returned by any API. It is defined as boundary of info codes.

4.16. PVMFInfoLast

This is a “placeholder” code and no api is expected to return this code in normal circumstances.

4.17. PVMFInfoLicenseAcquisitionStarted

A notification to indicate the License acquisition has started in case of CPM plug-ins. It depends on the author of the CPM plug-in to use this event or not.

4.18. PVMFInfoMetadataAvailable

Notification that metadata is available with source node.

4.19. PVMFInfoOverflow

This is a notification that an overflow occurred (not fatal error). It just informs that one of the ports trying to be processed is full. This would usually pop up when the receiving node is slower in consuming data when compared to the sending node.

4.20. PVMFInfoPlayListClipTransition

This event notifies the user of the pvPlayer SDK that a clip transition has occurred while playing a playlist.

```
if(PVMFInfoPlayListClipTransition == aEvent.GetEventType() )
{
    PVExclusivePtr aPtr;
    aEvent.GetEventData(aPtr);
    PVMFRTSPClientEngineNodePlaylistInfoType *myType =
```

```
                                        (PVMFRTSPClientEngineNodePlaylistI
nfoType*)(aPtr);
    fprintf(iTestMsgOutputFile,"###PVMFInfoPlayListClipTransition:\n
             <%s, %d, %d.%03d>;npt=%d.%03d;mediaName=%s;userData=%s\n",
                                 myType->iPlaylistUrlPtr
                                , myType->iPlaylistIndex
                                , myType->iPlaylistOffsetSec
                                , myType->iPlaylistOffsetMillsec
                                , myType->iPlaylistNPTSec
                                , myType->iPlaylistNPTMillsec
                                , myType->iPlaylistMediaNamePtr
                                , myType->iPlaylistUserDataPtr);
}
```

4.21. PVMFInfoPoorlyInterleavedContent

This is a notification that the content is poorly interleaved.

For Progressive Streaming scenarios, there is only X number of bytes buffered at any point of time. If the content being streamed is so interleaved that the location of a media sample of one track differs by more than X bytes compared to the location of the media sample of another track for the same timestamp, this informational event is sent to the user of the SDK. It is not a favorable way of streaming such content because for every sample being retrieved, the entire buffer has to be flushed out and refilled. This would result in a very poor user experience. It is advisable that upon receiving such an event, the playback session is stopped.

4.22. PVMFInfoPortConnected

This is a notification that a port was connected. This is an internal event.

4.23. PVMFInfoPortCreated

This is a notification that a port was created. This is an internal event.

4.24. PVMFInfoPortDeleted

This is a notification that a port was deleted. This is an internal event.

4.25. PVMFInfoPortDisconnected

This is a notification that a port was disconnected. This is an internal event.

4.26. PVMFInfoPositionStatus

PVPlayer engine sends the current playback position periodically as an unsolicited informational event with PVMFInfoPositionStatus event code and player specific event code of PVPlayerInfoPlaybackPositionStatus in PVPlayerErrorInfoEventTypesUUID event code space. The position value is stored in the local data buffer of the informational event. The application is responsible for “listening” for this event in the informational event callback handler if it wants to obtain the current playback position by this method.

The position units and the time length of the reporting period can be queried and modified via the capability-and-configuration extension interface of PVPlayer engine.(please refer player guide for more details) The default settings are milliseconds for playback position units and 1000 milliseconds for the reporting period.

Support for non-time position units (like percentage) would change based on the underlying nodes and source media being used. Therefore, if support for non-time position units becomes unavailable, PVPlayer engine will automatically change to the default of milliseconds.
A sample code handling this event would look like

```
if (aEvent.GetEventType()==PVMFInfoPositionStatus)
{
    PVInterface* iface=(PVInterface*)
(aEvent.GetEventExtensionInterface());
    if(iface==NULL)
    {
        return;
    }
    PVUuid infomsguuid=PVMFErrorInfoMessageInterfaceUUID;
    PVMFErrorInfoMessageInterface* infomsgiface=NULL;
    if (iface->queryInterface(infomsguuid,
(PVInterface*&)infomsgiface)==true)
    {
        int32 infocode;
        PVUuid infouuid;
        infomsgiface->GetCodeUUID(infocode, infouuid);
        if ((infouuid==PVPlayerErrorInfoEventTypesUUID) &&
            (infocode==PVPlayerInfoPlaybackPositionStatus))
        {
            uint8* localbuf=aEvent.GetLocalBuffer();
            uint32 pbpos=0;
            oscl_memcpy(&pbpos, &(localbuf[4]), 4);
            fprintf(iTestMsgOutputFile, "Playback status(time) %d
ms\n", pbpos);
        }
    }
}
```

4.27. PVMFInfoProcessingFailure

This is a notification that a processing failure occurred (not fatal error). A usual scenario when this event is observed is in the case of processing a buffer by the decoder. Example: bitstream corruption.

4.28. PVMFInfoRemoteSourceNotification

This is a notification from a remote source. This event is an indicator of a notification sent by a remote source, as in a server in the case of streaming.
An example would be receiving the PVMFMP4FFParserInfoNotPseudostreamableFile info-code in a PVMFFileFormatEventTypesUUID event code space. Another example would be receiving a redirect code from a HTTP server.

4.29. PVMFInfoReportObserverRecieved

Notification that node has processed a command with ReportObserver marker info. This is also an internal event.

4.30. PVMFInfoSessionDisconnect

For PS/PDL scenarios, this event is sent that indicates that a server disconnect occurred after content download is complete. In case of PS, if server disconnect happens during download, we will do a retry instead of treating it as download complete. For PDL, in case of server disconnect during the download, and if content-length is not available, this event will be treated as download completion.

4.31. PVMFInfoSourceFormatNotSupported

This notification event is sent when the source format is not supported, typically sent during protocol rollover.

4.32. PVMFInfoStartOfData

Notification that new media stream has been started. This informational event is circulated **internally** in the pvPlayer SDK. The event is propagated to the player engine to indicate that the first media message has arrived the sink node for the particular playback segment. The playback clock is started only when this event is received from all the tracks/datapaths involved in the playback.

4.33. PVMFInfoStateChanged

This event notifies a change in the state of a component. This is an internal event.

4.34. PVMFInfoTrackDisable

Notification that paticular track is disable. This one is on a per track basis.
For uncompressed audio/video formats, during the process of selecting tracks available in content, if the decoder doesn't support the track, a PVMFInfoTrackDisable event is sent. The event, if necessary, will be sent once per track.

4.35. PVMFInfoUnderflow

This is a notification that an underflow occurred (not fatal error). An event sent to the user of the SDK to inform that there is not enough data to be processed. This is usually observed in case of streaming, pseudo-streaming, or any other scenarios where the data required to be processed can fall short of expectation. The user of the SDK can make use of this event to notify the end user that the playback session has temporarily paused, and that it will resume soon. A common usecase is to show a “buffering” notification under such a scenario.

4.36. PVMFInfoUnexpectedData

Notification that unexpected data has been obtained, especially for download, when client receives from server more data than requested in content-length header. This event is NOT sent if the PVMFInfoSessionDisconnect event has already been sent.

4.37. PVMFInfoVideoTrackFallingBehind

PVPlayer engine sends this event when messages of the video track are continuously discarded for a fixed number of frames. Currently, the number of frames is fixed to 120. As a future enhancement, the user of the PVPlayer SDK may be given an option to explicitly provide this value. This event can be used to indicate why audio continuous to play and video appears to freeze.

4.38. PVMFInfoSourceOverflow

This informational event is sent when the source node runs out of memory for incoming RTP packets.

5. Appendix

5.1. PVPlayer engine error and extension codes with their likely causes

The table in this appendix lists error codes used by the PVPlayer engine and associated extension codes along with a likely cause.

Error code	Error Extension codes	Likely cause
PVMFErrCorrupt	PVPlayerErrSourceMediaData	Invalid media data detected in the source node (e.g. invalid sample in file)
	PVPlayerErrSinkMediaData	Invalid media data detected in sink node (e.g. invalid decoded data for video display)
	PVPlayerErrDatapathMediaData	Invalid media data detected in one of the datapath node (e.g. invalid bitstream in the decoder node)
PVMFErrOverflow	PVPlayerErrSourceMediaData	Memory buffer overflowed in the source node (e.g. buffer to hold media data is not big enough)
	PVPlayerErrSinkMediaData	Memory buffer overflowed in the sink node (e.g. buffer to hold media data is not big enough)
	PVPlayerErrDatapathMediaData	Memory buffer overflowed in one of the datapath node (e.g. buffer to hold media data is not big enough)

Error code	Error Extension codes	Likely cause
PVMFErrResource	PVPlayerErrSourceMediaData	Error detected in an underlying resource used by the source node (e.g. file parser error)
	PVPlayerErrSourceInit	Error while initializing the source (e.g. file parsing error, file corrupt). Check error message for more specific info if available
	PVPlayerErrSinkFatal	Error detected in an underlying resource used by the sink node (e.g. video render device encountered unrecoverable error)
	PVPlayerErrDatapathFatal	Error detected in an underlying resource used by a datapath node (e.g. audio decoder encountered unrecoverable error)
PVMFErrProcessing	PVPlayerErrSourceMediaData	Error occurred in source node while processing media data (e.g. could not send on output port)
	PVPlayerErrSinkMediaData	Error occurred in sink node while processing media data (e.g. could not receive on input port)
	PVPlayerErrDatapathMediaData	Error occurred in datapath node while processing media data (e.g. could not send on output port)
PVMFErrUnderflow	PVPlayerErrSourceMediaUnavailable	Media data ran out unexpectedly in the source node (e.g. reached end of file)
	PVPlayerErrSinkMediaData	Media data ran out unexpectedly in the sink node (e.g. audio device underflows and then errors out)
	PVPlayerErrDatapathMediaData	Media data ran out unexpectedly in the one of the datapath node
PVMFErrNoResources	PVPlayerErrSourceFatal	Underlying resource needed by source node not available (e.g. socket connection not available for streaming)
	PVPlayerErrSinkFatal	Underlying resource needed by sink node not available (e.g. audio device not available)
	PVPlayerErrDatapathFatal	Underlying resource needed by datapath node not available (e.g. DSP video memory not available)
PVMFErrResourceConfiguration	PVPlayerErrSourceFatal	Underlying resource used by source node has a configuration error
	PVPlayerErrSinkFatal	Underlying resource used by sink node has a configuration error (e.g. audio device cannot handle the specified sampling rate)
	PVPlayerErrDatapathFatal	Underlying resource used by datapath node has a configuration error (e.g. video decoder cannot handle the specified dimension)

Error code	Error Extension codes	Likely cause
PVMFErrTimeout	PVPlayerErrSourceFatal	Timeout occurred in the source node or underlying resource (e.g. socket connection timeout in streaming)
	PVPlayerErrSinkFatal	Timeout occurred in the sink node or underlying resource (e.g. audio device timed out)
	PVPlayerErrDatapathFatal	Timeout occurred in the datapath node or underlying resource (e.g. hardware audio decoder timed out)
PVMFErrNoMemory	PVPlayerErrSourceFatal	Required amount of memory not available in the source node
	PVPlayerErrSinkFatal	Required amount of memory not available in the sink node
	PVPlayerErrDatapathFatal	Required amount of memory not available in the one of the datapath or one of the nodes in the datapath.
PVMFErrLicenseRequired	PVPlayerErrSourceInit	Authorization license needed to initialize the specified source
PVMFErrAccessDenied	PVPlayerErrSourceInit	Rights management does not allow playback of the specified source
PVMFFailure		General failure code. Components are not behaving as expected.