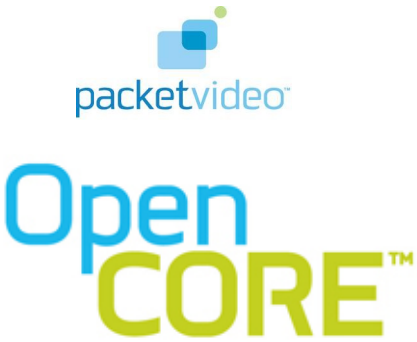


PV Player Engine

Build Version: CORE_9.000.1.1_RC3

May 14, 2010

# Contents

# Chapter 1

# Data Structure Index

## 1.1 Data Structures

Here are the data structures with brief descriptions:

# Chapter 2

# File Index

## 2.1 File List

Here is a list of all files with brief descriptions:

# Chapter 3

# Data Structure Documentation

## 3.1 PVPlayerInterface Class Reference

`#include <pv_player_interface.h>`

### Public Member Functions

- virtual ~PVPlayerInterface ()
- virtual PVCommandId GetSDKModuleInfo (PVSDKModuleInfo &aSDKModuleInfo, const OsclAny *aContextData=NULL)=0
- virtual PVCommandId SetLogAppender (const char *aTag, OsclSharedPtr< PVLoggerAppender > &aAppender, const OsclAny *aContextData=NULL)=0
- virtual PVCommandId RemoveLogAppender (const char *aTag, OsclSharedPtr< PVLoggerAppender > &aAppender, const OsclAny *aContextData=NULL)=0
- virtual PVCommandId SetLogLevel (const char *aTag, int32 aLevel, bool aSetSubtree=false, const OsclAny *aContextData=NULL)=0
- virtual PVCommandId GetLogLevel (const char *aTag, PVLogLevelInfo &aLogInfo, const OsclAny *aContextData=NULL)=0
- virtual PVCommandId QueryInterface (const PVUuid &aUuid, PVInterface *&aInterfacePtr, const OsclAny *aContextData=NULL)=0
- virtual PVCommandId CancelCommand (PVCommandId aCancelCmdId, const OsclAny *aContextData=NULL)=0
- virtual PVCommandId CancelAllCommands (const OsclAny *aContextData=NULL)=0
- virtual PVCommandId GetPVPlayerState (PVPlayerState &aState, const OsclAny *aContextData=NULL)=0
- virtual PVMFStatus GetPVPlayerStateSync (PVPlayerState &aState)=0
- virtual PVCommandId AddDataSource (PVPlayerDataSource &aDataSource, const OsclAny *aContextData=NULL)=0
- virtual PVCommandId UpdateDataSource (PVPlayerDataSource &aDataSource, const OsclAny *aContextData=NULL)=0
- virtual PVCommandId Init (const OsclAny *aContextData=NULL)=0
- virtual PVCommandId GetMetadataValues (PVPMetadataList &aKeyList, int32 aStartingValueIndex, int32 aMaxValueEntries, int32 &aNumAvailableValueEntries, Oscl_Vector< PvmiKvp, OsclMemAllocator > &aValueList, const OsclAny *aContextData=NULL, bool aMetadataValuesCopiedInCallBack=true, uint32 aClipIndex=0)=0

- virtual PVCommandId ReleaseMetadataValues (Oscl_Vector< PvmiKvp, OsclMemAllocator > &aValueList, const OsclAny *aContextData=NULL, uint32 aClipIndex=0)=0
- virtual PVCommandId AddDataSink (PVPlayerDataSink &aDataSink, const OsclAny *aContextData=NULL)=0
- virtual PVCommandId SetPlaybackRange (PVPPlaybackPosition aBeginPos, PVPPlaybackPosition aEndPos, bool aQueueRange, const OsclAny *aContextData=NULL, bool aSkipToRequestedPosition=true, bool aSeekToSyncPoint=true)=0
- virtual PVCommandId GetPlaybackRange (PVPPlaybackPosition &aBeginPos, PVPPlaybackPosition &aEndPos, bool aQueued, const OsclAny *aContextData=NULL)=0
- virtual PVCommandId GetCurrentPosition (PVPPlaybackPosition &aPos, const OsclAny *aContextData=NULL)=0
- virtual PVCommandId SetPlaybackRate (int32 aRate, PVMFTimebase *aTimebase=NULL, const OsclAny *aContextData=NULL)=0
- virtual PVCommandId GetPlaybackRate (int32 &aRate, PVMFTimebase *&aTimebase, const OsclAny *aContextData=NULL)=0
- virtual PVCommandId GetPlaybackMinMaxRate (int32 &aMinRate, int32 &aMaxRate, const OsclAny *aContextData=NULL)=0
- virtual PVMFStatus GetCurrentPositionSync (PVPPlaybackPosition &aPos)=0
- virtual PVCommandId Prepare (const OsclAny *aContextData=NULL)=0
- virtual PVCommandId Start (const OsclAny *aContextData=NULL)=0
- virtual PVCommandId Pause (const OsclAny *aContextData=NULL)=0
- virtual PVCommandId Resume (const OsclAny *aContextData=NULL)=0
- virtual PVCommandId Stop (const OsclAny *aContextData=NULL)=0
- virtual PVCommandId RemoveDataSink (PVPlayerDataSink &aDataSink, const OsclAny *aContextData=NULL)=0
- virtual PVCommandId Reset (const OsclAny *aContextData=NULL)=0
- virtual PVCommandId RemoveDataSource (PVPlayerDataSource &aDataSource, const OsclAny *aContextData=NULL)=0

## Static Public Member Functions

- static OSCL_IMPORT_REF void GetSDKInfo (PVSDKInfo &aSDKInfo)

### 3.1.1 Detailed Description

PVPlayerInterface is the interface to the pvPlayer SDK, which allows control of a multimedia playback engine. The PVPlayerFactory factory class is to be used to create and delete instances of this object

### 3.1.2 Constructor & Destructor Documentation

#### 3.1.2.1 virtual PVPlayerInterface::~PVPlayerInterface () [inline, virtual]

Object destructor function Releases all resources prior to destruction

## 3.1.3 Member Function Documentation

### 3.1.3.1 virtual PVCommandId PVPlayerInterface::AddDataSink (PVPlayerDataSink & *aDataSink*, const OsclAny * *aContextData* = NULL) [pure virtual]

This function allows a player data sink to be specified for playback. This function must be called when pvPlayer is in PVP_STATE_INITIALIZED state. The specified data sink must be a valid PVPlayerDataSink to be accepted for use in playback. This command request is asynchronous. PVCommandStatusObserver's CommandCompleted() callback handler will be called when this command request completes.

**Parameters**

***aDataSink*** The player data sink to be used for playback.

***aContextData*** Optional opaque data that will be passed back to the user with the command response This method can leave with one of the following error codes OsclErrNotSupported if the format of the sink is incompatible with what the SDK can handle OsclErrInvalidState if invoked in the incorrect state OsclErrNoMemory if the SDK failed to allocate memory during this operation

**Returns**

A unique command id for asynchronous completion

### 3.1.3.2 virtual PVCommandId PVPlayerInterface::AddDataSource (PVPlayerDataSource & *aDataSource*, const OsclAny * *aContextData* = NULL) [pure virtual]

This function allows a player data source to be specified for playback. This function must be called when pvPlayer is in PVP_STATE_IDLE state and before calling Init. The specified data source must be a valid PVPlayerDataSource to be accepted for use in playback. This command request is asynchronous. PVCommandStatusObserver's CommandCompleted() callback handler will be called when this command request completes.

**Parameters**

***aDataSource*** Reference to the player data source to be used for playback

***aContextData*** Optional opaque data that will be passed back to the user with the command response This method can leave with one of the following error codes OsclErrNotSupported if the format of the source is incompatible with what the SDK can handle OsclErrInvalidState if invoked in the incorrect state OsclErrNoMemory if the SDK failed to allocate memory during this operation

**Returns**

A unique command id for asynchronous completion

### 3.1.3.3 virtual PVCommandId PVPlayerInterface::CancelAllCommands (const OsclAny * *aContextData* = NULL) [pure virtual]

This API is to allow the user to cancel all pending requests in pvPlayer. The current request being processed, if any, will also be aborted. The user of PV-SDK should get the state of PVPlayer Engine after the command completes and before issuing any other command. This command request is asynchronous. PVCommandStatusObserver's CommandCompleted() callback handler will be called when this command request completes.

**Parameters**

***aContextData*** Optional opaque data that will be passed back to the user with the command response

**Returns**

A unique command id for asynchronous completion

#### 3.1.3.4 virtual PVCommandId PVPlayerInterface::CancelCommand (PVCommandId *aCancelCmdId*, const OsclAny * *aContextData* = NULL) [pure virtual]

This API is to allow user of the SDK to cancel any specific command which is pending on pvPlayer. If the request is to cancel a command which still has to be processed pvPlayer will just remove the command from its queue of commands to be processed. If the request is to cancel a command that is ongoing then player will attempt to interrupt the ongoing command. The state of player after a cancel can vary. So the user of pvPlayerSDK must always query for state before issuing any subsequent commands. This command request is asynchronous. PVCommandStatusObserver's CommandCompleted() callback handler will be called when this command request completes.

**Parameters**

***aCancelCmdId*** Command Id to be cancelled.

***aContextData*** Optional opaque data that will be passed back to the user with the command response

**Returns**

A unique command id for asynchronous completion

#### 3.1.3.5 virtual PVCommandId PVPlayerInterface::GetCurrentPosition (PVPPlaybackPosition & *aPos*, const OsclAny * *aContextData* = NULL) [pure virtual]

This function allows querying of the current playback position. The playback position units will be in the one specified by the passed-in reference to PVPPlaybackPosition. Currently only milliseconds units is supported. This command request is asynchronous. PVCommandStatusObserver's CommandCompleted() callback handler will be called when this command request completes.

**Parameters**

***aPos*** Reference to place the current playback position

***aContextData*** Optional opaque data that will be passed back to the user with the command response This method can leave with one of the following error codes OsclErrInvalidState if invoked in the incorrect state

**Returns**

A unique command id for asynchronous completion

#### 3.1.3.6 virtual PVMFStatus PVPlayerInterface::GetCurrentPositionSync (PVPPlaybackPosition & *aPos*) [pure virtual]

This function allows querying of the current playback position as a synchronous command. The playback position units will be in the one specified by the passed-in reference to PVPPlaybackPosition. Currently only millisecond units is supported.

**Parameters**

***aPos*** Reference to place the current playback position This method can leave with one of the following error codes OsclErrInvalidState if invoked in the incorrect state

**Returns**

Status indicating whether the command succeeded or not.

### 3.1.3.7 virtual PVCommandId PVPlayerInterface::GetLogLevel (const char * *aTag*, PVLogLevelInfo & *aLogInfo*, const OsclAny * *aContextData* = NULL) [pure virtual]

Allows the logging level to be queried for a particular logging tag. A larger log level will result in more messages being logged. In the asynchronous response, this should return the log level along with an indication of where the level was inherited (i.e., the ancestor tag). This command request is asynchronous. PVCommandStatusObserver's CommandCompleted() callback handler will be called when this command request completes.

**Parameters**

***aTag*** Specifies the logger tree tag where the log level should be retrieved.

***aLogInfo*** An output parameter which will be filled in with the log level information.

***aContextData*** Optional opaque data that will be passed back to the user with the command response

**Exceptions**

***This*** method can leave with one of the following error codes OsclErrNoMemory if the SDK failed to allocate memory during this operation

**Returns**

A unique command ID for asynchronous completion

### 3.1.3.8 virtual PVCommandId PVPlayerInterface::GetMetadataValues (PVPMetadataList & *aKeyList*, int32 *aStartingValueIndex*, int32 *aMaxValueEntries*, int32 & *aNumAvailableValueEntries*, Oscl_Vector< PvmiKvp, OsclMemAllocator > & *aValueList*, const OsclAny * *aContextData* = NULL, bool *aMetadataValuesCopiedInCallBack* = true, uint32 *aClipIndex* = 0) [pure virtual]

The function makes a request to return the metadata value(s) specified by the passed in metadata key list. If the requeted metadata value is unavailable or the metadata key is invalid, the returned list will not contain a KVP entry for the key. Note that value indexed in the returned aValueList does not necessary match the same index into the specified aKeyList since this command can return none or more than one KVP for a specified key. This command request is asynchronous. PVCommandStatusObserver's CommandCompleted() callback handler will be called when this command request completes.

**Parameters**

***aKeyList*** Reference to a list of metadata keys for which metadata values are requested.

***aStartingValueIndex*** The starting index refers to the an index into the whole value list specified by the keys in aKeyList. This command would populate the aValueList starting from the specified index.

***aMaxValueEntries*** Input parameter to specify the maximum number of entries to be added to aValueList. If there is no limit, set to -1.

***aNumAvailableValueEntries*** Output parameter which will be filled with number of available values for the specified key list.

***aValueList*** Reference to a vector of KVP to place the specified metadata values

***aContextData*** Optional opaque data that will be passed back to the user with the command response

***aMetadataValuesCopiedInCallBack*** Boolean to let engine know if metadata values are copied by User of SDK in command complete callback. By default the SDK assumes this to be the case. If this argument is set to false by the caller, then SDK assumes that user will call ReleaseMetaDataValues at a later point.

***aClipIndex,:*** an optional parameter for use with playlists to select the clip of interest. This method can leave with one of the following error codes OsclErrInvalidState if invoked in the incorrect state OsclErrNoMemory if the SDK failed to allocate memory during this operation

**Returns**

A unique command id for asynchronous completion

#### 3.1.3.9 virtual PVCommandId PVPlayerInterface::GetPlaybackMinMaxRate (int32 & *aMinRate*, int32 & *aMaxRate*, const OsclAny * *aContextData* = NULL) [pure virtual]

This function retrieves the minimum and maximum playback rate expressed as a millipercent of "real-time" playback rate. This function can be called anytime between pvPlayer instantiation and destruction. This command request is asynchronous. PVCommandStatusObserver's CommandCompleted() callback handler will be called when this command request completes.

**Parameters**

***aMinRate*** A reference to an integer which will be filled in with the minimum playback rate allowed expressed as millipercent of "real-time" playback rate.

***aMaxRate*** A reference to an integer which will be filled in with the maximum playback rate allowed expressed as millipercent of "real-time" playback rate.

***aContextData*** Optional opaque data that will be passed back to the user with the command response This method can leave with one of the following error codes

**Returns**

A unique command id for asynchronous completion

#### 3.1.3.10 virtual PVCommandId PVPlayerInterface::GetPlaybackRange (PVPPlaybackPosition & *aBeginPos*, PVPPlaybackPosition & *aEndPos*, bool *aQueued*, const OsclAny * *aContextData* = NULL) [pure virtual]

This function retrieves the playback range information for the current or queued playback range. The user can choose which playback range by the aQueued flag. This function can be called when pvPlayer is in PVP_STATE_INITIALIZED, PVP_STATE_PREPARED, PVP_STATE_STARTED, or PVP_STATE_PAUSED state. The units of position is specified in the passed-in PVPlaybackPosition parameters which will be filled in when the command completes. This command request is asynchronous. PVCommandStatusObserver's CommandCompleted() callback handler will be called when this command request completes.

**Parameters**

***aBeginPos*** Reference to place the begin position for the playback range

***aEndPos*** Reference to place the end position for the playback range

***aQueued*** Input flag to choose inof of which playback range to return. Set(true)for queued range. Reset(false) for current range.

***aContextData*** Optional opaque data that will be passed back to the user with the command response This method can leave with one of the following error codes OsclErrInvalidState if invoked in the incorrect state

**Returns**

A unique command id for asynchronous completion

#### 3.1.3.11 virtual PVCommandId PVPlayerInterface::GetPlaybackRate (int32 & *aRate*, PVMFTimebase *& *aTimebase*, const OsclAny * *aContextData* = NULL) [pure virtual]

This function retrieves the current playback rate setting. If the playback rate is set as a millipercent of "real-time" playback rate, then aRate will be filled in with the milliperecent value when this command completes successfully. If the playback rate is set by an outside timebase, aRate will be set to 0 and aTimebase pointer will point to the PVMFTimebase being used when the command completes successfully. This function can be called when pvPlayer is in PVP_STATE_PREPARED, PVP_STATE_STARTED, or PVP_STATE_-PAUSED state. This command request is asynchronous. PVCommandStatusObserver's CommandCompleted() callback handler will be called when this command request completes.

**Parameters**

***aRate*** A reference to an integer which will be filled in with the current playback rate expressed as millipercent of "real-time" playback rate. If an outside timebase is being used, aRate would be set to 0.

***aTimebase*** Reference to an PVMFTimebase pointer which will be valid if an outside timebase is being used for the playback clock.

***aContextData*** Optional opaque data that will be passed back to the user with the command response This method can leave with one of the following error codes

**Returns**

A unique command id for asynchronous completion

#### 3.1.3.12 virtual PVCommandId PVPlayerInterface::GetPVPlayerState (PVPlayerState & *aState*, const OsclAny * *aContextData* = NULL) [pure virtual]

This function returns the current state of pvPlayer. Application may use this info for updating display or determine if the pvPlayer is ready for the next request. This command request is asynchronous. PVCommandStatusObserver's CommandCompleted() callback handler will be called when this command request completes.

**Parameters**

***aState*** A reference to a PVPlayerState. Upon successful completion of this command, it will contain the current state of pvPlayer.

***aContextData*** Optional opaque data that will be passed back to the user with the command response

**Returns**

A unique command id for asynchronous completion

#### 3.1.3.13 virtual PVMFStatus PVPlayerInterface::GetPVPlayerStateSync (PVPlayerState & *aState*) [pure virtual]

This function returns the current state of pvPlayer as a synchronous command. Application may use this info for updating display or determine if the pvPlayer is ready for the next request.

**Parameters**

***aState*** A reference to a PVPlayerState. Upon successful completion of this command, it will contain the current state of pvPlayer.

**Returns**

Status indicating whether the command succeeded or not.

#### 3.1.3.14 static OSCL_IMPORT_REF void PVPlayerInterface::GetSDKInfo (PVSDKInfo & *aSDKInfo*) [static]

Returns SDK version information about pvPlayer.

**Parameters**

***aSDKInfo*** A reference to a PVSDKInfo structure which contains product name, supported hardware platform, supported software platform, version, part number, and PV UUID. These fields will contain info .for the currently instantiated pvPlayer engine when this function returns success.

#### 3.1.3.15 virtual PVCommandId PVPlayerInterface::GetSDKModuleInfo (PVSDKModuleInfo & *aSDKModuleInfo*, const OsclAny * *aContextData* = NULL) [pure virtual]

Returns information about all modules currently used by pvPlayer SDK. This command request is asynchronous. PVCommandStatusObserver's CommandCompleted() callback handler will be called when this command request completes.

**Parameters**

***aSDKModuleInfo*** A reference to a PVSDKModuleInfo structure which contains the number of modules currently used by pvPlayer SDK and the PV UUID and description string for each module. The PV UUID and description string for modules will be returned in one string buffer allocated by the client. If the string buffer is not large enough to hold the all the module's information, the information will be written up to the length of the buffer and truncated.

***aContextData*** Optional opaque data that will be passed back to the user with the command response This method can leave with one of the following error codes OsclErrNoMemory if the SDK failed to allocate memory during this operation

**Returns**

A unique command ID for asynchronous completion

#### 3.1.3.16 virtual PVCommandId PVPlayerInterface::Init (const OsclAny * *aContextData* = NULL) [pure virtual]

This function switches pvPlayer from PVP_STATE_IDLE state to the PVP_STATE_INITIALIZED state. During the transition, pvPlayer is in the PVP_STATE_INITIALIZING transitional state and the data source is being initialized to obtain metadata and track information of the source media. If initialization fails, pvPlayer will revert to PVP_STATE_IDLE state and the data source will be closed. The Command should only be called in PVP_STATE_IDLE. This command request is asynchronous. PVCommandStatusObserver's CommandCompleted() callback handler will be called when this command request completes.

**Parameters**

***aContextData*** Optional opaque data that will be passed back to the user with the command response This method can leave with one of the following error codes OsclErrInvalidState if invoked in the incorrect state OsclErrNoMemory if the SDK failed to allocate memory during this operation

**Returns**

A unique command id for asynchronous completion

#### 3.1.3.17 virtual PVCommandId PVPlayerInterface::Pause (const OsclAny * *aContextData* = NULL) [pure virtual]

This function pauses the currently ongoing playback. pvPlayer must be in PVP_STATE_STARTED state to call this function. When pause successfully completes, pvPlayer will be in PVP_STATE_PAUSED state. This command request is asynchronous. PVCommandStatusObserver's CommandCompleted() callback handler will be called when this command request completes.

**Parameters**

***aContextData*** Optional opaque data that will be passed back to the user with the command response This method can leave with one of the following error codes OsclErrInvalidState if invoked in the incorrect state OsclErrNoMemory if the SDK failed to allocate memory during this operation

**Returns**

A unique command id for asynchronous completion

#### 3.1.3.18 virtual PVCommandId PVPlayerInterface::Prepare (const OsclAny * *aContextData* = NULL) [pure virtual]

This functions prepares pvPlayer for playback. pvPlayer connects the data source with the data sinks and starts the data source to queue the media data for playback(e.g. for 3GPP streaming, fills the jitter buffer). pvPlayer also checks to make sure each component needed for playback is ready and capable. When successful, pvPlayer will be in PVP_STATE_PREPARED state, The command should be called only in PVP_STATE_INITIALISED. This command request is asynchronous. PVCommandStatusObserver's CommandCompleted() callback handler will be called when this command request completes.

**Parameters**

***aContextData*** Optional opaque data that will be passed back to the user with the command response This method can leave with one of the following error codes OsclErrInvalidState if invoked in the incorrect state OsclErrNoMemory if the SDK failed to allocate memory during this operation

**Returns**

A unique command id for asynchronous completion

#### 3.1.3.19 virtual PVCommandId PVPlayerInterface::QueryInterface (const PVUuid & *aUuid*, PVInterface *&& *aInterfacePtr*, const OsclAny * *aContextData* = NULL) [pure virtual]

This API is to allow for extensibility of the pvPlayer interface. It allows a caller to ask for an instance of a particular interface object to be returned. The mechanism is analogous to the COM IUnknown method. The interfaces are identified with an interface ID that is a UUID as in DCE and a pointer to the interface object is returned if it is supported. Otherwise the returned pointer is NULL. This command request is asynchronous. PVCommandStatusObserver's CommandCompleted() callback handler will be called when this command request completes.

**Parameters**

***aUuid*** The UUID of the desired interface

***aInterfacePtr*** A reference to the output pointer to the desired interface

***aContextData*** Optional opaque data that will be passed back to the user with the command response

**Exceptions**

***This*** method can leave with one of the following error codes OsclErrNotSupported if the specified interface UUID is not supported

**Returns**

A unique command ID for asynchronous completion

#### 3.1.3.20 virtual PVCommandId PVPlayerInterface::ReleaseMetadataValues (Oscl_Vector< PvmiKvp, OsclMemAllocator > & *aValueList*, const OsclAny * *aContextData* = NULL, uint32 *aClipIndex* = 0) [pure virtual]

The function makes a request to release the metadata value(s) specified by the passed in metadata value list. This command request is asynchronous. PVCommandStatusObserver's CommandCompleted() callback handler will be called when this command request completes.If a GetMetaDataValues were called in PVP_-STATE_INITIALIZED state, then corresponding ReleaseMetaDataValues must be called before Reset. If a GetMetaDataValues were called in PVP_STATE_PREPARED, PVP_STATE_STARTED, PVP_STATE_-PAUSED states, then corresponding ReleaseMetaDataValues must be called before Stop.

**Parameters**

***aValueList*** Reference to a vector of KVP to place the specified metadata values

***aContextData*** Optional opaque data that will be passed back to the user with the command response

***aClipIndex,:*** an optional parameter for use with playlists to select the clip of interest. This method can leave with one of the following error codes OsclErrInvalidState if invoked in the incorrect state OsclErrNoMemory if the SDK failed to allocate memory during this operation

**Returns**

A unique command id for asynchronous completion

#### 3.1.3.21 virtual PVCommandId PVPlayerInterface::RemoveDataSink (PVPlayerDataSink & *aDataSink*, const OsclAny * *aContextData* = NULL) [pure virtual]

This function may be used to close and unbind a data sink that has been previously added. This function must be called when pvPlayer is in PVP_STATE_INITIALIZED state. If the data sink is in use for playback, Stop must be called first to stop the playback and free the data sink. This command request is asynchronous. PVCommandStatusObserver's CommandCompleted() callback handler will be called when this command request completes.

**Parameters**

***aDataSink*** Reference to the data sink to be removed

***aContextData*** Optional opaque data that will be passed back to the user with the command response This method can leave with one of the following error codes OsclErrBadHandle if the passed in sink parameter is invalid OsclErrInvalidState if invoked in the incorrect state OsclErrNoMemory if the SDK failed to allocate memory during this operation

**Returns**

A unique command id for asynchronous completion

#### 3.1.3.22 virtual PVCommandId PVPlayerInterface::RemoveDataSource (PVPlayerDataSource & *aDataSource*, const OsclAny * *aContextData* = NULL) [pure virtual]

This function may be used to close and unbind a data source that has been previously added. This function must be called when pvPlayer is in PVP_STATE_IDLE state. If the data source has already been initialized, Reset must be called first. This command request is asynchronous. PVCommandStatusObserver's CommandCompleted() callback handler will be called when this command request completes.

**Parameters**

***aDataSource*** Reference to the data source to be removed.

***aContextData*** Optional opaque data that will be passed back to the user with the command response This method can leave with one of the following error codes OsclErrBadHandle if the passed in sink parameter is invalid OsclErrInvalidState if invoked in the incorrect state OsclErrNoMemory if the SDK failed to allocate memory during this operation

**Returns**

A unique command id for asynchronous completion

#### 3.1.3.23 virtual PVCommandId PVPlayerInterface::RemoveLogAppender (const char * *aTag*, OsclSharedPtr< PVLoggerAppender > & *aAppender*, const OsclAny * *aContextData* = NULL) [pure virtual]

Allows a logging appender to be removed from the logger tree at the point specified by the input tag. If the input tag is NULL then the appender will be removed from locations in the tree. This command request is asynchronous. PVCommandStatusObserver's CommandCompleted() callback handler will be called when this command request completes.

**Parameters**

***aTag*** Specifies the logger tree tag where the appender should be removed. Can be NULL to remove at all locations.

***aAppender*** The log appender to remove.

***aContextData*** Optional opaque data that will be passed back to the user with the command response

**Exceptions**

***This*** method can leave with one of the following error codes OsclErrNoMemory if the SDK failed to allocate memory during this operation

**Returns**

A unique command ID for asynchronous completion

#### 3.1.3.24 virtual PVCommandId PVPlayerInterface::Reset (const OsclAny * *aContextData* = NULL) [pure virtual]

This function cleans up resources used for playback to transition pvPlayer to PVP_STATE_IDLE state. While processing this command, pvPlayer is in the PVP_STATE_RESETTING state. If any data sinks are still referenced by pvPlayer when this function is called, the data sinks will be closed and removed from pvPlayer during the Reset. If already in PVP_STATE_IDLE state, then nothing will occur. This command request is asynchronous. PVCommandStatusObserver's CommandCompleted() callback handler will be called when this command request completes.

**Parameters**

***aContextData*** Optional opaque data that will be passed back to the user with the command response This method can leave with one of the following error codes OsclErrNoMemory if the SDK failed to allocate memory during this operation

**Returns**

A unique command id for asynchronous completion

#### 3.1.3.25 virtual PVCommandId PVPlayerInterface::Resume (const OsclAny * *aContextData* = NULL) [pure virtual]

This function resumes the currently paused playback. pvPlayer must be in PVP_STATE_PAUSED state to call this function. When resume successfully completes, pvPlayer will be in PVP_STATE_STARTED state. This command request is asynchronous. PVCommandStatusObserver's CommandCompleted() callback handler will be called when this command request completes.

**Parameters**

***aContextData*** Optional opaque data that will be passed back to the user with the command response This method can leave with one of the following error codes OsclErrInvalidState if invoked in the incorrect state OsclErrNoMemory if the SDK failed to allocate memory during this operation

**Returns**

A unique command id for asynchronous completion

#### 3.1.3.26 virtual PVCommandId PVPlayerInterface::SetLogAppender (const char * *aTag*, OsclSharedPtr< PVLoggerAppender > & *aAppender*, const OsclAny * *aContextData* = NULL) [pure virtual]

Allows a logging appender to be attached at some point in the logger tag tree. The location in the tag tree is specified by the input tag string. A single appender can be attached multiple times in the tree, but it may result in duplicate copies of log messages if the appender is not attached in disjoint portions of the tree. A logging appender is responsible for actually writing the log message to its final location (e.g., memory, file, network, etc). This API can be called anytime after creation of pvPlayer. This command request is asynchronous. PVCommandStatusObserver's CommandCompleted() callback handler will be called when this command request completes.

**Parameters**

***aTag*** Specifies the logger tree tag where the appender should be attached.

***aAppender*** The log appender to attach.

***aContextData*** Optional opaque data that will be passed back to the user with the command response

**Exceptions**

***This*** method can leave with one of the following error codes OsclErrNoMemory if the SDK failed to allocate memory during this operation

**Returns**

A unique command ID for asynchronous completion

#### 3.1.3.27 virtual PVCommandId PVPlayerInterface::SetLogLevel (const char * *aTag*, int32 *aLevel*, bool *aSetSubtree* = false, const OsclAny * *aContextData* = NULL) [pure virtual]

Allows the logging level to be set for the logging node specified by the tag. A larger log level will result in more messages being logged. A message will only be logged if its level is LESS THAN or equal to the current log level. The aSetSubtree flag will allow an entire subtree, with the specified tag as the root, to be reset to the specified value. This command request is asynchronous. PVCommandStatusObserver's CommandCompleted() callback handler will be called when this command request completes.

**Parameters**

***aTag*** Specifies the logger tree tag where the log level should be set.

***aLevel*** Specifies the log level to set.

***aSetSubtree*** Specifies whether the entire subtree with aTag as the root should be reset to the log level.

***aContextData*** Optional opaque data that will be passed back to the user with the command response

**Exceptions**

***This*** method can leave with one of the following error codes OsclErrNoMemory if the SDK failed to allocate memory during this operation

**Returns**

A unique command ID for asynchronous completion

#### 3.1.3.28 virtual PVCommandId PVPlayerInterface::SetPlaybackRange (PVPPlaybackPosition *aBeginPos*, PVPPlaybackPosition *aEndPos*, bool *aQueueRange*, const OsclAny * *aContextData* = NULL, bool *aSkipToRequestedPosition* = true, bool *aSeekToSyncPoint* = true) [pure virtual]

This function sets the begin and end positions for the new playback range or changes the end position of the current playback range. This function must be called when pvPlayer is in PVP_STATE_INITIALIZED, PVP_STATE_PREPARED, PVP_STATE_STARTED, or PVP_STATE_PAUSED state. The specified positions must be between beginning of clip and clip duration. The units of position is specified in the passed-in parameter PVPPlaybackPosition. If either of the positions is indeterminate, use the indeterminate flag in PVPPlaybackPosition structure. The queued playback range can be done using aQueueRange flag which is Not Supported as of now by PV-SDK. This function will overwrite any previous playback range info. The only exception is the changing of end position for the current playback range during playback. Command if called in player state as PVP_STATE_INITIALISED or PVP_STATE_PAUSED, will complete in one Engine AO run without actually changing the position. The change in position will come into affect when Prepare or Resume respectively is called on Engine by the app. If reposition request is not honored by the source node during Prepare or Resume, engine will continue to complete Prepare or Resume but will send an informational event "PVMFInfoChangePlaybackPositionNotSupported" to the app informing that the SetPlaybackRange request could not be honored. This command request is asynchronous. PVCommandStatusObserver's CommandCompleted() callback handler will be called when this command request completes.

**Parameters**

***aBeginPos*** Beginning position for the new playback range

***aEndPos*** Ending position for the new playback range.

***aQueueRange*** Input flag to tell pvPlayer to queue the new playback range (Set/true) or use the new playback range right away (Reset/false)

***aContextData*** Optional opaque data that will be passed back to the user with the command response

***aSkipToRequestedPosition*** Boolean value to indicate whether or not the display should start exactly from the requested position Default is set to 'true'.

***aSeekToSyncPoint*** Boolean value to indicate whether or not the source should seek to a sync point. true - Means that the decoder will be fed from a key-frame thereby promising that there are no artefacts false - Means that the decoder will be fed the nearest frame from the requested seek point, which can be a non-sync frame, with a possibility of causing artefacts Default is set to 'true' This method can leave with one of the following error codes OsclErrInvalidState if invoked in the incorrect state

**Returns**

A unique command id for asynchronous completion

#### 3.1.3.29 virtual PVCommandId PVPlayerInterface::SetPlaybackRate (int32 *aRate*, PVMFTimebase * *aTimebase* = NULL, const OsclAny * *aContextData* = NULL) [pure virtual]

This function allows the setting of the playback rate. The playback rate can be set as millipercent of "real-time" playback rate. For example, 100000 means 1X "real-time", 400000 means 4X, 25000 means 0.25X, and -100000 means 1X backward. The playback rate can also be modified by specifying the timebase to use for the playback clock. This is accomplished by setting the aRate parameter to 0 and passing in a pointer to an PVMFTimebase. This function can be called when pvPlayer is in PVP_STATE_PREPARED, PVP_STATE_STARTED, or PVP_STATE_PAUSED state. Changing to or from an outside timebase is only

allowed in PVP_STATE_PREPARED. Command if called in player state PVP_STATE_PAUSED with a direction change, will complete in one Engine AO run without actually changing the direction. The change in direction will come into affect when Resume is called on Engine by the app. If the request is not honored by the source node during Resume, engine will continue to complete Resume but will send an informational event "PVMFInfoChangePlaybackPositionNotSupported" to the app informing that the SetPlaybackRate request could not be honored. This command request is asynchronous. PVCommandStatusObserver's CommandCompleted() callback handler will be called when this command request completes.

**Parameters**

***aRate*** The playback rate specified as millipercent of "real-time". A millipercent is 1/1000 of a percent. So 2X = 200% of realtime is 200,000 millipercent. The motivation is to povide precision with an integer parameter. Negative rates specify backward playback. The valid range of absolute value of playback rates will be limited to the minimum and maximum returned by GetPlaybackMinMaxRate().

***aTimebase*** Reference to an PVMFTimebase which will be used to drive the playback clock. aRate must be set to 0, 1X, or -1X to use the timebase.

***aContextData*** Optional opaque data that will be passed back to the user with the command response This method can leave with one of the following error codes OsclErrArgument if rate or timebase is invalid

**Returns**

A unique command id for asynchronous completion

#### 3.1.3.30 virtual PVCommandId PVPlayerInterface::Start (const OsclAny * *aContextData* = NULL) [pure virtual]

This function kicks off the actual playback. Media data are sent out from the data source to the data sink(s). pvPlayer will transition to PVP_STATE_STARTED state after playback starts successfully. The command should be called only in PVP_STATE_PREPARED. This command request is asynchronous. PVCommandStatusObserver's CommandCompleted() callback handler will be called when this command request completes.

**Parameters**

***aContextData*** Optional opaque data that will be passed back to the user with the command response This method can leave with one of the following error codes OsclErrInvalidState if invoked in the incorrect state OsclErrNoMemory if the SDK failed to allocate memory during this operation

**Returns**

A unique command id for asynchronous completion

#### 3.1.3.31 virtual PVCommandId PVPlayerInterface::Stop (const OsclAny * *aContextData* = NULL) [pure virtual]

This function stops the current playback and transitions pvPlayer to the PVP_STATE_INITIALIZED state. During the transition, data transmission from data source to all data sinks are terminated. Also all connections between data source and data sinks are torn down. This command request is asynchronous. PVCommandStatusObserver's CommandCompleted() callback handler will be called when this command request completes.

**Parameters**

***aContextData*** Optional opaque data that will be passed back to the user with the command response This method can leave with one of the following error codes OsclErrInvalidState if invoked in the incorrect state OsclErrNoMemory if the SDK failed to allocate memory during this operation

**Returns**

A unique command id for asynchronous completion

#### 3.1.3.32 virtual PVCommandId PVPlayerInterface::UpdateDataSource (PVPlayerDataSource & *aDataSource*, const OsclAny * *aContextData* = NULL) [pure virtual]

This function allows extending or updating a track list during playback. Changes can be applied prior to beginning initialization for any track.

**Parameters**

***aDataSource*** aDataSource contains an updated version of the data source previously provided in AddDataSource command. The updated data source can contain modifications to existing clips in the list, or new clips added to the end of the list. This API cannot be used to update data for clips that have already been initialized, or to delete clips from the clip list.

***aContextData*** Optional opaque data that will be passed back to the user with the command response This method can leave with one of the following error codes OsclErrNoMemory if the SDK failed to allocate memory during this operation

**Returns**

A unique command id for asynchronous completion

The documentation for this class was generated from the following file:

- pv_player_interface.h

# Chapter 4

# File Documentation

## 4.1 pv_player_interface.h File Reference

```
#include "oscl_base.h"
#include "oscl_string.h"
#include "oscl_vector.h"
#include "oscl_mem.h"
#include "pvlogger.h"
#include "pvmf_return_codes.h"
#include "pv_engine_types.h"
#include "pv_player_types.h"
#include "oscl_string_containers.h"
#include "pvmf_format_type.h"
#include "pvmi_kvp.h"
#include "pvmf_media_clock.h"
```

### Data Structures

- class PVPlayerInterface

# Index