PV Author Engine

Build Version: OPENCORE_20090120

January 20, 2009

Contents

Chapter 1

pvauthor_engine Hierarchical Index

1.1 pvauthor_engine Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Chapter 2

pvauthor_engine Data Structure Index

2.1 pvauthor_engine Data Structures

Here are the data structures with brief descriptions:

Chapter 3

pvauthor_engine File Index

3.1 pvauthor_engine File List

Here is a list of all files with brief descriptions:

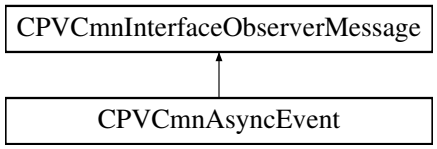
Chapter 4

pvauthor_engine Data Structure Documentation

4.1 CPVCmnAsyncEvent Class Reference

`#include <pv_common_types.h>`

Inheritance diagram for CPVCmnAsyncEvent::



Public Methods

- CPVCmnAsyncEvent (TPVCmnEventType aEventType, TPVCmnExclusivePtr aExclusivePtr, const uint8 *aLocalBuffer=NULL, uint32 aLocalBufSize=0, TPVCmnResponseType aResponseType=NULL)
- ~CPVCmnAsyncEvent ()
- TPVCmnEventType GetEventType () const
- void GetEventData (TPVCmnExclusivePtr &aPtr) const
- uint8 * GetLocalBuffer ()

Protected Attributes

- TPVCmnEventType iEventType
- TPVCmnExclusivePtr iExclusivePtr
- uint8 iLocalBuffer [PV_COMMON_ASYNC_EVENT_LOCAL_BUF_SIZE]

4.1.1 Detailed Description

CPVCmnAsyncEvent Class

CPVCmnAsyncEvent is the base class used to pass unsolicited error and informational indications to the user. Additional information can be tagged based on the specific event

4.1.2 Constructor & Destructor Documentation

4.1.2.1 CPVCmnAsyncEvent::CPVCmnAsyncEvent (TPVCmnEventType *aEventType*, TPVCmnExclusivePtr *aExclusivePtr*, const uint8 * *aLocalBuffer* = NULL, uint32 *aLocalBufSize* = 0, TPVCmnResponseType *aResponseType* = NULL) [inline]

4.1.2.2 CPVCmnAsyncEvent::~CPVCmnAsyncEvent () [inline]

4.1.3 Member Function Documentation

4.1.3.1 void CPVCmnAsyncEvent::GetEventData (TPVCmnExclusivePtr & *aPtr*) const [inline]

Returns:
Returns the opaque data asociated with the event.

4.1.3.2 TPVCmnEventType CPVCmnAsyncEvent::GetEventType () const [inline]

Returns:
Returns the Event type that has been received

4.1.3.3 uint8* CPVCmnAsyncEvent::GetLocalBuffer () [inline]

Returns:
Returns the local data asociated with the event.

4.1.4 Field Documentation

4.1.4.1 TPVCmnEventType CPVCmnAsyncEvent::iEventType [protected]

4.1.4.2 TPVCmnExclusivePtr CPVCmnAsyncEvent::iExclusivePtr [protected]

4.1.4.3 uint8 CPVCmnAsyncEvent::iLocalBuffer[PV_COMMON_ASYNC_EVENT_LOCAL_BUF_SIZE] [protected]

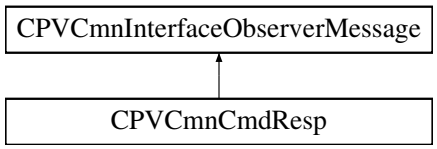
The documentation for this class was generated from the following file:

- pv_common_types.h

4.2 CPVCmnCmdResp Class Reference

`#include <pv_common_types.h>`

Inheritance diagram for CPVCmnCmdResp::



Public Methods

- CPVCmnCmdResp (TPVCmnCommandType aType, TPVCmnCommandId aId, void *aContext, TPVCmnCommandStatus aStatus, void *aResponseData=NULL, int aResponseDataSize=0, TPVCmnResponseType aResponseType=NULL)
- TPVCmnCommandType GetCmdType () const
- TPVCmnCommandId GetCmdId () const
- void * GetContext () const
- TPVCmnCommandStatus GetCmdStatus () const
- void * GetResponseData () const
- int GetResponseDataSize () const

Protected Attributes

- TPVCmnCommandType iCmdType
- TPVCmnCommandId iCmdId
- void * iContext
- TPVCmnCommandStatus iStatus
- void * iResponseData
- int iResponseDataSize

4.2.1 Constructor & Destructor Documentation

4.2.1.1 CPVCmnCmdResp::CPVCmnCmdResp (TPVCmnCommandType *aType*, TPVCmnCommandId *aId*, void * *aContext*, TPVCmnCommandStatus *aStatus*, void * *aResponseData* = NULL, int *aResponseDataSize* = 0, TPVCmnResponseType *aResponseType* = NULL) [inline]

Constructor for CPVCmnCmdResp

4.2.2 Member Function Documentation

4.2.2.1 TPVCmnCommandId CPVCmnCmdResp::GetCmdId () const [inline]

Returns:
Returns the unique ID associated with a command of this type.

4.2.2.2 TPVCmnCommandStatus CPVCmnCmdResp::GetCmdStatus () const [inline]

Returns:
Returns the completion status of the command

4.2.2.3 TPVCmnCommandType CPVCmnCmdResp::GetCmdType () const [inline]

Returns:
Returns the command type that is being completed.

4.2.2.4 void* CPVCmnCmdResp::GetContext () const [inline]

Returns:
Returns the opaque data that was passed in with the command.

4.2.2.5 void* CPVCmnCmdResp::GetResponseData () const [inline]

Returns:
Returns additional data asociated with the command. This is to be interpreted based on the command type and the return status

4.2.2.6 int CPVCmnCmdResp::GetResponseDataSize () const [inline]

4.2.3 Field Documentation

4.2.3.1 TPVCmnCommandId CPVCmnCmdResp::iCmdId [protected]

4.2.3.2 TPVCmnCommandType CPVCmnCmdResp::iCmdType [protected]

4.2.3.3 void* CPVCmnCmdResp::iContext [protected]

4.2.3.4 void* CPVCmnCmdResp::iResponseData [protected]

4.2.3.5 int CPVCmnCmdResp::iResponseDataSize [protected]

4.2.3.6 TPVCmnCommandStatus CPVCmnCmdResp::iStatus [protected]

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

- pv_common_types.h

4.3 CPVCmnInterfaceCmdMessage Class Reference

`#include <pv_interface_cmd_message.h>`

Public Methods

- CPVCmnInterfaceCmdMessage (int aType, OsclAny *aContextData)
- CPVCmnInterfaceCmdMessage ()
- virtual ~CPVCmnInterfaceCmdMessage ()
- PVCommandId GetCommandId ()
- int GetType ()
- OsclAny * GetContextData ()
- int compare (CPVCmnInterfaceCmdMessage *a, CPVCmnInterfaceCmdMessage *b) const
- int32 GetPriority () const
- void SetId (PVCommandId aId)

Protected Attributes

- PVCommandId iId
- int iType
- int32 iPriority
- OsclAny * iContextData

Friends

- class PVInterfaceProxy
- int32 operator< (const CPVCmnInterfaceCmdMessage &a, const CPVCmnInterfaceCmdMessage &b)

4.3.1 Detailed Description

CPVInterfaceCmdMessage Class

CPVInterfaceCmdMessage is the interface to the pv2way SDK, which allows initialization, control, and termination of a two-way terminal. The application is expected to contain and maintain a pointer to the CPV2WayInterface instance at all times that a call is active. The CPV2WayFactory factory class is to be used to create and delete instances of this class

4.3.2 Constructor & Destructor Documentation

4.3.2.1 CPVCmnInterfaceCmdMessage::CPVCmnInterfaceCmdMessage (int *aType*, OsclAny * *aContextData*) [inline]

4.3.2.2 CPVCmnInterfaceCmdMessage::CPVCmnInterfaceCmdMessage () [inline]

4.3.2.3 virtual CPVCmnInterfaceCmdMessage::~CPVCmnInterfaceCmdMessage () [inline, virtual]

4.3.3 Member Function Documentation

4.3.3.1 int CPVCmnInterfaceCmdMessage::compare (CPVCmnInterfaceCmdMessage * *a*, CPVCmnInterfaceCmdMessage * *b*) const [inline]

The algorithm used in OsclPriorityQueue needs a compare function that returns true when A's priority is less than B's

Returns:
true if A's priority is less than B's, else false

4.3.3.2 PVCommandId CPVCmnInterfaceCmdMessage::GetCommandId () [inline]

4.3.3.3 OsclAny* CPVCmnInterfaceCmdMessage::GetContextData () [inline]

4.3.3.4 int32 CPVCmnInterfaceCmdMessage::GetPriority () const [inline]

4.3.3.5 int CPVCmnInterfaceCmdMessage::GetType () [inline]

4.3.3.6 void CPVCmnInterfaceCmdMessage::SetId (PVCommandId *aId*) [inline]

4.3.4 Friends And Related Function Documentation

4.3.4.1 int32 operator< (const CPVCmnInterfaceCmdMessage & *a*, const CPVCmnInterfaceCmdMessage & *b*) [friend]

4.3.4.2 friend class PVInterfaceProxy [friend]

4.3.5 Field Documentation

4.3.5.1 OsclAny* CPVCmnInterfaceCmdMessage::iContextData [protected]

4.3.5.2 PVCommandId CPVCmnInterfaceCmdMessage::iId [protected]

4.3.5.3 int32 CPVCmnInterfaceCmdMessage::iPriority [protected]

4.3.5.4 int CPVCmnInterfaceCmdMessage::iType [protected]

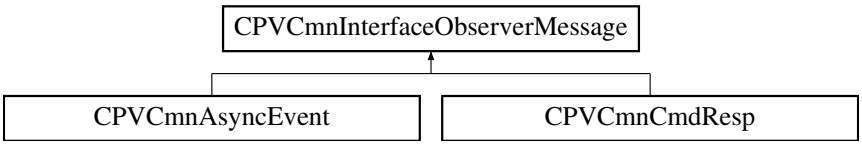
The documentation for this class was generated from the following file:

- pv_interface_cmd_message.h

4.4 CPVCmnInterfaceObserverMessage Class Reference

`#include <pv_common_types.h>`

Inheritance diagram for CPVCmnInterfaceObserverMessage::



Public Methods

- CPVCmnInterfaceObserverMessage ()
- CPVCmnInterfaceObserverMessage (TPVCmnResponseType aResponseType)
- virtual ~CPVCmnInterfaceObserverMessage ()
- TPVCmnResponseType GetResponseType () const
- virtual int GetPriority () const

Data Fields

- TPVCmnResponseType iResponseType
- int iPriority
- int iOrder

4.4.1 Detailed Description

CPVCmnInterfaceObserverMessage Class

CPVCmnInterfaceObserverMessage is the interface to the pv2way SDK, which allows initialization, control, and termination of a two-way terminal. The application is expected to contain and maintain a pointer to the CPV2WayInterface instance at all times that a call is active. The CPV2WayFactory factory class is to be used to create and delete instances of this class

4.4.2 Constructor & Destructor Documentation

4.4.2.1 CPVCmnInterfaceObserverMessage::CPVCmnInterfaceObserverMessage () [inline]

4.4.2.2 CPVCmnInterfaceObserverMessage::CPVCmnInterfaceObserverMessage (TPVCmnResponseType *aResponseType*) [inline]

4.4.2.3 virtual CPVCmnInterfaceObserverMessage::~CPVCmnInterfaceObserverMessage () [inline, virtual]

4.4.3 Member Function Documentation

4.4.3.1 virtual int CPVCmnInterfaceObserverMessage::GetPriority () const [inline, virtual]

4.4.3.2 TPVCmnResponseType CPVCmnInterfaceObserverMessage::GetResponseType () const [inline]

4.4.4 Field Documentation

4.4.4.1 int CPVCmnInterfaceObserverMessage::iOrder

4.4.4.2 int CPVCmnInterfaceObserverMessage::iPriority

4.4.4.3 TPVCmnResponseType CPVCmnInterfaceObserverMessage::iResponseType

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

- pv_common_types.h

4.5 CPVCmnInterfaceObserverMessageCompare Class Reference

`#include <pv_common_types.h>`

Public Methods

- int compare (CPVCmnInterfaceObserverMessage *a, CPVCmnInterfaceObserverMessage *b) const

4.5.1 Member Function Documentation

4.5.1.1 int CPVCmnInterfaceObserverMessageCompare::compare (CPVCmnInterfaceObserverMessage * *a*, CPVCmnInterfaceObserverMessage * *b*) const [inline]

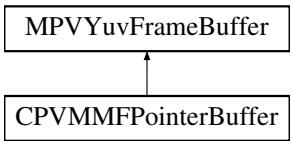
The documentation for this class was generated from the following file:

- pv_common_types.h

4.6 CPVMMFPointerBuffer Class Reference

`#include <pv_plugin_interfaces.h>`

Inheritance diagram for CPVMMFPointerBuffer::



Public Methods

- ~CPVMMFPointerBuffer ()
- virtual TDes8 & Data ()
- virtual const TDesC8 & Data () const
- virtual void SetRequestSizeL (TInt aSize)
- virtual TUint BufferSize () const
- void SetData (TUint8 *aData, TInt aLength)
- void SetFrameSize (const TSize &size)
- virtual TSize GetFrameSize () const

Static Public Methods

- CPVMMFPointerBuffer * NewL ()

4.6.1 Constructor & Destructor Documentation

4.6.1.1 CPVMMFPointerBuffer::~CPVMMFPointerBuffer () [inline]

4.6.2 Member Function Documentation

4.6.2.1 virtual TUint CPVMMFPointerBuffer::BufferSize () const [inline, virtual]

4.6.2.2 virtual const TDesC8& CPVMMFPointerBuffer::Data () const [inline, virtual]

4.6.2.3 virtual TDes8& CPVMMFPointerBuffer::Data () [inline, virtual]

4.6.2.4 virtual TSize CPVMMFPointerBuffer::GetFrameSize () const [inline, virtual]

Implements MPVYuvFrameBuffer.

4.6.2.5 CPVMMFPointerBuffer* CPVMMFPointerBuffer::NewL () [inline, static]

4.6.2.6 void CPVMMFPointerBuffer::SetData (TUint8 * *aData*, TInt *aLength*) [inline]

4.6.2.7 void CPVMMFPointerBuffer::SetFrameSize (const TSize & *size*) [inline]

4.6.2.8 virtual void CPVMMFPointerBuffer::SetRequestSizeL (TInt *aSize*) [inline, virtual]

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

- pv_plugin_interfaces.h

4.7 MPVAudioInput Class Reference

`#include <pv_plugin_interfaces.h>`

Public Methods

- virtual IMPORT_C void SetFormatL (const TDesC8 &aFormat, const TDesC8 &aFmtSpecific, TInt &aMaxRequestSize)=0
- virtual IMPORT_C void SetConfigL (const TDesC8 &aSampleRate, const TDesC8 &aChannels)=0
- virtual IMPORT_C void CancelCommand ()=0
- virtual IMPORT_C TInt Reset ()=0

4.7.1 Detailed Description

MPVAudioInput Class

MPVAudioInput cab be implemented by any audio data source that needs to work with PV SDKs.

4.7.2 Member Function Documentation

4.7.2.1 virtual IMPORT_C void MPVAudioInput::CancelCommand () [pure virtual]

4.7.2.2 virtual IMPORT_C TInt MPVAudioInput::Reset () [pure virtual]

4.7.2.3 virtual IMPORT_C void MPVAudioInput::SetConfigL (const TDesC8 & *aSampleRate*, const TDesC8 & *aChannels*) [pure virtual]

4.7.2.4 virtual IMPORT_C void MPVAudioInput::SetFormatL (const TDesC8 & *aFormat*, const TDesC8 & *aFmtSpecific*, TInt & *aMaxRequestSize*) [pure virtual]

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

- pv_plugin_interfaces.h

4.8 MPVAudioOutput Class Reference

`#include <pv_plugin_interfaces.h>`

Public Methods

- virtual IMPORT_C void SetFormatL (const TDesC8 &aFormat)=0
- virtual IMPORT_C void SetConfigL (const TDesC8 &aSampleRate, const TDesC8 &aChannels)=0
- virtual IMPORT_C void CancelCommand ()=0
- virtual IMPORT_C TInt Reset ()=0

4.8.1 Detailed Description

MPVAudioOutput Class

MPVAudioOutput can be implemented by any audio data sink that needs to work with PV SDKs.

4.8.2 Member Function Documentation

4.8.2.1 virtual IMPORT_C void MPVAudioOutput::CancelCommand () [pure virtual]

4.8.2.2 virtual IMPORT_C TInt MPVAudioOutput::Reset () [pure virtual]

4.8.2.3 virtual IMPORT_C void MPVAudioOutput::SetConfigL (const TDesC8 & *aSampleRate*, const TDesC8 & *aChannels*) [pure virtual]

4.8.2.4 virtual IMPORT_C void MPVAudioOutput::SetFormatL (const TDesC8 & *aFormat*) [pure virtual]

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

- pv_plugin_interfaces.h

4.9 MPVCmnCmdStatusObserver Class Reference

`#include <pv_common_types.h>`

Public Methods

- virtual ~MPVCmnCmdStatusObserver ()
- virtual void CommandCompletedL (const CPVCmnCmdResp &aResponse)=0

4.9.1 Constructor & Destructor Documentation

4.9.1.1 virtual MPVCmnCmdStatusObserver::~MPVCmnCmdStatusObserver () [inline, virtual]

4.9.2 Member Function Documentation

4.9.2.1 virtual void MPVCmnCmdStatusObserver::CommandCompletedL (const CPVCmnCmdResp & *aResponse*) [pure virtual]

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

- pv_common_types.h

4.10 MPVCmnErrorEventObserver Class Reference

`#include <pv_common_types.h>`

Public Methods

- virtual ~MPVCmnErrorEventObserver ()
- virtual void HandleErrorEventL (const CPVCmnAsyncErrorEvent &aEvent)=0

4.10.1 Constructor & Destructor Documentation

4.10.1.1 virtual MPVCmnErrorEventObserver::~MPVCmnErrorEventObserver () [inline, virtual]

4.10.2 Member Function Documentation

4.10.2.1 virtual void MPVCmnErrorEventObserver::HandleErrorEventL (const CPVCmnAsyncErrorEvent & *aEvent*) [pure virtual]

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

- pv_common_types.h

4.11 MPVCmnInfoEventObserver Class Reference

`#include <pv_common_types.h>`

Public Methods

- virtual ~MPVCmnInfoEventObserver ()
- virtual void HandleInformationalEventL (const CPVCmnAsyncInfoEvent &aEvent)=0

4.11.1 Constructor & Destructor Documentation

4.11.1.1 virtual MPVCmnInfoEventObserver::~MPVCmnInfoEventObserver () [inline, virtual]

4.11.2 Member Function Documentation

4.11.2.1 virtual void MPVCmnInfoEventObserver::HandleInformationalEventL (const CPVCmnAsyncInfoEvent & *aEvent*) [pure virtual]

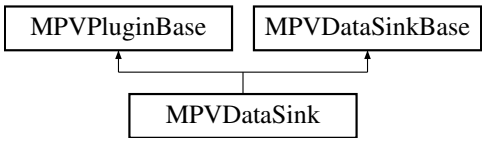
The documentation for this class was generated from the following file:

- pv_common_types.h

4.12 MPVDataSink Class Reference

`#include <pv_plugin_interfaces.h>`

Inheritance diagram for MPVDataSink::



Public Methods

- MPVDataSink (TUid aSinkType)
- virtual ~MPVDataSink ()

4.12.1 Detailed Description

MPVDataSink Class

PV extension to MDataSource that supports basic PV requirements like exposing capabilities, configuration interfaces etc

4.12.2 Constructor & Destructor Documentation

4.12.2.1 MPVDataSink::MPVDataSink (TUid *aSinkType*) [inline]

4.12.2.2 virtual MPVDataSink::~MPVDataSink () [inline, virtual]

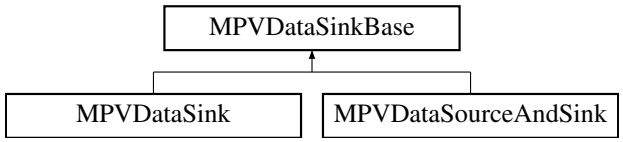
The documentation for this class was generated from the following file:

- pv_plugin_interfaces.h

4.13 MPVDataSinkBase Class Reference

`#include <pv_plugin_interfaces.h>`

Inheritance diagram for MPVDataSinkBase::



Public Methods

- MPVDataSinkBase (TUid aType)
- virtual ~MPVDataSinkBase ()
- virtual void EmptyBufferL (CMMFBuffer *aBuffer, MPVDataSourceBase *aSupplier, TMediaId)=0
- virtual void BufferFilledL (CMMFBuffer *aBuffer)=0
- virtual TBool CanCreateSinkBuffer ()=0
- virtual CMMFBuffer * CreateSinkBufferL (TMediaId, TBool &)
- virtual TInt SinkThreadLogon (MAsyncEventHandler &)
- virtual void SinkThreadLogoff ()
- virtual TInt SinkPrimeL ()
- virtual TInt SinkPlayL ()
- virtual TInt SinkPauseL ()
- virtual TInt SinkStopL ()

4.13.1 Detailed Description

MPVDataSourceBase Class

Base class for data sinks

4.13.2 Constructor & Destructor Documentation

4.13.2.1 MPVDataSinkBase::MPVDataSinkBase (TUid *aType*) `[inline]`

4.13.2.2 virtual MPVDataSinkBase::~MPVDataSinkBase () `[inline, virtual]`

4.13.3 Member Function Documentation

4.13.3.1 virtual void MPVDataSinkBase::BufferFilledL (CMMFBuffer * *aBuffer*) `[pure virtual]`

Method called by a data source to pass back an filled buffer to the sink

This is a pure virtual function that each derived class must implement. This method is used as the callback when the data sink actively requests a supplier ie a data source to fill a buffer by calling the data sources FillBufferL. When the data sink gets this callback it knows that the buffer has been filled and is ready to be emptied

Parameters:

aBuffer The buffer that has been filled by a data source and is now available for processing

4.13.3.2 virtual TBool MPVDataSinkBase::CanCreateSinkBuffer () [pure virtual]

Method to indicate whether the data sink can create a buffer.

This is a pure virtual function that each derived class must implement.

Returns:

ETrue if the data sink can create a buffer else EFalse

4.13.3.3 virtual CMMFBuffer* MPVDataSinkBase::CreateSinkBufferL (TMediaId, TBool &) [inline, virtual]

Returns a buffer created by the data sink

This is a pure virtual function that each derived class must implement.

Parameters:

aMediaId This identifies the type of media eg audio or video and the stream id. This parameter is required in cases where the source can supply data of more than one media type and/or multiple strams of data.

aReference This must be written to by the method to indicate whether the created buffer is a 'reference' buffer. A 'reference' buffer is a buffer that is owned by the sink and should be used in preference to the source buffer provided the source buffer is also not a reference buffer.

Returns:

The created buffer

4.13.3.4 virtual void MPVDataSinkBase::EmptyBufferL (CMMFBuffer * *aBuffer*, MPVDataSourceBase * *aSupplier*, TMediaId) [pure virtual]

Method called by a MDataSource to request the data sink to empty aBuffer of data.

This is a pure virtual function that each derived class must implement. This method is used when a data sink is passively waiting for requests from a supplier ie a data source to empty a buffer. The data sink must call the BufferEmptiedL member on aSupplier when it has emptied the buffer of it's data - the data sink can either make this callback synchronously or asynchronously.

Parameters:

aBuffer The full buffer that needs emptying of it's data

aSupplier The data source that supplied the data. The data sink needs this to make the BufferEmptiedL callback on aSupplier to indicate to the data source that the data sink has finished with the buffer.

aMediaId This identifies the type of media eg audio or video and the stream id. This parameter is required in cases where the source can supply data of more than one media type and/or multiple strams of data

4.13.3.5 virtual TInt MPVDataSinkBase::SinkPauseL () [inline, virtual]

Method to 'pause' the data sink

This is a virtual function that a derrived data sink can implement if any data sink specific action is required to 'pause'

4.13.3.6 virtual TInt MPVDataSinkBase::SinkPlayL () [inline, virtual]

Method to 'play' the data sink

This is a virtual function that a derrived data sink can implement if any data sink specific action is required prior to 'playing'ie the start of data transfer

4.13.3.7 virtual TInt MPVDataSinkBase::SinkPrimeL () [inline, virtual]

Method to 'prime' the data sink

This is a virtual function that a derrived data sink can implement if any data sink specific 'priming' is required

4.13.3.8 virtual TInt MPVDataSinkBase::SinkStopL () [inline, virtual]

Method to 'stop' the data sink

This is a virtual function that a derrived data sink can implement if any data sink specific action is required to 'stop'

4.13.3.9 virtual void MPVDataSinkBase::SinkThreadLogoff () [inline, virtual]

Method to 'logoff' the data sink from the same thread that sink consumes data in.

This method may be required as the thread that the data sink is deleted in may not be the same thread that the data transfer took place in. Therefore any thread specific releasing of resources needs to be performed in the SinkThreadLogoff rather than in the destructor

This is a virtual function that a derrived data sink can implement if any thread specific releasing of resources is required.

4.13.3.10 virtual TInt MPVDataSinkBase::SinkThreadLogon (MAsyncEventHandler &) [inline, virtual]

Method to 'logon' the data sink to the same thread that sink will be consuming data in.

This method may be required as the thread that the data sink was created in is not always the same thread that the data transfer will take place in. Therefore any thread specific initialisation needs to be performed in the SinkThreadLogon rather than in the creation of the data sink.

This is a virtual function that a derrived data sink can implement if any thread specific initialisation is required and/or the data sink can create any asynchronous events.

Parameters:

aEventHandler This is an MAsyncEventHandler to handle asynchronous events that occur during the transfer of multimedia data. The event handler must be in the same thread as the data transfer thread - hence the reason it is passed in the SinkThreadLogon as opposed to say the constructor.

Returns:
KErrNone if successful, otherwise a system wide error code.

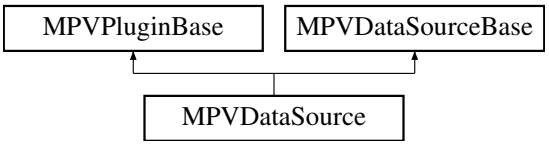
The documentation for this class was generated from the following file:

- pv_plugin_interfaces.h

4.14 MPVDataSource Class Reference

`#include <pv_plugin_interfaces.h>`

Inheritance diagram for MPVDataSource::



Public Methods

- MPVDataSource (TUid aSourceType)
- virtual ~MPVDataSource ()

4.14.1 Detailed Description

MPVDataSource Class

PV extension to MDataSource that supports basic PV requirements like exposing capabilities, configuration interfaces etc

4.14.2 Constructor & Destructor Documentation

4.14.2.1 MPVDataSource::MPVDataSource (TUid *aSourceType*) [inline]

4.14.2.2 virtual MPVDataSource::~MPVDataSource () [inline, virtual]

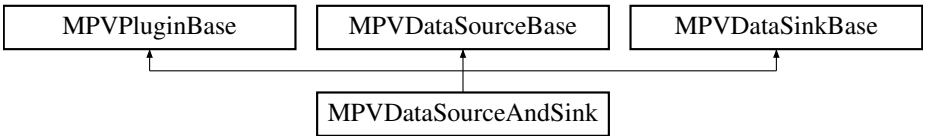
The documentation for this class was generated from the following file:

- pv_plugin_interfaces.h

4.15 MPVDataSourceAndSink Class Reference

`#include <pv_plugin_interfaces.h>`

Inheritance diagram for MPVDataSourceAndSink::



Public Methods

- MPVDataSourceAndSink (TUid aSourceType, TUid aSinkType)
- virtual ~MPVDataSourceAndSink ()

4.15.1 Detailed Description

Supports the basic functionality of both PV Data Sources and Sinks.

4.15.2 Constructor & Destructor Documentation

4.15.2.1 MPVDataSourceAndSink::MPVDataSourceAndSink (TUid *aSourceType*, TUid *aSinkType*) [inline]

4.15.2.2 virtual MPVDataSourceAndSink::~MPVDataSourceAndSink () [inline, virtual]

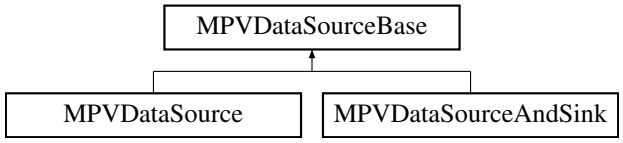
The documentation for this class was generated from the following file:

- pv_plugin_interfaces.h

4.16 MPVDataSourceBase Class Reference

`#include <pv_plugin_interfaces.h>`

Inheritance diagram for MPVDataSourceBase::



Public Methods

- MPVDataSourceBase (TUid aType)
- virtual ~MPVDataSourceBase ()
- virtual void FillBufferL (CMMFBuffer *aBuffer, MPVDataSinkBase *aConsumer, TMediaId)=0
- virtual void BufferEmptiedL (CMMFBuffer *aBuffer)=0
- virtual TBool CanCreateSourceBuffer ()=0
- virtual CMMFBuffer * CreateSourceBufferL (TMediaId, TBool &)
- virtual CMMFBuffer * CreateSourceBufferL (TMediaId, CMMFBuffer &, TBool &)
- virtual TInt SourceThreadLogon (MAsyncEventHandler &)
- virtual void SourceThreadLogoff ()
- virtual TInt SourcePrimeL ()
- virtual TInt SourcePlayL ()
- virtual TInt SourcePauseL ()
- virtual TInt SourceStopL ()

4.16.1 Detailed Description

MPVDataSourceBase Class

Base class for data sources

4.16.2 Constructor & Destructor Documentation

4.16.2.1 MPVDataSourceBase::MPVDataSourceBase (TUid *aType*) [inline]

4.16.2.2 virtual MPVDataSourceBase::~MPVDataSourceBase () [inline, virtual]

4.16.3 Member Function Documentation

4.16.3.1 virtual void MPVDataSourceBase::BufferEmptiedL (CMMFBuffer * *aBuffer*) [pure virtual]

Method called by a data sink to pass back an emptied buffer to the source

This is a pure virtual function that each derived class must implement. This method is used as the callback when the data source actively requests a consumer ie a data sink to empty a buffer by calling the data sinks EmptyBufferL. When the data source gets this callback it knows that the buffer has been emptied and can be reused

Parameters:

aBuffer The buffer that has been emptied by a data sink and is now available for reuse

4.16.3.2 virtual TBool MPVDataSourceBase::CanCreateSourceBuffer () [pure virtual]

Method to indicate whether the data source can create a buffer.

This is a pure virtual function that each derived class must implement.

Returns:

ETrue if the data source can create a buffer else EFalse

4.16.3.3 virtual CMMFBuffer* MPVDataSourceBase::CreateSourceBufferL (TMediaId, CMMFBuffer &, TBool &) [inline, virtual]

Returns a buffer created by the data source

This is a virtual function that a derived class can implement. This can be used in preference to the above CreateSourceBufferL method in cases where the source buffer creation has a dependancy on the sink buffer

Parameters:

aMediaId This identifies the type of media eg audio or video and the stream id. This parameter is required in cases where the source can supply data of more than one media type and/or multiple strams of data eg a multimedia file

aSinkBuffer The sink buffer the nature of which may influence the creation of the source buffer

aReference This must be written to by the method to indicate whether the created buffer is a 'reference' buffer. A 'reference' buffer is a buffer that is owned by the source and should be used in preference to the sink buffer provided the sink buffer is not a reference buffer

Returns:

The created buffer

4.16.3.4 virtual CMMFBuffer* MPVDataSourceBase::CreateSourceBufferL (TMediaId, TBool &) [inline, virtual]

Returns a buffer created by the data source

This is a pure virtual function that each derived class must implement.

Parameters:

aMediaId This identifies the type of media eg audio or video and the stream id. This parameter is required in cases where the source can supply data of more than one media type and/or multiple strams of data eg a multimedia file

aReference This must be written to by the method to indicate whether the created buffer is a 'reference' buffer. A 'reference' buffer is a buffer that is owned by the source and should be used in preference to the sink buffer provided the sink buffer is also not a reference buffer

Returns:

The created buffer

4.16.3.5 virtual void MPVDataSourceBase::FillBufferL (CMMFBuffer * *aBuffer*, MPVDataSinkBase * *aConsumer*, TMediaId) [pure virtual]

Method called by a MDataSink to request the data source to fill aBuffer with data.

This is a pure virtual function that each derived class must implement. This method is used when a data source is passively waiting for requests from a consumer ie a data sink to fill a buffer. The data source must call the BufferFilledL member on aConsumer when it has filled the buffer with data - the data source can either make this callback synchronously or asynchronously.

Parameters:

aBuffer The buffer that needs filling with data

aConsumer The data sink that consumes the data. The data source needs this to make the BufferFilledL callback on aConsumer when the data source has completed filling the aBuffer.

aMediaId This identifies the type of media eg audio or video and the stream id. This parameter is required in cases where the source can supply data of more than one media type and/or multiple strams of data eg a multimedia file

4.16.3.6 virtual TInt MPVDataSourceBase::SourcePauseL () [inline, virtual]

Method to 'pause' the data source

This is a virtual function that a derrived data source can implement if any data source specific action is required to 'pause'

4.16.3.7 virtual TInt MPVDataSourceBase::SourcePlayL () [inline, virtual]

Method to 'play' the data source

This is a virtual function that a derrived data source can implement if any data source specific action is required prior to 'playing'ie the start of data transfer

4.16.3.8 virtual TInt MPVDataSourceBase::SourcePrimeL () [inline, virtual]

Method to 'prime' the data source

This is a virtual function that a derrived data source can implement if any data source specific 'priming' is required

4.16.3.9 virtual TInt MPVDataSourceBase::SourceStopL () [inline, virtual]

Method to 'stop' the data source

This is a virtual function that a derrived data source can implement if any data source specific action is required to 'stop'

4.16.3.10 virtual void MPVDataSourceBase::SourceThreadLogoff () [inline, virtual]

Method to 'logoff' the data source from the same thread that source supplies data in.

This method may be required as the thread that the data source is deleted in may not be the same thread that the data transfer took place in. Therefore any thread specific releasing of resources needs to be performed in the SourceThreadLogoff rather than in the destructor

This is a virtual function that a derrived data source can implement if any thread specific releasing of resources is required.

4.16.3.11 virtual TInt MPVDataSourceBase::SourceThreadLogon (MAsyncEventHandler &) [inline, virtual]

Method to 'logon' the data source to the same thread that source will be supplying data in.

This method may be required as the thread that the data source was created in is not always the same thread that the data transfer will take place in. Therefore any thread specific initialisation needs to be performed in the SourceThreadLogon rather than in the creation of the data source.

This is a virtual function that a derrived data source can implement if any thread specific initialisation is required and/or the data source can create any asynchronous events.

Parameters:

aEventHandler This is an MAsyncEventHandler to handle asynchronous events that occur during the transfer of multimedia data. The event handler must be in the same thread as the data transfer thread - hence the reason it is passed in the SourceThreadLogon as opposed to say the constructor.

Returns:

KErrNone if successful, otherwise a system wide error code.

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

- pv_plugin_interfaces.h

4.17 MPVDevSoundAudioInput Class Reference

`#include <pv_plugin_interfaces.h>`

Public Methods

- virtual IMPORT_C void SetPrioritySettings (const TMMFPrioritySettings &aSettings)=0
- virtual IMPORT_C void SetInputFormatL (const TDesC8 &aFormat, MPVDataSourceBase *)=0
- virtual IMPORT_C TPVAudioOutputSwitch OutputSwitch ()=0
- virtual IMPORT_C TBool FillAmrBuffersToEnd ()=0

4.17.1 Member Function Documentation

4.17.1.1 virtual IMPORT_C TBool MPVDevSoundAudioInput::FillAmrBuffersToEnd () [pure virtual]

4.17.1.2 virtual IMPORT_C TPVAudioOutputSwitch MPVDevSoundAudioInput::OutputSwitch () [pure virtual]

4.17.1.3 virtual IMPORT_C void MPVDevSoundAudioInput::SetInputFormatL (const TDesC8 & *aFormat*, MPVDataSourceBase *) [pure virtual]

4.17.1.4 virtual IMPORT_C void MPVDevSoundAudioInput::SetPrioritySettings (const TMMFPrioritySettings & *aSettings*) [pure virtual]

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

- pv_plugin_interfaces.h

4.18 MPVDevSoundAudioOutput Class Reference

`#include <pv_plugin_interfaces.h>`

Public Methods

- virtual IMPORT_C void SetPrioritySettings (const TMMFPrioritySettings &aSettings)=0
- virtual IMPORT_C void ConcealErrorForNextBuffer ()=0
- virtual IMPORT_C void SetOutputFormatL (const TDesC8 &aFormat, const TDesC8 &aFmtSpecific, MPVDataSinkBase *aConsumer, TInt &aMaxRequestSize)=0
- virtual IMPORT_C TPVAudioOutputSwitch OutputSwitch ()=0
- virtual IMPORT_C TBool FillAmrBuffersToEnd ()=0

4.18.1 Member Function Documentation

4.18.1.1 virtual IMPORT_C void MPVDevSoundAudioOutput::ConcealErrorForNextBuffer () [pure virtual]

4.18.1.2 virtual IMPORT_C TBool MPVDevSoundAudioOutput::FillAmrBuffersToEnd () [pure virtual]

4.18.1.3 virtual IMPORT_C TPVAudioOutputSwitch MPVDevSoundAudioOutput::OutputSwitch () [pure virtual]

4.18.1.4 virtual IMPORT_C void MPVDevSoundAudioOutput::SetOutputFormatL (const TDesC8 & *aFormat*, const TDesC8 & *aFmtSpecific*, MPVDataSinkBase * *aConsumer*, TInt & *aMaxRequestSize*) [pure virtual]

4.18.1.5 virtual IMPORT_C void MPVDevSoundAudioOutput::SetPrioritySettings (const TMMFPrioritySettings & *aSettings*) [pure virtual]

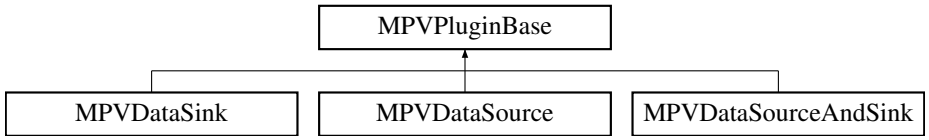
The documentation for this class was generated from the following file:

- pv_plugin_interfaces.h

4.19 MPVPluginBase Class Reference

`#include <pv_plugin_interfaces.h>`

Inheritance diagram for MPVPluginBase::



Public Methods

- virtual ~MPVPluginBase ()
- virtual IMPORT_C const RArray< TPVMIMEType * > & GetMultimediaTypesL () const=0
- virtual IMPORT_C void QueryUUID (const TPVMIMEType &aMimeType, RArray< TPVUuid > &aUuids, bool aExactUuidsOnly=false)=0
- virtual IMPORT_C void QueryInterface (const TPVUuid &aUuid, TPVInterfacePtr &aInterfacePtr)=0

4.19.1 Detailed Description

MPVPluginBase Class

Base class for all supported plugins

4.19.2 Constructor & Destructor Documentation

4.19.2.1 virtual MPVPluginBase::~MPVPluginBase () `[inline, virtual]`

4.19.3 Member Function Documentation

4.19.3.1 virtual IMPORT_C const RArray<TPVMIMEType *>& MPVPluginBase::GetMultimediaTypesL () `[pure virtual]`

This API returns multimedias type supported by the data source/sink - Audio, Video, Data etc. Each supported type is indicated by a MIME type structure.

Returns:

Multimedia types supported by the data source/sink. The reference is valid until the MPVPluginBase derived object is destroyed.

4.19.3.2 virtual IMPORT_C void MPVPluginBase::QueryInterface (const TPVUuid & *aUuid*, TPVInterfacePtr & *aInterfacePtr*) `[pure virtual]`

This API is to allow for extensibility of the plugin 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. TBD: Define the UIID, InterfacePtr structures

Parameters:

aUuid The UUID of the desired interface

aInterfacePtr The output pointer to the desired interface

4.19.3.3 virtual IMPORT_C void MPVPluginBase::QueryUUID (const TPVMIMEType & *aMimeType*, RArray< TPVUuid > & *aUuids*, bool *aExactUuidsOnly* = false) [pure virtual]

This API is to allow for extensibility of the plugin interface. It allows a caller to ask for all UUIDs associated with a particular MIME type. If interfaces of the requested MIME type are found within the plugin, they are added to the UUIDs array.

Also added to the UUIDs array will be all interfaces which have the requested MIME type as a base MIME type. This functionality can be turned off.

Parameters:

aMimeType The MIME type of the desired interfaces

aUuids An array to hold the discovered UUIDs

aExactUuidsOnly Turns on/off the retrival of UUIDs with aMimeType as a base type

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

- pv_plugin_interfaces.h

4.20 MPVVideoInput Class Reference

`#include <pv_plugin_interfaces.h>`

Public Methods

- virtual IMPORT_C void SetFormatL (const TDesC8 &aFormat)=0
- virtual IMPORT_C void SetFrameRateL (TReal32 aFrameRate)=0
- virtual IMPORT_C void SetVideoFrameSizeL (const TSize &aSize)=0
- virtual IMPORT_C void GetVideoFrameSizeL (TSize &aSize) const=0

4.20.1 Detailed Description

MPVVideoInput Class

MPVVideoInput can be implemented by any video data source that needs to work with PV SDKs.

4.20.2 Member Function Documentation

4.20.2.1 virtual IMPORT_C void MPVVideoInput::GetVideoFrameSizeL (TSize & *aSize*) const [pure virtual]

Get the video frame size

Parameters:
aSize The video frame size, in pixels

Exceptions:
Can leave with one of the system wide error codes

4.20.2.2 virtual IMPORT_C void MPVVideoInput::SetFormatL (const TDesC8 & *aFormat*) [pure virtual]

Set the video frame format. This must be from the list of supported formats.

Parameters:
aFormat The mime string describing the video frame format.

Exceptions:
Can leave with one of the system wide error codes

4.20.2.3 virtual IMPORT_C void MPVVideoInput::SetFrameRateL (TReal32 *aFrameRate*) [pure virtual]

Set the video frame rate. This must be within the range of supported frame rates for the current frame size.

Parameters:
aFrameRate The video frame rate to set.

Exceptions:

Can leave with one of the system wide error codes

4.20.2.4 virtual IMPORT_C void MPVVideoInput::SetVideoFrameSizeL (const TSize & *aSize*) [pure virtual]

Set the video frame size

Parameters:

aSize The video frame size, in pixels

Exceptions:

Can leave with one of the system wide error codes

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

- pv_plugin_interfaces.h

4.21 MPVVideoOutput Class Reference

`#include <pv_plugin_interfaces.h>`

Public Methods

- virtual IMPORT_C void SetFormatL (const TDesC8 &aFormat)=0
- virtual IMPORT_C void SetVideoFrameSizeL (const TSize &aSize)=0
- virtual IMPORT_C void GetVideoFrameSizeL (TSize &aSize) const=0

4.21.1 Detailed Description

MPVVideoOutput Class

MPVVideoOutput can be implemented by any video data sink that needs to work with PV SDKs.

4.21.2 Member Function Documentation

4.21.2.1 virtual IMPORT_C void MPVVideoOutput::GetVideoFrameSizeL (TSize & *aSize*) const [pure virtual]

Get the video frame size

Parameters:
aSize The video frame size, in pixels

Exceptions:
Can leave with one of the system wide error codes

4.21.2.2 virtual IMPORT_C void MPVVideoOutput::SetFormatL (const TDesC8 & *aFormat*) [pure virtual]

Set the video frame format. This must be from the list of supported formats.

Parameters:
aFormat A mime string describing the video frame format.

Exceptions:
Can leave with one of the system wide error codes

4.21.2.3 virtual IMPORT_C void MPVVideoOutput::SetVideoFrameSizeL (const TSize & *aSize*) [pure virtual]

Set the video frame size

Parameters:
aSize The video frame size, in pixels

Exceptions:

Can leave with one of the system wide error codes

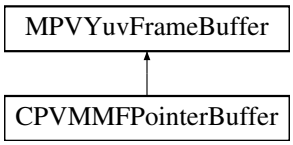
The documentation for this class was generated from the following file:

- pv_plugin_interfaces.h

4.22 MPVYuvFrameBuffer Class Reference

`#include <pv_plugin_interfaces.h>`

Inheritance diagram for MPVYuvFrameBuffer::



Public Methods

- virtual ~MPVYuvFrameBuffer ()
- virtual TSize GetFrameSize () const=0

4.22.1 Constructor & Destructor Documentation

4.22.1.1 virtual MPVYuvFrameBuffer::~MPVYuvFrameBuffer () [inline, virtual]

4.22.2 Member Function Documentation

4.22.2.1 virtual TSize MPVYuvFrameBuffer::GetFrameSize () [pure virtual]

Implemented in CPVMMFPointerBuffer.

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

- pv_plugin_interfaces.h

4.23 PVAsyncErrorEvent Class Reference

`#include <pv_engine_observer_message.h>`

Public Methods

- PVAsyncErrorEvent (PVEventType aEventType, PVExclusivePtr aEventData=NULL, uint8 *aLocalBuffer=NULL, int32 aLocalBufferSize=0)
- PVAsyncErrorEvent (PVEventType aEventType, OsclAny *aContext, PVInterface *aEventExtInterface, PVExclusivePtr aEventData=NULL, uint8 *aLocalBuffer=NULL, int32 aLocalBufferSize=0)
- ~PVAsyncErrorEvent ()
- PVResponseType GetResponseType () const
- PVEventType GetEventType () const
- void GetEventData (PVExclusivePtr &aPtr) const

4.23.1 Detailed Description

PVAsyncErrorEvent Class

PVAsyncErrorEvent is used to pass unsolicited error indications to the user. Additional information can be tagged based on the specific event

4.23.2 Constructor & Destructor Documentation

4.23.2.1 PVAsyncErrorEvent::PVAsyncErrorEvent (PVEventType *aEventType*, PVExclusivePtr *aEventData* = NULL, uint8 * *aLocalBuffer* = NULL, int32 *aLocalBufferSize* = 0) [inline]

Constructor for PVAsyncErrorEvent

4.23.2.2 PVAsyncErrorEvent::PVAsyncErrorEvent (PVEventType *aEventType*, OsclAny * *aContext*, PVInterface * *aEventExtInterface*, PVExclusivePtr *aEventData* = NULL, uint8 * *aLocalBuffer* = NULL, int32 *aLocalBufferSize* = 0) [inline]

Constructor with context and event extension interface

4.23.2.3 PVAsyncErrorEvent::~PVAsyncErrorEvent () [inline]

Destructor

4.23.3 Member Function Documentation

4.23.3.1 void PVAsyncErrorEvent::GetEventData (PVExclusivePtr & *aPtr*) const [inline]

Returns:
Returns the opaque data asociated with the event.

4.23.3.2 PVEventType PVAsyncErrorEvent::GetEventType () const [inline]

Returns:
Returns the Event type that has been received

4.23.3.3 PVResponseType PVAsyncErrorEvent::GetResponseType () const [inline]

WILL BE DEPRECATED SINCE IT IS NOT BEING USED. CURRENTLY RETURNING 0.

Returns:
Returns the type of Response we get

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

- pv_engine_observer_message.h

4.24 PVAsyncInformationalEvent Class Reference

`#include <pv_engine_observer_message.h>`

Public Methods

- PVAsyncInformationalEvent (PVEventType aEventType, PVExclusivePtr aEventData=NULL, uint8 *aLocalBuffer=NULL, int32 aLocalBufferSize=0)
- PVAsyncInformationalEvent (PVEventType aEventType, OsclAny *aContext, PVInterface *aEventExtInterface, PVExclusivePtr aEventData=NULL, uint8 *aLocalBuffer=NULL, int32 aLocalBufferSize=0)
- ~PVAsyncInformationalEvent ()
- PVResponseType GetResponseType () const
- PVEventType GetEventType () const
- void GetEventData (PVExclusivePtr &aPtr) const

4.24.1 Detailed Description

PVAsyncInformationalEvent Class

PVAsyncInformationalEvent is used to pass unsolicited informational indications to the user. Additional information can be tagged based on the specific event

4.24.2 Constructor & Destructor Documentation

4.24.2.1 PVAsyncInformationalEvent::PVAsyncInformationalEvent (PVEventType *aEventType*, PVExclusivePtr *aEventData* = NULL, uint8 * *aLocalBuffer* = NULL, int32 *aLocalBufferSize* = 0) [inline]

Constructor for PVAsyncInformationalEvent

4.24.2.2 PVAsyncInformationalEvent::PVAsyncInformationalEvent (PVEventType *aEventType*, OsclAny * *aContext*, PVInterface * *aEventExtInterface*, PVExclusivePtr *aEventData* = NULL, uint8 * *aLocalBuffer* = NULL, int32 *aLocalBufferSize* = 0) [inline]

Constructor with context and event extension interface

4.24.2.3 PVAsyncInformationalEvent::~PVAsyncInformationalEvent () [inline]

Destructor

4.24.3 Member Function Documentation

4.24.3.1 void PVAsyncInformationalEvent::GetEventData (PVExclusivePtr & *aPtr*) const [inline]

Returns:
Returns the opaque data asociated with the event.

4.24.3.2 PVEventType PVAsyncInformationalEvent::GetEventType () const [inline]

Returns:
Returns the Event type that has been received

4.24.3.3 PVResponseType PVAsyncInformationalEvent::GetResponseType () const [inline]

WILL BE DEPRECATED SINCE IT IS NOT BEING USED. CURRENTLY RETURNING 0.

Returns:
Returns the type of Response we get

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

- pv_engine_observer_message.h

4.25 PVAuthorEngineFactory Class Reference

`#include <pvauthorenginefactory.h>`

Static Public Methods

- OSCL_IMPORT_REF PVAuthorEngineInterface * CreateAuthor (PVCommandStatusObserver *aCmdStatusObserver, PVErrorEventObserver *aErrorEventObserver, PVInformationalEventObserver *aInfoEventObserver)
- OSCL_IMPORT_REF bool DeleteAuthor (PVAuthorEngineInterface *aAuthor)

4.25.1 Detailed Description

PVAuthorEngineFactory Class

PVAuthorEngineFactory class is a singleton class which instantiates and provides access to pvAuthor engine. It returns an PVAuthorEngineInterface reference, the interface class of the pvAuthor SDK.

The application is expected to contain and maintain a pointer to the PVAuthorEngineInterface instance at all time that pvAuthor engine is active.

4.25.2 Member Function Documentation

4.25.2.1 OSCL_IMPORT_REF PVAuthorEngineInterface* PVAuthorEngineFactory::CreateAuthor (PVCommandStatusObserver * *aCmdStatusObserver*, PVErrorEventObserver * *aErrorEventObserver*, PVInformationalEventObserver * *aInfoEventObserver*) [static]

Creates an instance of a pvAuthor engine. If the creation fails, this function will leave.

Parameters:

aCmdStatusObserver The observer for command status

aErrorEventObserver The observer for unsolicited error events

aInfoEventObserver The observer for unsolicited informational events

Returns:

A pointer to an author or leaves if instantiation fails

4.25.2.2 OSCL_IMPORT_REF bool PVAuthorEngineFactory::DeleteAuthor (PVAuthorEngineInterface * *aAuthor*) [static]

This function allows the application to delete an instance of a pvAuthor and reclaim all allocated resources. An author can be deleted only in the idle state. An attempt to delete an author in any other state will fail and return false.

Parameters:

aAuthor The author to be deleted.

Returns:

A status code indicating success or failure.

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

- pvauthorenginefactory.h

4.26 PVAuthorEngineInterface Class Reference

`#include <pvauthorengineinterface.h>`

Public Methods

- virtual ~PVAuthorEngineInterface ()
- virtual PVCommandId SetLogAppender (const char *aTag, PVLoggerAppender &aAppender, const OsclAny *aContextData=NULL)=0
- virtual PVCommandId RemoveLogAppender (const char *aTag, 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 Open (const OsclAny *aContextData=NULL)=0
- virtual PVCommandId Close (const OsclAny *aContextData=NULL)=0
- virtual PVCommandId AddDataSource (const PVMFNodeInterface &aDataSource, const OsclAny *aContextData=NULL)=0
- virtual PVCommandId RemoveDataSource (const PVMFNodeInterface &aDataSource, const OsclAny *aContextData=NULL)=0
- virtual PVCommandId SelectComposer (const PvmfMimeString &aComposerType, PVInterface *&aConfigInterface, const OsclAny *aContextData=NULL)=0
- virtual PVCommandId SelectComposer (const PVUuid &aComposerUuid, PVInterface *&aConfigInterface, const OsclAny *aContextData=NULL)=0
- virtual PVCommandId AddMediaTrack (const PVMFNodeInterface &aDataSource, const PvmfMimeString &aEncoderType, const OsclAny *aComposer, PVInterface *&aConfigInterface, const OsclAny *aContextData=NULL)=0
- virtual PVCommandId AddMediaTrack (const PVMFNodeInterface &aDataSource, const PVUuid &aEncoderUuid, const OsclAny *aComposer, PVInterface *&aConfigInterface, const OsclAny *aContextData=NULL)=0
- virtual PVCommandId AddDataSink (const PVMFNodeInterface &aDataSink, const OsclAny *aComposer, const OsclAny *aContextData=NULL)=0
- virtual PVCommandId RemoveDataSink (const PVMFNodeInterface &aDataSink, const OsclAny *aContextData=NULL)=0
- virtual PVCommandId Init (const OsclAny *aContextData=NULL)=0
- virtual PVCommandId Reset (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 PVAEState GetPVAuthorState ()=0
- virtual PVCommandId QueryUUID (const PvmfMimeString &aMimeType, Oscl_Vector< PVUuid, OsclMemAllocator > &aUuids, bool aExactUuidsOnly=false, const OsclAny *aContextData=NULL)=0
- virtual PVCommandId QueryInterface (const PVUuid &aUuid, PVInterface *&aInterfacePtr, const OsclAny *aContextData=NULL)=0
- virtual PVCommandId GetSDKModuleInfo (PVSDKModuleInfo &aSDKModuleInfo, const OsclAny *aContextData=NULL)=0
- virtual PVCommandId CancelAllCommands (const OsclAny *aContextData=NULL)=0

Static Public Methods

- OSCL_IMPORT_REF void GetSDKInfo (PVSDKInfo &aSDKInfo)

4.26.1 Detailed Description

PVAuthorEngineInterface

4.26.2 Constructor & Destructor Documentation

4.26.2.1 virtual PVAuthorEngineInterface::~PVAuthorEngineInterface () [inline, virtual]

Destructor.

4.26.3 Member Function Documentation

4.26.3.1 virtual PVCommandId PVAuthorEngineInterface::AddDataSink (const PVMFNodeInterface & *aDataSink*, const OsclAny * *aComposer*, const OsclAny * *aContextData* = NULL) [pure virtual]

Adds a media sink where output data from the specified composer will be written to. Currently this API does not cause any action as it is not relevant.

This command is valid only when pvAuthor Engine is in PVAE_STATE_OPENED state. The referenced composer must be previously selected.

This command does not change the pvAuthor Engine engine state.

Parameters:

aDataSink Reference to the data sink to be used

aComposer Opaque data identifying the composer to which the data sink will connect to.

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

Returns:

A unique command id for asynchronous completion

4.26.3.2 virtual PVCommandId PVAuthorEngineInterface::AddDataSource (const PVMFNodeInterface & *aDataSource*, const OsclAny * *aContextData* = NULL) [pure virtual]

Adds a media source to be used as input to an authoring session.

This command is valid only when pvAuthor Engine is in PVAE_STATE_OPENED state. This command does not change the pvAuthor Engine engine state.

Parameters:

aDataSource Reference to the data source

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

Returns:

Unique command ID to identify this command in command response

4.26.3.3 virtual PVCommandId PVAuthorEngineInterface::AddMediaTrack (const PVMFNodeInterface & *aDataSource*, const PVUuid & *aEncoderUuid*, const OsclAny * *aComposer*, PVInterface *& *aConfigInterface*, const OsclAny * *aContextData* = NULL) [pure virtual]

Add a media track to the specified composer.

The source data of this media track will come from the specified data source. pvAuthor engine will encoder of the specified Uuid to encode the source data. A media track will be added to the specified composer, and encoded data will be written to the composer during the authoring session.

A configuration object for the selected composer will be saved to the PVInterface pointer provided in aConfigInterface parameter. User should call queryInterface to query for the configuration interfaces supported by the encoder. Before calling Reset(), user must call removeRef on the PVInterface object to remove its reference to the object.

This command is valid only when pvAuthor Engine is in PVAE_STATE_OPENED state. The referenced data source and composer must be already added before this method is called. This command does not change the pvAuthor Engine engine state.

Parameters:

aDataSource Data source node to provide input data

aEncoderUuid Uuid of encoder to encode the source data

aComposer Opaque data to identify the composer in which a media track will be added.

aConfigInterface Pointer to configuration object for the selected encoder will be saved to this parameter upon completion of this call

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

Returns:

A unique command id for asynchronous completion

4.26.3.4 virtual PVCommandId PVAuthorEngineInterface::AddMediaTrack (const PVMFNodeInterface & *aDataSource*, const PvmfMimeString & *aEncoderType*, const OsclAny * *aComposer*, PVInterface *& *aConfigInterface*, const OsclAny * *aContextData* = NULL) [pure virtual]

Add a media track to the specified composer.

The source data of this media track will come from the specified data source. pvAuthor engine will select the most suitable available encoder of the specified type. A media track will be added to the specified composer, and encoded data will be written to the composer during the authoring session.

A configuration object for the selected composer will be saved to the PVInterface pointer provided in aConfigInterface parameter. User should call queryInterface to query for the configuration interfaces supported by the encoder. Before calling Reset(), user must call removeRef on the PVInterface object to remove its reference to the object.

This command is valid only when pvAuthor Engine is in PVAE_STATE_OPENED state. The referenced data source and composer must be already added before this method is called. This command does not change the pvAuthor Engine engine state.

Parameters:

aDataSource Data source node to provide input data

aEncoderType MIME type of encoder to encode the source data

aComposer Opaque data to identify the composer in which a media track will be added.

aConfigInterface Pointer to configuration object for the selected encoder will be saved to this parameter upon completion of this call

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

Returns:
A unique command id for asynchronous completion

4.26.3.5 virtual PVCommandId PVAuthorEngineInterface::CancelAllCommands (const OsclAny * *aContextData* = NULL) [pure virtual]

Cancel all pending requests. The current request being processed, if any, will also be aborted. PVAE_CMD_CANCEL_ALL_COMMANDS will be passed to the command observer on completion. Currently this API is NOT SUPPORTED.

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

4.26.3.6 virtual PVCommandId PVAuthorEngineInterface::Close (const OsclAny * *aContextData* = NULL) [pure virtual]

Closes an authoring session.

All resources added and allocated to the authoring session will be released.

This command is valid only when pvAuthor engine is in PVAE_STATE_OPENED state and Upon completion of this command, pvAuthor Engine will be in PVAE_STATE_IDLE state.

Parameters:
aContextData Optional opaque data to be passed back to user with the command response

Returns:
Unique command ID to identify this command in command response

4.26.3.7 virtual PVCommandId PVAuthorEngineInterface::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). Currently this API is NOT SUPPORTED.

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:

memory_error leaves on memory allocation error.

Returns:

A unique command id for asynchronous completion

4.26.3.8 virtual PVAEState PVAuthorEngineInterface::GetPVAuthorState () [pure virtual]

This function returns the current state of the pvAuthor Engine. Application may use this info for updating display or determine if the pvAuthor Engine is ready for the next command.

Parameters:

aState Output parameter to hold state information

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

Returns:

A unique command id for synchronous completion

4.26.3.9 OSCL_IMPORT_REF void PVAuthorEngineInterface::GetSDKInfo (PVSDKInfo & *aSDKInfo*) [static]

Returns SDK version information about author engine.

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.

4.26.3.10 virtual PVCommandId PVAuthorEngineInterface::GetSDKModuleInfo (PVSDKModuleInfo & *aSDKModuleInfo*, const OsclAny * *aContextData* = NULL) [pure virtual]

Returns information about all modules currently used by the SDK. Currently this API is NOT SUPPORTED.

Parameters:

aSDKModuleInfo A reference to a PVSDKModuleInfo structure which contains the number of modules currently used by pvAuthor Engine and the PV UID and description string for each module. The PV UID 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

Returns:

A unique command id for asynchronous completion

4.26.3.11 virtual PVCommandId PVAuthorEngineInterface::Init (const OsclAny * *aContextData* = NULL) [pure virtual]

Initialize an authoring session.

Upon calling this method, no more data sources and sinks can be added to the session. Also, all configuration settings will be locked and cannot be modified until the session is reset by calling Reset(). Resources for the session will allocated and initialized to the configuration settings specified. This command is valid only when pvAuthor Engine is in PVAE_STATE_OPENED state.

Upon completion of this command, pvAuthor Engine will be in PVAE_STATE_INITIALIZED state, and the authoring session is ready to start.

Parameters:

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

Returns:

A unique command id for asynchronous completion

4.26.3.12 virtual PVCommandId PVAuthorEngineInterface::Open (const OsclAny * *aContextData* = NULL) [pure virtual]

Opens an authoring session.

This command is valid only when pvAuthor engine is in PVAE_STATE_IDLE state. Upon completion of this method, pvAuthor engine will be in PVAE_STATE_OPENED state.

Parameters:

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

Returns:

Unique command ID to identify this command in command response

4.26.3.13 virtual PVCommandId PVAuthorEngineInterface::Pause (const OsclAny * *aContextData* = NULL) [pure virtual]

Pause the authoring session.

The authoring session will be paused and no encoded output data will be sent to the data sink. This function is valid only in the PVAE_STATE_RECORDING state.

Upon completion of this command, pvAuthor Engine will be in PVAE_STATE_PAUSED state.

Parameters:

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

Returns:

A unique command id for asynchronous completion

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

This API is to allow for extensibility of the pvAuthor engine 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. TBD: Define the UIID, InterfacePtr structures

Parameters:

aUuid The UUID of the desired interface

aInterfacePtr The output pointer to the desired interface

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

Returns:

A unique command id for asynchronous completion

4.26.3.15 virtual PVCommandId PVAuthorEngineInterface::QueryUUID (const PvmfMimeString & *aMimeType*, Oscl_Vector< PVUuid, OsclMemAllocator > & *aUuids*, bool *aExactUuidsOnly* = false, const OsclAny * *aContextData* = NULL) [pure virtual]

Discover the UUIDs of interfaces associated with the specified MIME type and node

This API is to allow for extensibility of the pvAuthor Engine interface. User can query for all UUIDs associated with a particular MIME type. The UUIDs will be added to the aUuids vector provided by the user. Currently this API is NOT SUPPORTED.

Parameters:

aMimeType The MIME type of the desired interfaces

aUuids A vector to hold the discovered UUIDs

aExactUuidsOnly Turns on/off the retrival of UUIDs with aMimeType as a base type

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

Returns:

A unique command id for asynchronous completion

4.26.3.16 virtual PVCommandId PVAuthorEngineInterface::RemoveDataSink (const PVMFNodeInterface & *aDataSink*, const OsclAny * *aContextData* = NULL) [pure virtual]

Removes a previously added data sink. Currently this API does not cause any action as it is not relevant.

This command is valid only when pvAuthor Engine is in PVAE_STATE_OPENED state. This command does not change the pvAuthor Engine engine state.

Parameters:

aDataSink Reference to the data sink to be removed

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

Returns:

A unique command id for asynchronous completion

4.26.3.17 virtual PVCommandId PVAuthorEngineInterface::RemoveDataSource (const PVMFNodeInterface & *aDataSource*, const OsclAny * *aContextData* = NULL) [pure virtual]

Unbinds a previously added data source.

This command is valid only when pvAuthor Engine is in PVAE_STATE_OPENED state. This command does not change the pvAuthor Engine engine state.

Parameters:

aDataSource Reference to the data source to be removed

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

Returns:

A unique command id for asynchronous completion

4.26.3.18 virtual PVCommandId PVAuthorEngineInterface::RemoveLogAppender (const char * *aTag*, 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. Currently this API is NOT SUPPORTED.

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:

memory_error leaves on memory allocation error.

Returns:

A unique command id for asynchronous completion

4.26.3.19 virtual PVCommandId PVAuthorEngineInterface::Reset (const OsclAny * *aContextData* = NULL) [pure virtual]

Reset an initialized authoring session.

The authoring session will be stopped and all composers and encoders selected for the session will be removed. All data sources and sinks will be reset but will continue to be available for authoring the next output clip.

User must call removeRef() to remove its reference to any PVInterface objects received from SelectComposer() or AddMediaTrack() or QueryInterface() APIs before calling this method. This method would fail otherwise.

This method can be called from ANY state but PVAE_STATE_IDLE. Upon completion of this command, pvAuthor Engine will be in PVAE_STATE_OPENED state.

Parameters:

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

Returns:

A unique command id for asynchronous completion

4.26.3.20 virtual PVCommandId PVAuthorEngineInterface::Resume (const OsclAny * *aContextData* = NULL) [pure virtual]

Resume a paused authoring session.

The authoring session will be resumed and pvAuthor Engine will resume sending encoded output data to the data sinks. This function is valid only in the PVAE_STATE_PAUSED state.

Upon completion of this command, pvAuthor Engine will be in PVAE_STATE_RECORDING state.

Parameters:

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

Returns:

A unique command id for asynchronous completion

4.26.3.21 virtual PVCommandId PVAuthorEngineInterface::SelectComposer (const PVUuid & *aComposerUuid*, PVInterface *& *aConfigInterface*, const OsclAny * *aContextData* = NULL) [pure virtual]

Selects an output composer by specifying its Uuid.

pvAuthor engine the composer of the specified Uuid in the authoring session. This command is valid only when pvAuthor Engine is in PVAE_STATE_OPENED state. This command does not change the pvAuthor Engine state.

Upon completion of this command, opaque data to indentify the selected composer is provided in the callback. The user needs to use this opaque data to identify the composer when calling AddMediaTrack(), AddDataSink(). A configuration interface for the selected composer will be saved to the PVInterface pointer provided in aConfigInterface parameter. User should call queryInterface to query for the configuration interfaces supported by the composer. When configuration is complete or before calling Reset(), user must call removeRef on the PVInterface object to remove its reference to the object.

Parameters:

aComposerUuid Uuid of output composer to be used

aConfigInterface Pointer to configuration object for the selected composer will be saved to this parameter upon completion of this call

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

Returns:

A unique command id for asynchronous completion

4.26.3.22 virtual PVCommandId PVAuthorEngineInterface::SelectComposer (const PvmfMimeString & *aComposerType*, PVInterface *& *aConfigInterface*, const OsclAny * *aContextData* = NULL) [pure virtual]

Selects an output composer by specifying its MIME type.

pvAuthor engine will use the most suitable output composer of the specified MIME type available in the authoring session. This command is valid only when pvAuthor Engine is in PVAE_STATE_OPENED state. This command does not change the pvAuthor Engine state.

Upon completion of this command, opaque data to indentify the selected composer is provided in the callback. The user needs to use this opaque data to identify the composer when calling AddMediaTrack(), AddDataSink(). A configuration interface for the selected composer will be saved to the PVInterface pointer provided in aConfigInterface parameter. User should call queryInterface to query for the configuration interfaces supported by the composer. When configuration is complete or before calling Reset(), user must call removeRef on the PVInterface object to remove its reference to the object.

Parameters:

aComposerType MIME type of output composer to be used

aConfigInterface Pointer to configuration object for the selected composer will be saved to this parameter upon completion of this call

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

Returns:

A unique command id for asynchronous completion

4.26.3.23 virtual PVCommandId PVAuthorEngineInterface::SetLogAppender (const char * *aTag*, 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). Currently this API is NOT SUPPORTED.

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:

memory_error leaves on memory allocation error.

Returns:

A unique command id for asynchronous completion

4.26.3.24 virtual PVCommandId PVAuthorEngineInterface::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 set_subtree flag will allow an entire subtree, with the specified tag as the root, to be reset to the specified value. Currently this API is NOT SUPPORTED.

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:

memory_error leaves on memory allocation error.

Returns:

A unique command id for asynchronous completion

4.26.3.25 virtual PVCommandId PVAuthorEngineInterface::Start (const OsclAny * *aContextData* = NULL) [pure virtual]

Start the authoring session.

pvAuthor Engine will begin to receive source data, encode them to the specified format and quality, and send the output data to the specified data sinks. This function is valid only in the PVAE_STATE_-INITIALIZED state.

Upon completion of this command, pvAuthor Engine will be in PVAE_STATE_RECORDING state.

Parameters:

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

Returns:

A unique command id for asynchronous completion

4.26.3.26 virtual PVCommandId PVAuthorEngineInterface::Stop (const OsclAny * *aContextData* = NULL) [pure virtual]

Stops an authoring session.

The authoring session will be stopped and pvAuthor Engine will stop receiving source data from the data sources, and no further encoded data will be sent to the data sinks. This function is valid only in the PVAE_STATE_RECORDING and PVAE_STATE_PAUSED states.

Upon completion of this command, pvAuthor Engine will be in PVAE_STATE_INITIALIZED state.

Parameters:

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

Returns:

A unique command id for asynchronous completion

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

- pvauthorengineinterface.h

4.27 PVCmdResponse Class Reference

`#include <pv_engine_observer_message.h>`

Public Methods

- PVCmdResponse (PVCommandId aId, OsclAny *aContext, PVMFStatus aStatus, OsclAny *aEventData=NULL, int32 aEventDataSize=0)
- PVCmdResponse (PVCommandId aId, OsclAny *aContext, PVMFStatus aStatus, PVInterface *aEventExtInterface=NULL, OsclAny *aEventData=NULL, int32 aEventDataSize=0)
- PVResponseType GetResponseType () const
- PVCommandId GetCmdId () const
- OsclAny * GetContext () const
- PVMFStatus GetCmdStatus () const
- OsclAny * GetResponseData () const
- int32 GetResponseDataSize () const

4.27.1 Detailed Description

PVCmdResponse Class

PVCmdResponse class is used to pass completion status on previously issued commands

4.27.2 Constructor & Destructor Documentation

4.27.2.1 PVCmdResponse::PVCmdResponse (PVCommandId *aId*, OsclAny * *aContext*, PVMFStatus *aStatus*, OsclAny * *aEventData* = NULL, int32 *aEventDataSize* = 0) [inline]

Constructor for PVCmdResponse

4.27.2.2 PVCmdResponse::PVCmdResponse (PVCommandId *aId*, OsclAny * *aContext*, PVMFStatus *aStatus*, PVInterface * *aEventExtInterface* = NULL, OsclAny * *aEventData* = NULL, int32 *aEventDataSize* = 0) [inline]

Constructor with event extension interface

4.27.3 Member Function Documentation

4.27.3.1 PVCommandId PVCmdResponse::GetCmdId () const [inline]

Returns:
Returns the unique ID associated with a command of this type.

4.27.3.2 PVMFStatus PVCmdResponse::GetCmdStatus () const [inline]

Returns:
Returns the completion status of the command

4.27.3.3 OsclAny* PVCmdResponse::GetContext () const [inline]

Returns:
Returns the opaque data that was passed in with the command.

4.27.3.4 OsclAny* PVCmdResponse::GetResponseData () const [inline]

WILL BE DEPRECATED WHEN PVMFCmdResp REMOVES EVENT DATA

Returns:
Returns additional data asociated with the command. This is to be interpreted based on the command issued and the return status

4.27.3.5 int32 PVCmdResponse::GetResponseDataSize () const [inline]

4.27.3.6 PVResponseType PVCmdResponse::GetResponseType () const [inline]

WILL BE DEPRECATED SINCE IT IS NOT BEING USED. CURRENTLY RETURNS 0

Returns:
Returns the type of Response we get

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

- pv_engine_observer_message.h

4.28 PVCommandStatusObserver Class Reference

`#include <pv_engine_observer.h>`

Public Methods

- virtual void CommandCompleted (const PVCmdResponse &aResponse)=0
- virtual ~PVCommandStatusObserver ()

4.28.1 Detailed Description

PVCommandStatusObserver Class

PVCommandStatusObserver is the PV SDK observer class for notifying the status of issued command messages. The API provides a mechanism for the status of each command to be passed back along with context specific information where applicable. Applications using the PV SDKs must have a class derived from PVCommandStatusObserver and implement the pure virtual function in order to receive event notifications from a PV SDK. Additional information is optionally provided via derived classes.

4.28.2 Constructor & Destructor Documentation

4.28.2.1 virtual PVCommandStatusObserver::~PVCommandStatusObserver () [inline, virtual]

4.28.3 Member Function Documentation

4.28.3.1 virtual void PVCommandStatusObserver::CommandCompleted (const PVCmdResponse & *aResponse*) [pure virtual]

Handle an event that has been generated.

Parameters:

aResponse The response to a previously issued command.

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

- pv_engine_observer.h

4.29 PVConfigInterface Class Reference

`#include <pv_config_interface.h>`

4.29.1 Detailed Description

Base interface for all configuration classes

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

- pv_config_interface.h

4.30 PVEngineAsyncEvent Class Reference

`#include <pv_engine_types.h>`

Public Methods

- PVEngineAsyncEvent (int32 aAsyncEventType)
- PVEngineAsyncEvent (const PVEngineAsyncEvent &aAsyncEvent)
- int32 GetAsyncEventType () const

Data Fields

- int32 iAsyncEventType

4.30.1 Detailed Description

PVEngineAsyncEvent Class

PVEngineAsyncEvent class is a data class to hold asynchronous events generated by the engine. The class is meant to be used inside the engine and not exposed to the interface layer or above.

4.30.2 Constructor & Destructor Documentation

4.30.2.1 PVEngineAsyncEvent::PVEngineAsyncEvent (int32 *aAsyncEventType*) [inline]

The constructor for PVEngineCommand which allows the data values to be set.

Parameters:

aCmdType The command type value for this command. The value is an engine-specific 32-bit value.

aCmdId The command ID assigned by the engine for this command.

aContextData The pointer to the passed-in context data for this command.

Returns:

None

4.30.2.2 PVEngineAsyncEvent::PVEngineAsyncEvent (const PVEngineAsyncEvent & *aAsyncEvent*) [inline]

The copy constructor for PVEngineAsyncEvent. Used mainly for Oscl_Vector.

Parameters:

aAsyncEvent The reference to the source PVEngineAsyncEvent to copy the data values from.

Returns:

None

4.30.3 Member Function Documentation

4.30.3.1 int32 PVEngineAsyncEvent::GetAsyncEventType () const [inline]

This function returns the stored asynchronous event type value.

Returns:
The signed 32-bit event type value.

4.30.4 Field Documentation

4.30.4.1 int32 PVEngineAsyncEvent::iAsyncEventType

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

- pv_engine_types.h

4.31 PVEngineCommand Class Reference

`#include <pv_engine_types.h>`

Public Methods

- PVEngineCommand (int32 aCmdType, PVCommandId aCmdId, OsclAny *aContextData=NULL, OsclAny *aParam1=NULL, OsclAny *aParam2=NULL, OsclAny *aParam3=NULL)
- PVEngineCommand (const PVEngineCommand &aCmd)
- int32 GetCmdType () const
- PVCommandId GetCmdId () const
- OsclAny * GetContext () const
- OsclAny * GetParam1 () const
- OsclAny * GetParam2 () const
- OsclAny * GetParam3 () const
- const PvmfMimeString & GetMimeType () const
- PVUuid GetUuid () const
- void SetMimeType (const PvmfMimeString &aMimeType)
- void SetUuid (const PVUuid &aUuid)

Data Fields

- int32 iCmdType
- PVCommandId iCmdId
- OsclAny * iContextData
- OsclAny * iParam1
- OsclAny * iParam2
- OsclAny * iParam3
- OSCL_HeapString< OsclMemAllocator > iMimeType
- PVUuid iUuid

4.31.1 Detailed Description

PVEngineCommand Class

PVEngineCommand class is a data class to hold issued commands. The class is meant to be used inside the engine and not exposed to the interface layer or above.

4.31.2 Constructor & Destructor Documentation

4.31.2.1 PVEngineCommand::PVEngineCommand (int32 *aCmdType*, PVCommandId *aCmdId*, OsclAny * *aContextData* = NULL, OsclAny * *aParam1* = NULL, OsclAny * *aParam2* = NULL, OsclAny * *aParam3* = NULL) [inline]

The constructor for PVEngineCommand which allows the data values to be set.

Parameters:

aCmdType The command type value for this command. The value is an engine-specific 32-bit value.

aCmdId The command ID assigned by the engine for this command.

aContextData The pointer to the passed-in context data for this command.

Returns:
None

4.31.2.2 PVEngineCommand::PVEngineCommand (const PVEngineCommand & *aCmd*) [inline]

The copy constructor for PVEngineCommand. Used mainly for Oscl_Vector.

Parameters:
aCmd The reference to the source PVEngineCommand to copy the data values from.

Returns:
None

4.31.3 Member Function Documentation

4.31.3.1 PVCommandId PVEngineCommand::GetCmdId () const [inline]

This function returns the stored command ID value.

Returns:
The PVCommandId value for this command.

4.31.3.2 int32 PVEngineCommand::GetCmdType () const [inline]

This function returns the stored command type value.

Returns:
The signed 32-bit command type value for this command.

4.31.3.3 OsclAny* PVEngineCommand::GetContext () const [inline]

This function returns the stored context data pointer.

Returns:
The pointer to the context data for this command

4.31.3.4 const PvmfMimeString& PVEngineCommand::GetMimeType () const [inline]

This function returns Mime type parameter for this command

Returns:
The Mime type parameter for this command

4.31.3.5 OsclAny* PVEngineCommand::GetParam1 () const [inline]

This function returns the first stored parameter pointer.

Returns:
The pointer to the first stored parameter for this command

4.31.3.6 OsclAny* PVEngineCommand::GetParam2 () const [inline]

This function returns the second stored parameter pointer.

Returns:
The pointer to the second stored parameter for this command

4.31.3.7 OsclAny* PVEngineCommand::GetParam3 () const [inline]

This function returns the third stored parameter pointer.

Returns:
The pointer to the third stored parameter for this command

4.31.3.8 PVUuid PVEngineCommand::GetUuid () const [inline]

This function returns Uuid parameter for this command

Returns:
The Uuid parameter for this command

4.31.3.9 void PVEngineCommand::SetMimeType (const PvmfMimeString & *aMimeType*) [inline]

This function stores Mime type parameter of this command

4.31.3.10 void PVEngineCommand::SetUuid (const PVUuid & *aUuid*) [inline]

This function stores the Uuid parameter of this command

4.31.4 Field Documentation

4.31.4.1 PVCommandId PVEngineCommand::iCmdId

4.31.4.2 int32 PVEngineCommand::iCmdType

4.31.4.3 OsclAny* PVEngineCommand::iContextData

4.31.4.4 OSCL_HeapString<OsclMemAllocator> PVEngineCommand::iMimeType

4.31.4.5 OsclAny* PVEngineCommand::iParam1

4.31.4.6 OsclAny* PVEngineCommand::iParam2

4.31.4.7 OsclAny* PVEngineCommand::iParam3

4.31.4.8 PVUuid PVEngineCommand::iUuid

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

- pv_engine_types.h

4.32 PVErrorEventObserver Class Reference

`#include <pv_engine_observer.h>`

Public Methods

- virtual void HandleErrorEvent (const PVAsyncErrorEvent &aEvent)=0
- virtual ~PVErrorEventObserver ()

4.32.1 Detailed Description

PVErrorEventObserver Class

PVErrorEventObserver is the PV SDK event observer class. It is used for communicating unsolicited error events back to the user of the SDK.

Applications using the PV SDKs must have a class derived from PVErrorEventObserver and implement the pure virtual function in order to receive error notifications from a PV SDK.

4.32.2 Constructor & Destructor Documentation

4.32.2.1 virtual PVErrorEventObserver::~PVErrorEventObserver () [inline, virtual]

4.32.3 Member Function Documentation

4.32.3.1 virtual void PVErrorEventObserver::HandleErrorEvent (const PVAsyncErrorEvent & *aEvent*) [pure virtual]

Handle an error event that has been generated.

Parameters:
aEvent The event to be handled.

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

- pv_engine_observer.h

4.33 PVInformationalEventObserver Class Reference

`#include <pv_engine_observer.h>`

Public Methods

- virtual void HandleInformationalEvent (const PVAsyncInformationalEvent &aEvent)=0
- virtual ~PVInformationalEventObserver ()

4.33.1 Detailed Description

PVInformationalEventObserver Class

PVInformationalEventObserver is the PV SDK event observer class. It is used for communicating unsolicited informational events back to the user of the SDK.

Applications using the PV SDKs must have a class derived from PVInformationalEventObserver and implement the pure virtual function in order to receive informational event notifications from a PV SDK.

4.33.2 Constructor & Destructor Documentation

4.33.2.1 virtual PVInformationalEventObserver::~PVInformationalEventObserver () [inline, virtual]

4.33.3 Member Function Documentation

4.33.3.1 virtual void PVInformationalEventObserver::HandleInformationalEvent (const PVAsyncInformationalEvent & *aEvent*) [pure virtual]

Handle an informational event that has been generated.

Parameters:

aEvent The event to be handled.

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

- pv_engine_observer.h

4.34 PVSDKInfo Struct Reference

`#include <pv_engine_types.h>`

Public Methods

- PVSDKInfo ()
- PVSDKInfo & operator= (const PVSDKInfo &aSDKInfo)

Data Fields

- OSCL_StackString< 80 > iLabel
- uint32 iDate

4.34.1 Constructor & Destructor Documentation

4.34.1.1 PVSDKInfo::PVSDKInfo () [inline]

4.34.2 Member Function Documentation

4.34.2.1 PVSDKInfo& PVSDKInfo::operator= (const PVSDKInfo & *aSDKInfo*) [inline]

4.34.3 Field Documentation

4.34.3.1 uint32 PVSDKInfo::iDate

4.34.3.2 OSCL_StackString<80> PVSDKInfo::iLabel

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

- pv_engine_types.h

4.35 TPVCmnSDKInfo Struct Reference

`#include <pv_common_types.h>`

Public Methods

- TPVCmnSDKInfo ()
- TPVCmnSDKInfo & operator= (const TPVCmnSDKInfo &aSDKInfo)

Data Fields

- OSCL_StackString< 80 > iLabel
- uint32 iDate

4.35.1 Constructor & Destructor Documentation

4.35.1.1 TPVCmnSDKInfo::TPVCmnSDKInfo () [inline]

4.35.2 Member Function Documentation

4.35.2.1 TPVCmnSDKInfo& TPVCmnSDKInfo::operator= (const TPVCmnSDKInfo & *aSDKInfo*) [inline]

4.35.3 Field Documentation

4.35.3.1 uint32 TPVCmnSDKInfo::iDate

4.35.3.2 OSCL_StackString<80> TPVCmnSDKInfo::iLabel

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

- pv_common_types.h

Chapter 5

pvauthor_engine File Documentation

5.1 pv_common_types.h File Reference

```
#include "oscl_types.h"
#include "oscl_mem.h"
#include "oscl_string_containers.h"
```

Data Structures

- class CPVCmnAsyncEvent
- class CPVCmnCmdResp
- class CPVCmnInterfaceObserverMessage
- class CPVCmnInterfaceObserverMessageCompare
- class MPVCmnCmdStatusObserver
- class MPVCmnErrorEventObserver
- class MPVCmnInfoEventObserver
- struct TPVCmnSDKInfo

Defines

- #define PV_COMMON_ASYNC_EVENT_LOCAL_BUF_SIZE 8

Typedefs

- typedef int32 TPVCmnCommandType
- typedef int32 TPVCmnCommandId
- typedef int32 TPVCmnCommandStatus
- typedef int32 TPVCmnEventType
- typedef void * TPVCmnExclusivePtr
- typedef void * TPVCmnInterfacePtr
- typedef int32 TPVCmnResponseType
- typedef int32 TPVCmnSDKModuleInfo
- typedef uint8 * TPVCmnMIMEType

- typedef uint32 TPVCmnUUID
- typedef int32 CPVCmnVideoCaps
- typedef int32 CPVCmnVideoPrefs
- typedef int32 CPVCmnAudioCaps
- typedef int32 CPVCmnAudioPrefs
- typedef CPVCmnAsyncEvent CPVCmnAsyncInfoEvent
- typedef CPVCmnAsyncEvent CPVCmnAsyncErrorEvent

5.1.1 Define Documentation

5.1.1.1 #define PV_COMMON_ASYNC_EVENT_LOCAL_BUF_SIZE 8

5.1.2 Typedef Documentation

5.1.2.1 typedef CPVCmnAsyncEvent CPVCmnAsyncErrorEvent

5.1.2.2 typedef CPVCmnAsyncEvent CPVCmnAsyncInfoEvent

5.1.2.3 typedef int32 CPVCmnAudioCaps

5.1.2.4 typedef int32 CPVCmnAudioPrefs

5.1.2.5 typedef int32 CPVCmnVideoCaps

5.1.2.6 typedef int32 CPVCmnVideoPrefs

5.1.2.7 typedef int32 TPVCmnCommandId

5.1.2.8 typedef int32 TPVCmnCommandStatus

5.1.2.9 typedef int32 TPVCmnCommandType

5.1.2.10 typedef int32 TPVCmnEventType

5.1.2.11 typedef void* TPVCmnExclusivePtr

5.1.2.12 typedef void* TPVCmnInterfacePtr

5.1.2.13 typedef uint8* TPVCmnMIMEType

5.1.2.14 typedef int32 TPVCmnResponseType

5.1.2.15 typedef int32 TPVCmnSDKModuleInfo

5.1.2.16 typedef uint32 TPVCmnUUID

5.2 pv_config_interface.h File Reference

```
#include "oscl_base.h"
#include "oscl_vector.h"
```

Data Structures

- class PVConfigInterface

5.3 pv_engine_observer.h File Reference

```
#include "pv_engine_observer_message.h"
```

Data Structures

- class PVCommandStatusObserver
- class PVErrorEventObserver
- class PVInformationalEventObserver

5.4 pv_engine_observer_message.h File Reference

```
#include "oscl_base.h"
#include "oscl_mem.h"
#include "pvmf_return_codes.h"
#include "pvmf_event_handling.h"
#include "pv_engine_types.h"
```

Data Structures

- class PVAsyncErrorEvent
- class PVAsyncInformationalEvent
- class PVCmdResponse

5.5 pv_engine_types.h File Reference

```
#include "oscl_base.h"
#include "oscl_string.h"
#include "oscl_string_containers.h"
#include "oscl_mem.h"
#include "pvmf_format_type.h"
#include "pv_uuid.h"
#include "pv_interface.h"
#include "oscl_vector.h"
```

Data Structures

- class PVEngineAsyncEvent
- class PVEngineCommand
- struct PVSDKInfo

Typedefs

- typedef int32 PVCommandId
- typedef int32 PVEventType
- typedef OsclAny * PVExclusivePtr
- typedef int32 PVResponseType
- typedef int32 PVLogLevelInfo
- typedef Oscl_Vector< OSCL_HeapString< OsclMemAllocator >, OsclMemAllocator > PVPMetadataList
- typedef int32 PVSDKModuleInfo

5.5.1 Typedef Documentation

5.5.1.1 typedef int32 PVCommandId

5.5.1.2 typedef int32 PVEventType

5.5.1.3 typedef OsclAny* PVExclusivePtr

5.5.1.4 typedef int32 PVLogLevelInfo

5.5.1.5 typedef Oscl_Vector<OSCL_HeapString<OsclMemAllocator>, OsclMemAllocator> PVPMetadataList

5.5.1.6 typedef int32 PVResponseType

5.5.1.7 typedef int32 PVSDKModuleInfo

5.6 pv_interface_cmd_message.h File Reference

```
#include "pv_common_types.h"
#include "pv_engine_types.h"
```

Data Structures

- class CPVCmnInterfaceCmdMessage

Functions

- int32 operator< (const CPVCmnInterfaceCmdMessage &a, const CPVCmnInterfaceCmdMessage &b)

5.6.1 Function Documentation

5.6.1.1 int32 operator< (const CPVCmnInterfaceCmdMessage & *a*, const CPVCmnInterfaceCmdMessage & *b*) [inline]

5.7 pv_plugin_interfaces.h File Reference

```
#include "pv_common_symbian_types.h"
```

Data Structures

- class CPVMMFPointerBuffer
- class MPVAudioInput
- class MPVAudioOutput
- class MPVDataSink
- class MPVDataSinkBase
- class MPVDataSource
- class MPVDataSourceAndSink
- class MPVDataSourceBase
- class MPVDevSoundAudioInput
- class MPVDevSoundAudioOutput
- class MPVPluginBase
- class MPVVideoInput
- class MPVVideoOutput
- class MPVYuvFrameBuffer

Defines

- #define KPVUidAudioInputInterface TPVUuid(0x194e8655,0x944c,0x402c,0xb0,0xc2,0xf7,0xbd,0xd5,0xe5,0x43,0x2f)
- #define KPVUidDevSoundAudioInputInterface TPVUuid(0x9e2c416e,0x0299,0x4775,0x88,0xfa,0x42,0x53,0xbc,0xbc,0x58
- #define KPVUidAudioOutputInterface TPVUuid(0xf5c5b825,0x90eb,0x4091,0xbe,0xea,0xa0,0xc3,0x9b,0xe2,0x00,0xaf)
- #define KPVUidDevSoundAudioOutputInterface TPVUuid(0x48edb46a,0x60e4,0x4e83,0xb1,0xad,0x92,0xa8,0xd4,0x07,0x
- #define KPVUidVideoInputInterface TPVUuid(0xfb320151,0x6d06,0x4bd5,0xa2,0x68,0x61,0x01,0xdb,0x25,0x1c,0x0e)
- #define KPVUidVideoOutputInterface TPVUuid(0x0bb9d8a8,0x9623,0x4dec,0x84,0x0b,0xb9,0xf2,0x66,0xf8,0x4e,0x3d)
- #define KPVUidProxiedInterface TPVUuid(0xf7076653,0x6088,0x47c6,0x88,0xc1,0xb7,0xed,0x28,0xe7,0x2b,0xea)
- #define PV_YUV_BUFFER_DEF_WIDTH 176
- #define PV_YUV_BUFFER_DEF_HEIGHT 144

Typedefs

- typedef TAny * RPvCommServer
- typedef MPVPluginBase MPVCommServerBase

Variables

- const TInt KPVUidDataSrcPrime = 0xFFFFFF08
- const TInt KPVUidDataSrcPlay = 0xFFFFFF09
- const TInt KPVUidDataSrcPause = 0xFFFFFF0A
- const TInt KPVUidDataSrcStop = 0xFFFFFF0B
- const TInt KPVUidDataSinkPrime = 0xFFFFFF0C
- const TInt KPVUidDataSinkPlay = 0xFFFFFF0D
- const TInt KPVUidDataSinkPause = 0xFFFFFF0E
- const TInt KPVUidDataSinkStop = 0xFFFFFF0F
- const TUid KPVUidYUVFrameBuffer = {0xFFFFFF0d}

5.7.1 Define Documentation

5.7.1.1 #define KPVUidAudioInputInterface TPVUuid(0x194e8655,0x944c,0x402c,0xb0,0xc2,0xf7,0xbd,0xd5,0xe5,0x43,0x2f)

Uuid for Querying the MPVAudioInput interface

5.7.1.2 #define KPVUidAudioOutputInterface TPVUuid(0xf5c5b825,0x90eb,0x4091,0xbe,0xea,0xa0,0xc3,0x9b,0xe2,0x00,0xaf)

Uuid for Querying the MPVAudioOutput interface

5.7.1.3 #define KPVUidDevSoundAudioInputInterface TPVUuid(0x9e2c416e,0x0299,0x4775,0x88,0xfa,0x42,0x53,0xbc,0xbc,0x58,0xbf)

Uuid for Querying the MPVDevSoundAudioInput interface

5.7.1.4 #define KPVUidDevSoundAudioOutputInterface TPVUuid(0x48edb46a,0x60e4,0x4e83,0xb1,0xad,0x92,0xa8,0xd4,0x07,0x04,0x7a)

Uuid for Querying the MPVDevSoundAudioInput interface

5.7.1.5 #define KPVUidProxiedInterface TPVUuid(0xf7076653,0x6088,0x47c6,0x88,0xc1,0xb7,0xed,0x28,0xe7,0x2b,0xea)

Uuid for Querying the Proxied version of any interface

5.7.1.6 #define KPVUidVideoInputInterface TPVUuid(0xfb320151,0x6d06,0x4bd5,0xa2,0x68,0x61,0x01,0xdb,0x25,0x1c,0x0e)

Uuid for Querying the MPVVideoInput interface

5.7.1.7 #define KPVUidVideoOutputInterface TPVUuid(0x0bb9d8a8,0x9623,0x4dec,0x84,0x0b,0xb9,0xf2,0x66,0xf8,0x4e,0x3d)

Uuid for Querying the MPVVideoOutput interface

5.7.1.8 #define PV_YUV_BUFFER_DEF_HEIGHT 144

5.7.1.9 #define PV_YUV_BUFFER_DEF_WIDTH 176

5.7.2 Typedef Documentation

5.7.2.1 typedef MPVPluginBase MPVCommServerBase

MPVCommServerBase Class

MPVCommServerBase is to be implemented by a server for COMM source and sink interfaces. It could be based on a serial comms interface in which case it aggregates a single comm source and sink. In the case of sockets it could support multiple sources and sinks

5.7.2.2 typedef TAny* RPvCommServer

5.7.3 Variable Documentation

5.7.3.1 const TInt KPVUidDataSinkPause = 0xFFFFFF0E

5.7.3.2 const TInt KPVUidDataSinkPlay = 0xFFFFFF0D

5.7.3.3 const TInt KPVUidDataSinkPrime = 0xFFFFFF0C

MPVDataSinkBase Event categories

These are the UIDs of the categories that should be returned via the MAsyncEventHandler interface for the async event callbacks.

5.7.3.4 const TInt KPVUidDataSinkStop = 0xFFFFFF0F

5.7.3.5 const TInt KPVUidDataSrcPause = 0xFFFFFF0A

5.7.3.6 const TInt KPVUidDataSrcPlay = 0xFFFFFF09

5.7.3.7 const TInt KPVUidDataSrcPrime = 0xFFFFFF08

MPVDataSourceBase Event categories

These are the UIDs of the categories that should be returned via the MAsyncEventHandler interface for the async event callbacks.

5.7.3.8 const TInt KPVUidDataSrcStop = 0xFFFFFF0B

5.7.3.9 const TUid KPVUidYUVFrameBuffer = {0xFFFFFF0d}

Uid for MPVYuvFrameBuffer interface

5.8 pvauthorenginefactory.h File Reference

Data Structures

- class PVAuthorEngineFactory

5.9 pvauthorengineinterface.h File Reference

```
#include "oscl_base.h"
#include "oscl_string.h"
#include "pv_engine_types.h"
```

Data Structures

- class PVAuthorEngineInterface

Enumerations

- enum PVAEState { PVAE_STATE_IDLE = 0, PVAE_STATE_OPENED, PVAE_STATE_-INITIALIZED, PVAE_STATE_RECORDING, PVAE_STATE_PAUSED, PVAE_STATE_ERROR }
- enum PVAEErrorEvent { PVAE_ENCODE_ERROR }
- enum PVAEInfoEvent { PVAE_OUTPUT_PROGRESS }

5.9.1 Enumeration Type Documentation

5.9.1.1 enum PVAEErrorEvent

Enumeration of errors from pvAuthor Engine.

Enumeration values:

PVAE_ENCODE_ERROR

5.9.1.2 enum PVAEInfoEvent

Enumeration of informational events from pvAuthor Engine.

Enumeration values:

PVAE_OUTPUT_PROGRESS

5.9.1.3 enum PVAEState

An enumeration of the major states of the pvAuthor Engine.

Enumeration values:

PVAE_STATE_IDLE

PVAE_STATE_OPENED

PVAE_STATE_INITIALIZED

PVAE_STATE_RECORDING

PVAE_STATE_PAUSED

PVAE_STATE_ERROR

Index