

AAX SDK

2.3.1

Generated by Doxygen 1.8.9.1

Fri Feb 2 2018 21:15:44

Contents

Chapter 1

Main Page

[AAX SDK Manual](#)

1.1 Welcome to AAX

Select the "Manual" tab to see a full list of documentation pages, or choose from the topics below.

Note

The search function only includes indexing of code symbols and page titles. To search for specific text strings in the AAX SDK manual it is best to use a text search tool such as grep or FINDSTR on the AAX SDK directory or search for the desired text within the PDF version of the AAX SDK documentation.

1.1.1 The basics

- For a general overview of AAX and a walk-through of the DemoGain example plug-in, see [Getting Started with AAX](#)
- If you are new to Pro Tools then you may also want to read through the first few sections of the [Pro Tools Guide](#)
- The [sample plug-ins](#) provide examples of both basic and advanced AAX features
- If you want to find out more about the basic structure of AAX plug-ins, see [Core AAX Interface](#)
- To learn more about audio processing in AAX, see [Real-time algorithm callback](#)
- To learn about adding parameters and controls to your plug-in, see [Data model interface](#)
- For more information about plug-in configuration and initial set-up, see [Description callback](#)
- Ready to ship your new plug-in? See [Distributing Your AAX Plug-In](#) for information about finalizing and distributing your AAX products

1.1.2 More topics

- To see the interface that an AAX plug-in's host-based modules use to interact with the host, see [AAX_IController](#)
- To find out more about how different modules in an AAX plug-in communicate with one another, see [AAX communication protocols](#)
- For information about creating advanced non-real-time AAX plug-ins, see [Offline processing interface](#)

- For more information about implementing custom parameter and control behavior, see [Taper delegates](#) and [Display delegates](#)
- To find out how to create and optimize an AAX plug-in for Avid's HDX platform, see the [TI Guide](#)
- You can find signal processing utility classes and functions available for optimizing on Native and DSP in the /TI/SignalProcessing folder

1.1.3 Test tools and utilities

- For information about the tool for testing basic functionality of your plug-in, see [DSH Guide](#)
- To learn how to automate different test scenarios for DSH, see [DTT Guide](#)

1.1.4 Supplemental information

- [Example Plug-Ins](#)
- [Change Log](#)
- [Host Support](#)
- [Known Issues](#)

1.2 SDK Folder hierarchy

Documentation

SDK documentation

ExamplePlugins

Example plug-in projects [More information](#)

Extensions

Demonstrations of how to extend the AAX SDK, for example by incorporating third-party GUI frameworks into AAX plug-ins. [More information](#)

Interfaces

Interface headers and other resources required for use of the AAX SDK library

Libs

Source code for the AAX SDK library, a collection of default implementations and utility classes for use in all AAX plug-ins

TI

Various resources for use with TI's Code Composer Studio IDE and Avid's TI testing toolset

Utilities

Common SDK utilities and resources

1.3 Contacting Avid

Your personal [avid user account](#) is your hub for AAX Toolkit services and developer support.

Log in at [avid.com](#) for access to the full range of tools and services provided to AAX developers, including the [AAX developer forum](#). If you have any questions on the AAX SDK documentation or require support with AAX development, we encourage you to post them to the forum as your first line of inquiry.

If you have time-sensitive or critical support inquiries, contact the AAX development team directly at devservices@avid.com. Any AAX questions sent to this alias will be promptly addressed by the most appropriate contact here at Avid.

If you require NFR (Not For Resale) licenses to Avid software for AAX development please send an e-mail to devauth@avid.com with "License Request" in the subject.

If you require access to the digital signing toolkit from PACE Anti-Piracy, Inc. for compatibility with Pro Tools then please follow the instructions [here](#).

The following chart describes these and other ways of connecting with Avid to take advantage of the services provided to AAX developers:

Chapter 2

Not Used by AAX Plug-Ins

globalScope> Member [AAX_eUpdateSource_Delay](#)

Chapter 3

Host Compatibility Notes

Member [AAX_CMidiPacket::mIsImmediate](#)

This value is not currently set. Use `mTimestamp == 0` to detect immediate packets

Member [AAX_CParameter< T >::AAX_CParameter](#) (`AAX_CParamID identifier, const AAX_IString &name, T defaultValue, const AAX_ITaperDelegate< T > &taperDelegate, const AAX_IDisplayDelegate< T > &displayDelegate, bool automatable=false`)

As of Pro Tools 10.2, DAE will check for a matching parameter NAME and not an ID when reading in automation data from a session saved with an AAX plug-ins RTAS/TDM counter part.

As of Pro Tools 11.1, AAE will first try to match ID. If that fails, AAE will fall back to matching by Name.

Module [AAX_DigiTrace_Guide](#)

This feature is available in Pro Tools 12.6 and higher

globalScope> Member [AAX_eConstraintLocationMask_FixedLatencyDomain](#)

This constraint is not currently supported by any AAX host

globalScope> Member [AAX_eCurveType_Dynamics](#)

Pro Tools requests this curve type for [Dynamics](#) plug-ins only

globalScope> Member [AAX_eCurveType_EQ](#)

Pro Tools requests this curve type for [EQ](#) plug-ins only

globalScope> Member [AAX_eCurveType_Reduction](#)

Pro Tools requests this curve type for [Dynamics](#) plug-ins only

globalScope> Member [AAX_eDataInPortType_Incremental](#)

Supported in Pro Tools 12.5 and higher; when [AAX_eDataInPortType_Incremental](#) is not supported the port will be treated as [AAX_eDataInPortType_Unbuffered](#)

globalScope> Member [AAX_EHostModeBits](#)

Supported in Venue 5.6 and higher

globalScope> Member [AAX_eNotificationEvent_ASPreviewState](#)

Supported in Pro Tools 11 and higher

Not supported by Media Composer

globalScope> Member [AAX_eNotificationEvent_ASProcessingState](#)

Supported in Pro Tools 11 and higher

Not supported by Media Composer

globalScope> Member [AAX_eNotificationEvent_DelayCompensationState](#)

Supported in Pro Tools 12.6 and higher

globalScope> Member [AAX_eNotificationEvent_EnteringOfflineMode](#)

Supported in Pro Tools 11 and higher

globalScope> Member AAX_eNotificationEvent_ExitingOfflineMode

Supported in Pro Tools 11 and higher

globalScope> Member AAX_eNotificationEvent_HostModeChanged

Supported in Venue 5.6 and higher

globalScope> Member AAX_eNotificationEvent_MaxViewSizeChanged

Supported in Pro Tools 11.1 and higher

globalScope> Member AAX_eNotificationEvent_PresetOpened

Supported in Pro Tools 11 and higher

globalScope> Member AAX_eNotificationEvent_SessionBeingOpened

Supported in Pro Tools 11 and higher

Not supported by Media Composer

globalScope> Member AAX_eNotificationEvent_SessionPathChanged

Supported in Pro Tools 11.1 and higher

globalScope> Member AAX_eNotificationEvent_SideChainBeingConnected

Supported in Pro Tools 11.1 and higher

globalScope> Member AAX_eNotificationEvent_SideChainBeingDisconnected

Supported in Pro Tools 11.1 and higher

globalScope> Member AAX_eNotificationEvent_SignalLatencyChanged

Supported in Pro Tools 11.1 and higher

globalScope> Member AAX_eNotificationEvent_TrackNameChanged

Supported in Pro Tools 11.2 and higher

Not supported by Media Composer

globalScope> Member AAX_ePlugInStrings_Progress

Not currently supported by Pro Tools

globalScope> Member AAX_eProcessingState_BeginPassGroup

AudioSuite pass group notifications are supported starting in Pro Tools 12.0

globalScope> Member AAX_eProcessingState_EndPassGroup

AudioSuite pass group notifications are supported starting in Pro Tools 12.0

globalScope> Member AAX_eProperty_Constraint_NeverUnload

AAX_eProperty_Constraint_NeverUnload is not currently implemented in DAE or AAE

globalScope> Member AAX_eProperty_DestinationTrack

This property is not supported on Media Composer

globalScope> Member AAX_eProperty_LatencyContribution

Maximum delay compensation limits will vary from host to host. If your plug-in exceeds the delay compensation sample limit for a given AAX host then you should note this limitation in your user documentation. Example limits:

- Pro Tools 9 and higher: 16,383 samples at 44.1/48 kHz, 32,767 samples at 88.2/96 kHz, or 65,534 samples at 176.4/192 kHz
- Media Composer 8.1 and higher: 16,383 samples at 44.1/48 kHz, 32,767 samples at 88.2/96 kHz

globalScope> Member AAX_eProperty_OptionalAnalysis

In Media Composer, optional analysis will also be performed automatically before each channel is rendered.
See [MCDEV-2904](#)

globalScope> Member AAX_eProperty_SideChainStemFormat

Currently Pro Tools supports only [AAX_eStemFormat_Mono](#) side chain inputs

AAX_eProperty_SideChainStemFormat is not currently implemented in DAE or AAE

AAX_eProperty_SideChainStemFormat is not currently implemented in DAE or AAE

globalScope > Member AAX_eProperty_UsesClientGUI

Currently supported by Pro Tools only

Member AAX_IACFEffectParameters::CompareActiveChunk (const AAX_SPlugInChunk *iChunkP, AAX_CBoolean *oIsEqual) const =0

In Pro Tools, this method will only be called if a prior call to [GetNumberOfChanges\(\)](#) has indicated that the plug-in's state has changed. If the plug-in's current settings are different from the settings in aChunkP then the plug-in's Compare Light will be illuminated in the plug-in header, allowing users to toggle between the plug-in's custom state and its saved state.

Member AAX_IACFEffectParameters::GetCurveData (AAX_CTypeID iCurveType, const float *iValues, uint32_t iNumValues, float *oValues) const =0

Versions of S6 software which support the [GetCurveDataDisplayRange\(\)](#) method will not display a plug-in's curve data unless both [GetCurveData\(\)](#) and [GetCurveDataDisplayRange\(\)](#) are supported by the plug-in.

Member AAX_IACFEffectParameters::GetParameterNameOfLength (AAX_CParamID iParamID, AAX_IString *oName, int32_t iNameLength) const =0

In most cases, the AAX host will call [GetParameterName\(\)](#) or [GetParameterNameOfLength\(\)](#) to retrieve parameter names for display. However, when Pro Tools is retrieving a plug-in name for display on a control surface the XML data stored in the plug-in's page tables will be used in preference to values retrieved from these methods.

Member AAX_IComponentDescriptor::AddAuxOutputStem (AAX_CFieldIndex inFieldIndex, int32_t inStemFormat, const char inNameUTF8[])=0

There is a hard limit to the number of outputs that Pro Tools supports for a single plug-in instance. This limit is currently set at 256 channels, which includes all of the plug-in's output channels in addition to the sum total of all of its aux output stem channels.

Pro Tools supports only mono and stereo auxiliary output stem formats

Member AAX_IComponentDescriptor::AddClock (AAX_CFieldIndex inFieldIndex)=0

As of Pro Tools 11.1, this field may be used in both Native and DSP plug-ins. The DSP clock data is a 16-bit cycling counter. This field was only available for Native plug-ins in previous Pro Tools versions.

Member AAX_IComponentDescriptor::AddMIDIINode (AAX_CFieldIndex inFieldIndex, AAX_EMIDIINodeType inNodeType, const char inNodeName[], uint32_t channelMask)=0

Due to current restrictions MIDI data won't be delivered to DSP algorithms, only to AAX Native.

Member AAX_IController::GetHostName (AAX_IString *outHostNameString) const =0

Pro Tools versions from Pro Tools 11.0 to Pro Tools 12.3.1 will return a generic version string to this call. This issue is resolved beginning in Pro Tools 12.4.

Member AAX_IMIDIINode::PostMIDIpacket (AAX_CMidiPacket *packet)=0

Pro Tools supports the following MIDI events from plug-ins:

- NoteOn
- NoteOff
- Pitch bend
- Polyphonic key pressure
- Bank select (controller #0)
- Program change (no bank)
- Channel pressure

Member AAX_ITransport::GetBarBeatPosition (int32_t *Bars, int32_t *Beats, int64_t *DisplayTicks, int64_t SampleLocation) const =0

There is a minor performance cost associated with using this API in Pro Tools. It should not be used excessively without need.

Member AAX_ITransport::GetCurrentLoopPosition (bool *bLooping, int64_t *LoopStartTick, int64_t *LoopEndTick) const =0

This does not indicate anything about the status of the "Loop Record" option. Even when the host is configured to loop playback, looping may not occur if certain conditions are not met (i.e. the length of the selection is too short)

Member [AAX_ITransport::GetCurrentTickPosition \(int64_t *TickPosition\) const =0](#)

The tick resolution here is different than that of the tick displayed in Pro Tools. "Display ticks" (as they are called) are 1/960 of a quarter note.

Member [AAX_ITransport::GetCustomTickPosition \(int64_t *oTickPosition, int64_t iSampleLocation\) const =0](#)

There is a minor performance cost associated with using this API in Pro Tools. It should not be used excessively without need.

Member [AAX_IViewContainer::GetModifiers \(uint32_t *outModifiers\)=0](#)

Although this method allows plug-ins to acquire the current state of the Windows key (normally blocked by Pro Tools), plug-ins should not use key combinations that require this key.

Module [AAX_Media_Composer_Guide](#)

Some early versions of Media Composer 8 do not search the system plug-ins directory recursively. If your plug-ins are installed into a sub-directory beneath this main directory then they will not be loaded by the affected versions of Media Composer.

Module [AAX_Page_Table_Guide](#)

Pro Tools versions prior to Pro Tools 11.1 use plug-ins' ProControl and ICON page tables (Dynamics, EQ, Channel Strip, Custom Fader, etc.) to map plug-in parameters to EUCON-enabled surfaces, so be sure that your plug-ins also implement these page tables correctly so that users with earlier versions of Pro Tools can have the best possible experience when using your plug-ins.

Module [AAX_Pro_Tools_Guide](#)

Pro Tools 11 requires PACE Eden digital signatures for AAX plug-ins.

Pro Tools 10 requires either PACE DSig or Eden digital signatures for AAX plug-ins.

As of Pro Tools 10.2, support has been added to allow binary-level encryption of AAX DSP algorithms. Please note that this is *NOT* a requirement of AAX DSP plug-ins, and serves only as an additional security measure to protect an algorithm's DLL.

In Pro Tools 11 and above, the Avid Audio Engine (AAE) no longer automatically generates clipping indication for plug-ins. This is because the new engine can operate properly well beyond a unity sample value of 1.0. Thus, it is up to the plug-in itself to set and clear its clip indicators as needed.

Supported in Pro Tools 12 and higher.

Supported in Pro Tools 12 and higher.

Supported in Pro Tools 12 and higher.

Supported in Pro Tools 12.6 and higher.

Supported in Pro Tools 12.8.2 and higher.

Module [AAX_TI_Guide](#)

32 and 64-sample quantum is available in Pro Tools 10.2 and higher

Beginning in Pro Tools 11, AAX DSP algorithms also support optional temporary data spaces that can be described in the Describe module and are shared among all instances on a DSP. This is an alternative to declaring large data blocks on the stack for better memory management and to prevent stack overflows. Please refer to [AAX_IComponentDescriptor::AddTemporaryData\(\)](#) for usage instructions.

Beginning with Pro Tools 10.2, the TI shell supports a "processor affinity" property, which indicates that a DSP ProcessProc should be preferentially loaded onto the same DSP as other instances from the same DLL binary. This is a requirement for some designs that must share global data between different processing configurations.

Note that this property should only be used when absolutely required, as it will constrain the DSP manager and reduce overall DSP plug-in instance counts on the system.

Module [AdditionalFeatures_CurveDisplays](#)

For S6 control surface displays, see [PT-226228](#) and [PT-226227](#) in the [Known Issues](#) page for more information about the requirements listed in this section.

Module advancedTopics_relatedTypes

Pro Tools versions prior to Pro Tools 12.3 do not allow explicit type conversion between types with different product ID values. Beginning in Pro Tools 12.3 both the product ID and the plug-in ID may differ between explicitly related types.

- Pro Tools versions before Pro Tools 12.3 treat deprecated and related type associations identically and do not support type deprecation features
- Media Composer does not support type deprecation features
- VENUE does not support type deprecation features

Module CommonInterface_Algorithm

As of Pro Tools 10.2.1 an algorithm's initialization callback routine will have up to 5 seconds to execute.

Module CommonInterface_FormatSpecification

The plug-in's binary filename must be the same as the outer .aaxplugin bundle name

- On Windows, the plug-in binary (DLL) must use the ".aaxplugin" suffix; i.e. the DLL must use exactly the same name as the outer .aaxplugin folder. On OS X, the plug-in binary does not require a specific suffix.
- On Windows, the plug-in's binary filename (and therefore also the outer .aaxplugin file name) must not contain any spaces. There is a bug in AAE that will prevent binaries with spaces from being loaded properly. This is logged as PTSW-189928.

* `_ACFGetSDKVersion` is required for 64-bit AAX plug-ins only

Module ExamplePlugins

The DemoDelay_DynamicLatencyComp example is compatible with Pro Tools 11.1 and higher.

Chapter 4

Todo List

globalScope> Member [AAX_CAudioInPort](#)

Not used directly by AAX plug-ins

globalScope> Member [AAX_CAudioOutPort](#)

Not used directly by AAX plug-ins

Class [AAX_CChunkDataParser](#)

Update this documentation for AAX

globalScope> Member [AAX_CComponentID](#)

Not used by AAX plug-ins

globalScope> Member [AAX_CCount](#)

Not used by AAX plug-ins

Member [AAX_CEffectDirectData::Controller](#) (**void**)

Change to GetController to match other AAX_CEffect modules

Member [AAX_CEffectDirectData::EffectParameters](#) (**void**)

Change to GetController to match other AAX_CEffect modules

Member [AAX_CEffectGUI::UpdateAllParameters](#) (**void**)

Rename to `UpdateAllParameterViews()` or another name that does not lead to confusion regarding what exactly this method should be doing.

globalScope> Member [AAX_CIndex](#)

Not used by AAX plug-ins (except as [AAX_CFieldIndex](#))

globalScope> Member [AAX_CMeterID](#)

Not used by AAX plug-ins

globalScope> Member [AAX_CMeterPort](#)

Not used directly by AAX plug-ins

Class [AAX_CParameterManager](#)

Should the Parameter Manager return error codes?

Member [AAX_CParameterManager::AddParameter](#) ([AAX_IParameter](#) *param)

Should this method return success/failure code?

Member [AAX_CParameterManager::RemoveAllParameters](#) ()

Should this method return success/failure code?

Member [AAX_CParameterManager::RemoveParameter](#) ([AAX_IParameter](#) *param)

Should this method return success/failure code?

Member [AAX_CParameterManager::RemoveParameterByID](#) ([AAX_CParamID](#) identifier)

Should this method return success/failure code?

Member `AAX_CRangeTaperDelegate< T, RealPrecision >::AAX_CRangeTaperDelegate` (`T *range, double *rangesSteps, long numRanges, bool useSmartRounding=true)`

Document useSmartRounding parameter

Member `AAX_CRangeTaperDelegate< T, RealPrecision >::SmartRound (double value) const`

Document

globalScope > Member `AAX_CSelector`

Clean up usage; currently used for a variety of ID-related values

globalScope > Member `AAX_EParameterOrientationBits`

FLAGGED FOR REVISION

globalScope > Member `AAX_EParameterType`

FLAGGED FOR REMOVAL

globalScope > Member `AAX_eProperty_CanBypass`

This property should always be enabled for AAX plug-ins

Member `AAX_IACFEffetGUI::SetControlHighlightInfo (AAC_CParamID iParameterID, AAC_CBoolean ils=Highlighted, AAC_EHighlightColor iColor)=0`

Document this method

Class `AAX_IACFEffetParameters`

Add documentation for expected error state return values

Member `AAX_IACFEffetParameters::GetParameterOrientation (AAC_CParamID iParameterID, AAC_EParameterOrientation *oParameterOrientation) const =0`

update this documentation

Member `AAX_IACFEffetParameters::GetParameterType (AAC_CParamID iParameterID, AAC_EParameterType *oParameterType) const =0`

The concept of parameter type needs more documentation

Member `AAX_IACFEffetParameters::SetParameterDefaultNormalizedValue (AAC_CParamID iParameterID, double iValue)=0`

THIS IS NOT CALLED FROM HOST. USEFUL FOR INTERNAL USE ONLY?

Member `AAX_IACFEffetParameters::SetParameterNormalizedRelative (AAC_CParamID iParameterID, double iValue)=0`

REMOVE THIS METHOD (?)

NOT CURRENTLY CALLED FROM THE HOST. USEFUL FOR INTERNAL USE ONLY?

NOT CURRENTLY CALLED FROM THE HOST. USEFUL FOR INTERNAL USE ONLY?

Member `AAX_IACFEffetParameters::Uninitialize ()=0`

Docs: When exactly is `AAX_IACFEffetParameters::Uninitialize()` called, and under what conditions?

Member `AAX_IACFEffetParameters::UpdateParameterNormalizedValue (AAC_CParamID iParameterID, double iValue, AAC_EUpdateSource iSource)=0`

FLAGGED FOR CONSIDERATION OF REVISION

Class `AAX_IACFEffetParameters_V2`

Add documentation for expected error state return values

Class `AAX_IACFEffetParameters_V3`

Add documentation for expected error state return values

Class `AAX_IACFEffetParameters_V4`

Add documentation for expected error state return values

Member `AAX_IComponentDescriptor::AddDmaInstance (AAC_CFieldIndex inFieldIndex, AAC_IDma::EMode inDmaMode)=0`

Update the DMA system management such that operation priority can be set arbitrarily

Member `AAX_IComponentDescriptor::AddProcessProc (AAX_IPropertyMap *inProperties, AAX_CSelector *outProcIDs=NULL, int32_t inProcIDsSize=0)=0`

document this parameter Returned array will be NULL-terminated

Member `AAX_IComponentDescriptor::AddProcessProc_Native (AAX_CProcessProc inProcessProc, AAX_IPropertyMap *inProperties=NULL, AAX_CInstanceInitProc inInstanceInitProc=NULL, AAX_CBackgroundProc inBackgroundProc=NULL, AAX_CSelector *outProcID=NULL)=0`

document this parameter

Member `AAX_IComponentDescriptor::AddProcessProc_TI (const char inDLLFileNameUTF8[], const char inProcessProcSymbol[], AAX_IPropertyMap *inProperties, const char inInstanceInitProcSymbol[]=NULL, const char inBackgroundProcSymbol[]=NULL, AAX_CSelector *outProcID=NULL)=0`

document this parameter

Member `AAX_IController::GetCycleCount (AAX_EProperty inWhichCycleCount, AAX_CPropertyValue *outNumCycles) const =0`

PLACEHOLDER - NOT CURRENTLY IMPLEMENTED IN HOST

Member `AAX_IController::SetCycleCount (AAX_EProperty *inWhichCycleCounts, AAX_CPropertyValue *iValues, int32_t numValues)=0`

PLACEHOLDER - NOT CURRENTLY IMPLEMENTED IN HOST

Member `AAX_IDma::IsTransferComplete ()=0`

Clarify return value meaning – ambiguity in documentation

Member `AAX_IParameter::GetType () const =0`

Document use cases for control type

Member `AAX_IParameter::SetTaperDelegate (AAX_ITaperDelegateBase &inTaperDelegate, bool inPreserveValue)=0`

Document this parameter

Module additionalFeatures_Sidechain

Is properties->AddProperty (AAX_eProperty_SupportsSideChainInput, true) even necessary?!?! I believe I saw a p.i. that does not declare this...

Module advancedTopics_parameterUpdates_sequences

Update this section with information about default chunk setting, which is a separate step following the procedure described below.

globalScope> Member `DBToGain (double dB)`

This should be incorporated into parameters' tapers and not called separately

globalScope> Member `GainToDB (double aGain)`

This should be incorporated into parameters' tapers and not called separately

Chapter 5

Legacy Porting Notes

Class [AAX_CEffectGUI](#)

The default implementations in this class are mostly derived from their equivalent implementations in CProcess and CEffectProcess. For additional CProcess-derived implementations, see [AAX_CEffectParameters](#).

Class [AAX_CEffectParameters](#)

The default implementations in this class are mostly derived from their equivalent implementations in CProcess and CEffectProcess. For additional CProcess-derived implementations, see [AAX_CEffectGUI](#).

Member [AAX_CHostProcessor::AnalyzeAudio](#) (`const float *const inAudioIns[], int32_t inAudioInCount, int32_t *ioWindowSize`) `AAX_OVERRIDE`

Ported from AudioSuite's `AnalyzeAudio(bool isMasterBypassed)` method

Member [AAX_CHostProcessor::InitOutputBounds](#) (`int64_t iSrcStart, int64_t iSrcEnd, int64_t *oDstStart, int64_t *oDstEnd`) `AAX_OVERRIDE`

DAE no longer makes use of the `mStartBound` and `mEndBounds` member variables that existed in the legacy RTAS/TDM SDK. Use `oDstStart` and `oDstEnd` instead (preferably by overriding `TranslateOutputBounds()`.)

Member [AAX_CHostProcessor::RenderAudio](#) (`const float *const inAudioIns[], int32_t inAudioInCount, float *const iAudioOuts[], int32_t iAudioOutCount, int32_t *ioWindowSize`) `AAX_OVERRIDE`

This method is a replacement for the AudioSuite `ProcessAudio` method

Class [AAX_CMidiPacket](#)

Corresponds to DirectMidiPacket in the legacy SDK

Class [AAX_CMidiStream](#)

Corresponds to DirectMidiNode in the legacy SDK

globalScope> Member [AAX_eMIDINodeType_Global](#)

Corresponds to RTAS Shared Buffer global nodes in the legacy SDK

globalScope> Member [AAX_eMIDINodeType_LocalInput](#)

Corresponds to RTAS Buffered MIDI input nodes in the legacy SDK

globalScope> Member [AAX_eMIDINodeType_LocalOutput](#)

Corresponds to RTAS Buffered MIDI output nodes in the legacy SDK

globalScope> Member [AAX_eNotificationEvent_ASPreviewState](#)

Replacement for `SetPreviewState()`

globalScope> Member [AAX_ePageTable_EQ_Band_Type](#)

converted from `eDigi_PageTable_EQ_Band_Type` in the legacy SDK

globalScope> Member [AAX_ePageTable_EQ_InCircuitPolarity](#)

converted from `eDigi_PageTable_EQ_InCircuitPolarity` in the legacy SDK

globalScope> Member [AAX_ePageTable_UseAlternateControl](#)

converted from `eDigi_PageTable_UseAlternateControl` in the legacy SDK

globalScope> Member AAX_EParameterType

Values must match unnamed type enum in FicTDMControl.h

globalScope> Member AAX_eParameterType_Continuous

Matches kDAE_ContinuousValues

globalScope> Member AAX_eParameterType_Discrete

Matches kDAE_DiscreteValues

globalScope> Member AAX_EParameterValueInfoSelector

converted from EControlValueInfo in the legacy SDK

globalScope> Member AAX_ePlugInStrings_AllSelectedRegionsAnalysis

Was pluginStrings_AllSelectedRegionsAnalysis in the RTAS/TDM SDK

globalScope> Member AAX_ePlugInStrings_Analysis

Was pluginStrings_Analysis in the RTAS/TDM SDK

globalScope> Member AAX_ePlugInStrings_Bypass

Was pluginStrings_Bypass in the RTAS/TDM SDK

globalScope> Member AAX_ePlugInStrings_ClipName

Was pluginStrings_RegionName in the RTAS/TDM SDK

globalScope> Member AAX_ePlugInStrings_MonoMode

Was pluginStrings_MonoMode in the RTAS/TDM SDK

globalScope> Member AAX_ePlugInStrings_MultiInputMode

Was pluginStrings_MultiInputMode in the RTAS/TDM SDK

globalScope> Member AAX_ePlugInStrings_Process

Was pluginStrings_Process in the RTAS/TDM SDK

globalScope> Member AAX_ePlugInStrings_Progress

Was pluginStrings_Progress in the RTAS/TDM SDK

globalScope> Member AAX_ePlugInStrings_RegionByRegionAnalysis

Was pluginStrings_RegionByRegionAnalysis in the RTAS/TDM SDK

globalScope> Member AAX_EProperty

These property IDs are somewhat analogous to the pluginGestalt system in the legacy SDK, and several AAX_EProperty values correlate directly with a corresponding legacy plug-in gestalt.

To ensure session interchange compatibility, make sure the 4 character IDs for [AAX_eProperty_ManufacturerID](#), [AAX_eProperty_ProductID](#), [AAX_eProperty_PluginID_Native](#), and [AAX_eProperty_PluginID_AudioSuite](#) are identical to the legacy SDK's counterpart.

globalScope> Member AAX_eProperty_AllowPreviewWithoutAnalysis

Was pluginGestalt_AnalyzeOnTheFly

globalScope> Member AAX_eProperty_CanBypass

Was pluginGestalt_CanBypass.

globalScope> Member AAX_eProperty_ContinuousOnly

Was pluginGestalt_ContinuousOnly

globalScope> Member AAX_eProperty_DestinationTrack

Was pluginGestalt_DestinationTrack

globalScope> Member AAX_eProperty_DisablePreview

Was pluginGestalt_DisablePreview

globalScope> Member AAX_eProperty_DoesntIncrOutputSample

Was pluginGestalt_DoesntIncrOutputSample

globalScope > Member AAX_eProperty_ManufacturerID

For legacy plug-in session compatibility, this ID should match the Manufacturer ID used in the corresponding legacy plug-ins.

globalScope > Member AAX_eProperty_MultiInputModeOnly

Was pluginGestalt_MultiInputModeOnly

globalScope > Member AAX_eProperty_NeedsOutputDithered

Was pluginGestalt_NeedsOutputDithered

globalScope > Member AAX_eProperty_OptionalAnalysis

Was pluginGestalt_OptionalAnalysis

globalScope > Member AAX_eProperty_PluginID_AudioSuite

For legacy plug-in session compatibility, this ID should match the Type ID used in the corresponding legacy AudioSuite plug-in Types.

globalScope > Member AAX_eProperty_PluginID_Native

For legacy plug-in session compatibility, this ID should match the Type ID used in the corresponding legacy RTAS plug-in Types.

globalScope > Member AAX_eProperty_PluginID_TI

For legacy plug-in session compatibility, this ID should match the Type ID used in the corresponding legacy TDM plug-in Types.

globalScope > Member AAX_eProperty_ProductID

For legacy plug-in session compatibility, this ID should match the Product ID used in the corresponding legacy plug-in.

globalScope > Member AAX_eProperty_RequestsAllTrackData

Was pluginGestalt_RequestsAllTrackData

globalScope > Member AAX_eProperty_RequiresAnalysis

Was pluginGestalt_RequiresAnalysis

globalScope > Member AAX_eProperty_SupportsProgressDialog

Was pluginGestalt_SupportsProgressDialog

globalScope > Member AAX_eProperty_SupportsSaveRestore

Was pluginGestalt_SupportsSaveRestore

globalScope > Member AAX_eProperty_UsesRandomAccess

Was pluginGestalt_UsesRandomAccess

Class AAX_IACFEffectGUI

In the legacy plug-in SDK, these methods were found in CProcess and CEfectProcess. For additional CProcess methods, see [AAX_IEffectParameters](#).

Member AAX_IACFEffectGUI::SetControlHighlightInfo (AAX_CParamID iParameterID, AAX_CBoolean ils← Highlighted, AAX_EHighlightColor iColor)=0

This method was re-named from `SetControlHighliteInfo()`, its name in the legacy plug-in SDK.

Member AAX_IACFEffectParameters::GetChunkSize (AAX_CTypeID iChunkID, uint32_t *oSize) const =0

In AAX, the value provided by `GetChunkSize()` should *NOT* include the size of the chunk header. The value should *ONLY* reflect the size of the chunk's data.

Member AAX_IACFEffectParameters::GetParameterOrientation (AAX_CParamID iParameterID, AAX_E← ParameterOrientation *oParameterOrientation) const =0

`AAX_IEffectParameters::GetParameterOrientation()` corresponds to the `GetControlOrientation()` method in the legacy RTAS/TDM SDK.

Member AAX_IACFEffectParameters::GetParameterStringFromValue (AAX_CParamID iParameterID, double iValue, AAX_IString *oValueString, int32_t iMaxLength) const =0

This method corresponds to `CProcess::MapControlValToString()` in the RTAS/TDM SDK

Member [AAX_IACFEffectParameters::GetParameterValueFromString](#) (AAX_CParamID iParameterID, double *oValue, const [AAX_IString](#) &iValueString) const =0

This method corresponds to CProcess::MapControlStringToVal() in the RTAS/TDM SDK

Class [AAX_IACFHostProcessor](#)

This interface provides offline processing features analogous to the legacy AudioSuite plug-in architecture

Class [AAX_ICollection](#)

The information in [AAX_ICollection](#) is roughly analogous to the information provided by CProcessGroup in the legacy plug-in library

Class [AAX_IEffectParameters](#)

In the legacy plug-in SDK, these methods were found in CProcess and CEfectProcess. For additional CProcess methods, see [AAX_IEffectGUI](#).

File [AAX_SliderConversions.h](#)

These utilities may be required in order to maintain settings chunk compatibility with plug-ins that were ported from the legacy RTAS/TDM format.

Class [AAX_SPlugInChunkHeader](#)

To ensure compatibility with TDM/RTAS plug-ins whose implementation requires fSize to be equal to the size of the chunk's header plus its data, AAE performs some behind-the-scenes record keeping.

The following actions are only taken for AAX plug-ins, so, e.g., if a chunk is stored by an RTAS or TDM plug-in that reports data+header size in fSize and this chunk is then loaded by the AAX version of the plug-in, the header size will be cached as-is from the legacy plug-in and will be subtracted out before the chunk data is passed to the AAX plug-in. If a chunk is stored by an AAX plug-in and is then loaded by a legacy plug-in, the legacy plug-in will receive the cached plug-in header with fSize equal to the data+header size.

These are the special actions that AAE takes to ensure backwards-compatibility when handling AAX chunk data:

- When AAE retrieves the size of a chunk from an AAX plug-in using [GetChunkSize\(\)](#), it adds the chunk header size to the amount of memory that it allocates for the chunk
- When AAE retrieves a chunk from an AAX plug-in using [GetChunk\(\)](#), it adds the chunk header size to fChunkSize before caching the chunk
- Before calling [SetChunk\(\)](#) or [CompareActiveChunk\(\)](#), AAE subtracts the chunk header size from the cached chunk's header's fChunkSize member

Module [AdditionalFeatures_Meters](#)

The gain-reduction meter handling for AAX plug-ins is different from that for RTAS/TDM plug-ins. AAX plug-ins must invert their gain-reduction meter values manually before reporting these values from the audio processing callback. The AAX host will always thin reported meter data using a "max" operation, and will later invert gain-reduction meter values before they are available to the plug-in GUI or to control surfaces.

Chapter 6

Deprecated List

Member [AAX::FastRndDbI2Int32 \(double iVal\)](#)

globalScope > Member [AAX_CInitPrivateDataProc](#)

globalScope > Member [AAX_CPacketAllocator](#)

globalScope > Member [AAX_EHostMode](#)

This enum is deprecated and will be removed in a future release.

globalScope > Member [AAX_eHostMode_Config](#)

Use [AAX_eHostModeBits_None](#)

globalScope > Member [AAX_eHostMode_Show](#)

Use [AAX_eHostModeBits_Live](#)

globalScope > Member [AAX_ePlugInStrings_PluginFileName](#)

globalScope > Member [AAX_ePlugInStrings_Preview](#)

globalScope > Member [AAX_ePlugInStrings_RegionName](#)

globalScope > Member [AAX_eProcessingState_Start](#)

globalScope > Member [AAX_eProcessingState_Stop](#)

globalScope > Member [AAX_eProperty_AudioBufferLength](#)

Use [AAX_eProperty_DSP_AudioBufferLength](#)

globalScope > Member [AAX_eProperty_Deprecated_Plugin_List](#)

Use [AAX_eProperty_Deprecated_Native_Plugin_List](#) and [AAX_eProperty_Deprecated_DSP_Plugin_List](#) See [AAX_eProperty_PlugInID_RTAS](#) for an example.

globalScope > Member [AAX_eProperty_PlugInID_RTAS](#)

Use [AAX_eProperty_PlugInID_Native](#)

globalScope > Member [AAX_eProperty_StoreXMLPageTablesByType](#)

Use [AAX_eProperty_StoreXMLPageTablesByEffect](#)

Member [AAX_IACFEffectorParameters::UpdateParameterNormalizedRelative](#) (AAx_CParamID iParameterID, double iValue)=0

This is not called from the host. It *may* still be useful for internal calls within the plug-in, though it should only ever be used to update non-automatable parameters. Automatable parameters should always be updated through the [AAx_IParameter](#) interface, which will ensure proper coordination with other automation clients.

Member [AAX_IACFHostServices::Assert](#) (const char *iFile, int32_t iLine, const char *iNote)=0

Legacy version of [AAX_IACFHostServices_V3::HandleAssertFailure\(\)](#) implemented by older hosts

Module [ExamplePlugins](#)

The DemoGain_Delay example is deprecated. See DemoDelay_HostProcessor

Chapter 7

Module Index

7.1 Manual

These pages provide further information about various aspects of AAX.

AAX SDK Manual	??
Getting Started with AAX	??
Core AAX Interface	??
Description callback	??
Real-time algorithm callback	??
Data model interface	??
GUI interface	??
AAX communication protocols	??
AAX Format Specification	??
Additional AAX features	??
Direct data access interface	??
Offline processing interface	??
Hybrid Processing architecture	??
MIDI	??
Plug-in meters	??
Sidechain Inputs	??
Auxiliary Output Stems	??
Direct Memory Access	??
Background processing callback	??
EQ and Dynamics Curve Displays	??
AAX Library features	??
Parameter Manager	??
Taper delegates	??
Display delegates	??
Display delegate decorators	??
Additional Topics	??
Parameter automation	??
Parameter updates	??
Parameter update timing	??
Token protocol	??
Basic parameter update sequences	??
Linked parameters	??
Linked parameter update sequences	??
Plug-in type conversion	??
The Avid Component Framework (ACF)	??
ACF Elements	??
AAX Host Guides	??

Pro Tools Guide	??
Media Composer Guide	??
TI Guide	??
Page Table Guide	??
DigiTrace Guide	??
DSH Guide	??
DTT Guide	??
VENUE Guide	??
Extensions	??
GUI Extensions	??
Monolithic VIs and Effects	??
Other Extensions	??
Supplemental Information	??
Distributing Your AAX Plug-In	??
AAX Interfaces	??
Host Support	??
Known Issues	??
Change Log	??
Example Plug-Ins	??

Chapter 8

Namespace Index

8.1 Namespace List

Here is a list of all namespaces with brief descriptions:

AAX	??
AAX::Exception	
AAX exception classes	??
AAX_ChunkDataParserDefs	
Constants used by ChunkDataParser class	??

Chapter 9

Hierarchical Index

9.1 Class Hierarchy

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

_acfUID	??
AAX_AggregateResult	??
AAX_CAutoreleasePool	??
AAX_CChunkDataParser	??
AAX_CheckedResult	??
AAX_CHostServices	??
AAX_CMidiPacket	??
AAX_CMidiStream	??
AAX_CMutex	??
AAX_Component< aContextType >	??
AAX_CPacket	??
AAX_CPacketDispatcher	??
AAX_CParameterManager	??
AAX_CStringAbbreviations	??
AAX_FastInterpolatedTableLookup< TFLOAT, DFLOAT >	??
AAX_IAutomationDelegate	??
AAX_VAutomationDelegate	??
AAX_ICollection	??
AAX_VCollection	??
AAX_IComponentDescriptor	??
AAX_VComponentDescriptor	??
AAX.IContainer	??
AAX_IPointerQueue< T >	??
AAX_CAtomicQueue< T, S >	??
AAX_IPointerQueue< const TParamValPair >	??
AAX_CAtomicQueue< const TParamValPair, 16 *kSynchronizedParameterQueueSize >	??
AAX_IPointerQueue< TNumberedParamStateList >	??
AAX_CAtomicQueue< TNumberedParamStateList, 256 >	??
AAX_IController	??
AAX_VController	??
AAX_IDescriptionHost	??
AAX_VDescriptionHost	??
AAX_IDisplayDelegateBase	??
AAX_IDisplayDelegate< T >	??
AAX_CBinaryDisplayDelegate< T >	??
AAX_CNumberDisplayDelegate< T, Precision, SpaceAfter >	??

AAX_CStateDisplayDelegate< T >	??
AAX_CStringDisplayDelegate< T >	??
AAX_IDisplayDelegateDecorator< T >	??
AAX_CDecibelDisplayDelegateDecorator< T >	??
AAX_CPercentDisplayDelegateDecorator< T >	??
AAX_CUnitDisplayDelegateDecorator< T >	??
AAX_CUnitPrefixDisplayDelegateDecorator< T >	??
AAX_IDma	??
AAX_IEffectDescriptor	??
AAX_VEffectDescriptor	??
AAX_IFeatureInfo	??
AAX_VFeatureInfo	??
AAX_IHostProcessorDelegate	??
AAX_VHostProcessorDelegate	??
AAX_IHostServices	??
AAX_VHostServices	??
AAX_IMIDIMessageInfoDelegate	??
AAX_IMIDINode	??
AAX_IPacketHandler	??
AAX_CPacketHandler< TWorker >	??
AAX_IPageTable	??
AAX_VPageTable	??
AAX_IParameter	??
AAX_CParameter< T >	??
AAX_CStatelessParameter	??
AAX_IParameterValue	??
AAX_CParameterValue< T >	??
AAX_IPrivateDataAccess	??
AAX_VPrivateDataAccess	??
AAX_IPropertyMap	??
AAX_VPropertyMap	??
AAX_IString	??
AAX_CString	??
AAX_ITaperDelegateBase	??
AAX_ITaperDelegate< T >	??
AAX_CBinaryTaperDelegate< T >	??
AAX_CLinearTaperDelegate< T, RealPrecision >	??
AAX_CLogTaperDelegate< T, RealPrecision >	??
AAX_CPieceWiseLinearTaperDelegate< T, RealPrecision >	??
AAX_CRangeTaperDelegate< T, RealPrecision >	??
AAX_CStateTaperDelegate< T >	??
AAX_ITransport	??
AAX_VTransport	??
AAX_IViewContainer	??
AAX_VViewContainer	??
AAX_Map	??
AAX_Point	??
AAX_Rect	??
AAX_SHybridRenderInfo	??
AAX_SInstrumentPrivateData	??
AAX_SInstrumentRenderInfo	??
AAX_SInstrumentSetupInfo	??
AAX_SPlugInChunk	??
AAX_SPlugInChunkHeader	??

AAX_SPlugInIdentifierTriad	??
AAX_StLock_Guard	??
AAX::Exception::Any	??
AAX::Exception::ResultError	??
CACFUnknown	
AAX_IEffectDirectData	??
AAX_CEffectDirectData	??
AAX_IEffectGUI	??
AAX_CEffectGUI	??
AAX_IEffectParameters	??
AAX_CEffectParameters	??
AAX_CMonolithicParameters	??
AAX_IHostProcessor	??
AAX_CHostProcessor	??
AAX_CChunkDataParser::DataValue	??
IACFPluginDefinition	
AAX_IACFCollection	??
IACFUnknown	??
AAX_IACFAutomationDelegate	??
AAX_IACFComponentDescriptor	??
AAX_IACFComponentDescriptor_V2	??
AAX_IACFComponentDescriptor_V3	??
AAX_IACFController	??
AAX_IACFController_V2	??
AAX_IACFController_V3	??
AAX_IACFDescriptionHost	??
AAX_IACFEffectorDescriptor	??
AAX_IACFEffectorDescriptor_V2	??
AAX_IACFEffectorDirectData	??
AAX_IEffectDirectData	??
AAX_IACFEffectorGUI	??
AAX_IEffectGUI	??
AAX_IACFEffectorParameters	??
AAX_IACFEffectorParameters_V2	??
AAX_IACFEffectorParameters_V3	??
AAX_IACFEffectorParameters_V4	??
AAX_IEffectParameters	??
AAX_IACFFeatureInfo	??
AAX_IACFHostProcessor	??
AAX_IACFHostProcessor_V2	??
AAX_IHostProcessor	??
AAX_IACFHostProcessorDelegate	??
AAX_IACFHostProcessorDelegate_V2	??
AAX_IACFHostProcessorDelegate_V3	??
AAX_IACFHostServices	??
AAX_IACFHostServices_V2	??
AAX_IACFHostServices_V3	??
AAX_IACFPageTable	??
AAX_IACFPageTable_V2	??
AAX_IACFPageTableController	??
AAX_IACFPageTableController_V2	??
AAX_IACFPrivateDataAccess	??
AAX_IACFPropertyMap	??
AAX_IACFPropertyMap_V2	??
AAX_IACFPropertyMap_V3	??

AAX_IACFTransport	??
AAX_IACFTransport_V2	??
AAX_IACFViewContainer	??
AAX_IACFViewContainer_V2	??
IACFDefinition	??
SAutoArray< T >	??

Chapter 10

Class Index

10.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

_acfUID	??
AAX_AggregateResult	??
AAX_CAtomicQueue< T, S >	??
AAX_CAutoreleasePool	??
AAX_CBinaryDisplayDelegate< T >	A binary display format conforming to AAX_IDisplayDelegate	??
AAX_CBinaryTaperDelegate< T >	A binary taper conforming to AAX_ITaperDelegate	??
AAX_CChunkDataParser	Parser utility for plugin chunks	??
AAX_CDecibelDisplayDelegateDecorator< T >	A percent decorator conforming to AAX_IDisplayDelegateDecorator	??
AAX_CEffectDirectData	Default implementation of the AAX_IEffectDirectData interface	??
AAX_CEffectGUI	Default implementation of the AAX_IEffectGUI interface	??
AAX_CEffectParameters	Default implementation of the AAX_IEffectParameters interface	??
AAX_CheckedResult	??
AAX_CHostProcessor	Concrete implementation of the AAX_IHostProcessor interface for non-real-time processing ..	??
AAX_CHostServices	Method access to a singleton implementation of the AAX_IHostServices interface	??
AAX_CLinearTaperDelegate< T, RealPrecision >	A linear taper conforming to AAX_ITaperDelegate	??
AAX_CLogTaperDelegate< T, RealPrecision >	A logarithmic taper conforming to AAX_ITaperDelegate	??
AAX_CMidiPacket	Packet structure for MIDI data	??
AAX_CMidiStream	MIDI stream data structure used by AAX_IMIDINode	??
AAX_CMonolithicParameters	Extension of the AAX_CEffectParameters class for monolithic VIs and effects	??
AAX_CMutex	Mutex with try lock functionality	??
AAX_CNumberDisplayDelegate< T, Precision, SpaceAfter >	A numeric display format conforming to AAX_IDisplayDelegate	??

AAX_Component< aContextType >	Empty class containing type declarations for the AAX algorithm and associated callbacks	??
AAX_CPacket	Container for packet-related data	??
AAX_CPacketDispatcher	Helper class for managing AAX packet posting	??
AAX_CPacketHandler< TWorker >	Callback container used by AAX_CPacketDispatcher	??
AAX_CParameter< T >	Generic implementation of an AAX_IParameter	??
AAX_CParameterManager	A container object for plug-in parameters	??
AAX_CParameterValue< T >	Concrete implementation of AAX_IParameterValue	??
AAX_CPercentDisplayDelegateDecorator< T >	A percent decorator conforming to AAX_IDisplayDelegateDecorator	??
AAX_CPieceWiseLinearTaperDelegate< T, RealPrecision >	A piece-wise linear taper conforming to AAX_ITaperDelegate	??
AAX_CRangeTaperDelegate< T, RealPrecision >	A piecewise-linear taper conforming to AAX_ITaperDelegate	??
AAX_CStateDisplayDelegate< T >	A generic display format conforming to AAX_IDisplayDelegate	??
AAX_CStatelessParameter	A stateless parameter implementation	??
AAX_CStateTaperDelegate< T >	A linear taper conforming to AAX_ITaperDelegate	??
AAX_CString	A generic AAX string class with similar functionality to <code>std::string</code>	??
AAX_CStringAbbreviations	Helper class to store a collection of name abbreviations	??
AAX_CStringDisplayDelegate< T >	A string, or list, display format conforming to AAX_IDisplayDelegate	??
AAX_CUnitDisplayDelegateDecorator< T >	A unit type decorator conforming to AAX_IDisplayDelegateDecorator	??
AAX_CUnitPrefixDisplayDelegateDecorator< T >	A unit prefix decorator conforming to AAX_IDisplayDelegateDecorator	??
AAX_FastInterpolatedTableLookup< TFLOAT, DFLOAT >	??
AAX_IACFAutomationDelegate	Versioned interface allowing an AAX plug-in to interact with the host's automation system	??
AAX_IACFCollection	Versioned interface to represent a plug-in binary's static description	??
AAX_IACFComponentDescriptor	Versioned description interface for an AAX plug-in algorithm callback	??
AAX_IACFComponentDescriptor_V2	Versioned description interface for an AAX plug-in algorithm callback	??
AAX_IACFComponentDescriptor_V3	Versioned description interface for an AAX plug-in algorithm callback	??
AAX_IACFController	Interface for the AAX host's view of a single instance of an effect. Used by both clients of the AAXHost and by effect components	??
AAX_IACFController_V2	Interface for the AAX host's view of a single instance of an effect. Used by both clients of the AAXHost and by effect components.	??
AAX_IACFController_V3	Interface for the AAX host's view of a single instance of an effect. Used by both clients of the AAXHost and by effect components.	??
AAX_IACFDescriptionHost	??

AAX_IACFEffectDescriptor	Versioned interface for an AAX_IEffectDescriptor	??
AAX_IACFEffectDescriptor_V2	Versioned interface for an AAX_IEffectDescriptor	??
AAX_IACFEffectDirectData	Optional interface for direct access to a plug-in's alg memory	??
AAX_IACFEffectGUI	The interface for a AAX Plug-in's GUI (graphical user interface)	??
AAX_IACFEffectParameters	The interface for an AAX Plug-in's data model	??
AAX_IACFEffectParameters_V2	Supplemental interface for an AAX Plug-in's data model	??
AAX_IACFEffectParameters_V3	Supplemental interface for an AAX Plug-in's data model	??
AAX_IACFEffectParameters_V4	Supplemental interface for an AAX Plug-in's data model	??
AAX_IACFFeatureInfo	??
AAX_IACFHostProcessor	Versioned interface for an AAX host processing component	??
AAX_IACFHostProcessor_V2	Supplemental interface for an AAX host processing component	??
AAX_IACFHostProcessorDelegate	Versioned interface for host methods specific to offline processing	??
AAX_IACFHostProcessorDelegate_V2	Versioned interface for host methods specific to offline processing	??
AAX_IACFHostProcessorDelegate_V3	Versioned interface for host methods specific to offline processing	??
AAX_IACFHostServices	Versioned interface to diagnostic and debugging services provided by the AAX host	??
AAX_IACFHostServices_V2	V2 of versioned interface to diagnostic and debugging services provided by the AAX host	??
AAX_IACFHostServices_V3	V3 of versioned interface to diagnostic and debugging services provided by the AAX host	??
AAX_IACFPageTable	Versioned interface to the host's representation of a plug-in instance's page table	??
AAX_IACFPageTable_V2	Versioned interface to the host's representation of a plug-in instance's page table	??
AAX_IACFPageTableController	Interface for host operations related to the page tables for this plug-in	??
AAX_IACFPageTableController_V2	Interface for host operations related to the page tables for this plug-in.	??
AAX_IACFPrivateDataAccess	Interface for the AAX host's data access functionality	??
AAX_IACFPropertyMap	Versioned interface for an AAX_IPropertyMap	??
AAX_IACFPropertyMap_V2	Versioned interface for an AAX_IPropertyMap	??
AAX_IACFPropertyMap_V3	Versioned interface for an AAX_IPropertyMap	??
AAX_IACFTransport	Versioned interface to information about the host's transport state	??
AAX_IACFTransport_V2	Versioned interface to information about the host's transport state	??
AAX_IACFViewContainer	Interface for the AAX host's view of a single instance of an effect. Used by both clients of the host app and by effect components	??

AAX_IACFViewContainer_V2	Supplemental interface for the AAX host's view of a single instance of an effect. Used by both clients of the host app and by effect components	??
AAX_IAutomationDelegate	Interface allowing an AAX plug-in to interact with the host's event system	??
AAX_ICollection	Interface to represent a plug-in binary's static description	??
AAX_IComponentDescriptor	Description interface for an AAX plug-in component	??
AAX.IContainer	??
AAX_IController	Interface for the AAX host's view of a single instance of an effect. Used by both clients of the AAX host and by effect components	??
AAX_IDescriptionHost	??
AAX_IDisplayDelegate< T >	Classes for parameter value string conversion.	??
AAX_IDisplayDelegateBase	Defines the display behavior for a parameter	??
AAX_IDisplayDelegateDecorator< T >	The base class for all concrete display delegate decorators	??
AAX_IDma	Cross-platform interface for access to the host's direct memory access (DMA) facilities	??
AAX_IEffectDescriptor	Description interface for an effect's (plug-in type's) components	??
AAX_IEffectDirectData	The interface for a AAX Plug-in's direct data interface	??
AAX_IEffectGUI	The interface for a AAX Plug-in's user interface	??
AAX_IEffectParameters	The interface for an AAX Plug-in's data model	??
AAX_IFeatureInfo	??
AAX_IHostProcessor	Base class for the host processor interface	??
AAX_IHostProcessorDelegate	Versioned interface for host methods specific to offline processing	??
AAX_IHostServices	Interface to diagnostic and debugging services provided by the AAX host	??
AAX_IMIDIMessageInfoDelegate	??
AAX_IMIDINode	Interface for accessing information in a MIDI node	??
AAX_IPacketHandler	Callback container used by AAX_CPacketDispatcher	??
AAX_IPageTable	Interface to the host's representation of a plug-in instance's page table	??
AAX_IParameter	The base interface for all normalizable plug-in parameters	??
AAX_IParameterValue	An abstract interface representing a parameter value of arbitrary type	??
AAX_IPointerQueue< T >	??
AAX_IPrivateDataAccess	Interface to data access provided by host to plug-in	??
AAX_IPropertyMap	Generic plug-in description property map	??
AAX_IString	A simple string container that can be passed across a binary boundary. This class, for simplicity, is not versioned and thus can never change	??
AAX_ITaperDelegate< T >	Classes for conversion to and from normalized parameter values.	??

AAX_ITaperDelegateBase	Defines the taper conversion behavior for a parameter	??
AAX_ITransport	Interface to information about the host's transport state	??
AAX_IViewContainer	Interface for the AAX host's view of a single instance of an effect. Used both by clients of the AAX host and by effect components	??
AAX_Map		??
AAX_Point	Data structure representing a two-dimensional coordinate point	??
AAX_Rect	Data structure representing a rectangle in a two-dimensional coordinate plane	??
AAX_SHybridRenderInfo	Hybrid render processing context	??
AAX_SInstrumentPrivateData	Utility struct for AAX_CMonolithicParameters	??
AAX_SInstrumentRenderInfo	Information used to parameterize AAX_CMonolithicParameters::RenderAudio()	??
AAX_SInstrumentSetupInfo	Information used to describe the instrument	??
AAX_SPlugInChunk	Plug-in chunk header + data	??
AAX_SPlugInChunkHeader	Plug-in chunk header	??
AAX_SPlugInIdentifierTriad	Plug-in Identifier Triad	??
AAX_StLock_Guard	Helper class for working with mutex	??
AAX_VAutomationDelegate	Version-managed concrete automation delegate class	??
AAX_VCollection	Version-managed concrete AAX_ICollection class	??
AAX_VComponentDescriptor	Version-managed concrete AAX_IComponentDescriptor class	??
AAX_VController	Version-managed concrete Controller class	??
AAX_VDescriptionHost		??
AAX_VEffectDescriptor	Version-managed concrete AAX_IEffectDescriptor class	??
AAX_VFeatureInfo		??
AAX_VHostProcessorDelegate	Version-managed concrete Host Processor delegate class	??
AAX_VHostServices	Version-managed concrete AAX_IHostServices class	??
AAX_VPageTable	Version-managed concrete AAX_IPageTable class	??
AAX_VPrivateDataAccess	Version-managed concrete AAX_IPrivateDataAccess class	??
AAX_VPropertyMap	Version-managed concrete AAX_IPropertyMap class	??
AAX_VTransport	Version-managed concrete AAX_ITransport class	??
AAX_VViewContainer	Version-managed concrete AAX_IViewContainer class	??
AAX::Exception::Any		??
AAX_CChunkDataParser::DataValue		??

IACFDefinition	Publicly inherits from IACFUnknown.This abstract interface is used to indentify all of the plug-in components in the host	??
IACFUnknown	COM compatible IUnknown C++ interface	??
AAX::Exception::ResultError		??
SAutoArray< T >		??

Chapter 11

File Index

11.1 File List

Here is a list of all files with brief descriptions:

AAX.h	Various utility definitions for AAX	??
AAX_Alignment.h	Alignment malloc and free methods for optimization	??
AAX_Assert.h	Declarations for cross-platform AAX_ASSERT, AAX_TRACE and related facilities	??
AAX_Atomic.h	Atomic operation utilities	??
AAX_Callbacks.h	AAX callback prototypes and ProcPtr definitions	??
AAX_CAtomicQueue.h	Atomic, non-blocking queue	??
AAX_CAutoreleasePool.h	Autorelease pool helper utility	??
AAX_CBinaryDisplayDelegate.h	A binary display delegate	??
AAX_CBinaryTaperDelegate.h	A binary taper delegate	??
AAX_CChunkDataParser.h	Parser utility for plugin chunks	??
AAX_CDecibelDisplayDelegateDecorator.h	A decibel display delegate	??
AAX_CEffectDirectData.h	A default implementation of the AAX_IEffectDirectData interface	??
AAX_CEffectGUI.h	A default implementation of the AAX_IEffectGUI interface	??
AAX_CEffectParameters.h	A default implementation of the AAX_IeffectParameters interface	??
AAX_CHostProcessor.h	Concrete implementation of the AAX_IHostProcessor interface for non-real-time processing	??
AAX_CHostServices.h	Concrete implementation of the AAX_IHostServices interface	??
AAX_CLinearTaperDelegate.h	A linear taper delegate	??
AAX_CLogTaperDelegate.h	A log taper delegate	??
AAX_CMonolithicParameters.cpp		??

AAX_CMonolithicParameters.h	A convenience class extending AAX_CEffectParameters for monolithic instruments	??
AAX_CMutex.h	Mutex	??
AAX_CNumberDisplayDelegate.h	A number display delegate	??
AAX_CommonConversions.h	??
AAX_Constants.h	Signal processing constants	??
AAX_CPacketDispatcher.h	Helper classes related to posting AAX packets and handling parameter update events	??
AAX_CParameter.h	Generic implementation of an AAX_IParameter	??
AAX_CParameterManager.h	A container object for plug-in parameters	??
AAX_CPercentDisplayDelegateDecorator.h	A percent display delegate decorator	??
AAX_CPieceWiseLinearTaperDelegate.h	A piece-wise linear taper delegate	??
AAX_CRangeTaperDelegate.h	A range taper delegate decorator	??
AAX_CStateDisplayDelegate.h	A state display delegate	??
AAX_CStateTaperDelegate.h	A state taper delegate (similar to a linear taper delegate.)	??
AAX_CString.h	A generic AAX string class with similar functionality to std::string	??
AAX_CStringDisplayDelegate.h	A string display delegate	??
AAX_CUnitDisplayDelegateDecorator.h	A unit display delegate decorator	??
AAX_CUnitPrefixDisplayDelegateDecorator.h	A unit prefix display delegate decorator	??
AAX_Denormal.h	Signal processing utilities for denormal/subnormal floating point numbers	??
AAX_EndianSwap.h	Utility functions for byte-swapping. Used by AAX_CChunkDataParser	??
AAX.Enums.h	Utility functions for byte-swapping. Used by AAX_CChunkDataParser	??
AAX_Errors.h	Definitions of error codes used by AAX plug-ins	??
AAX_Exception.h	AAX SDK exception classes and utilities	??
AAX_Exports.cpp	??
AAX_FastInterpolatedTableLookup.h	A set of functions that provide lookup table functionality. Not necessarily optimized for TI, but used internally	??
AAX_FastInterpolatedTableLookup.hpp	??
AAX_FastPow.h	Set of functions to optimize pow	??
AAX_GUITypes.h	Constants and other definitions used by AAX plug-in GUIs	??
AAX_IACFAutomationDelegate.h	Versioned interface allowing an AAX plug-in to interact with the host's automation system	??
AAX_IACFCollection.h	Versioned interface to represent a plug-in binary's static description	??
AAX_IACFComponentDescriptor.h	Versioned description interface for an AAX plug-in algorithm callback	??

AAX_IACFController.h	Interface for the AAX host's view of a single instance of an effect. Used by both clients of the AAXHost and by effect components	??
AAX_IACFDscriptionHost.h	??
AAX_IACFEffectDescriptor.h	Versioned interface for an AAX_IEffectDescriptor	??
AAX_IACFEffectDirectData.h	The direct data access interface that gets exposed to the host application	??
AAX_IACFEffectGUI.h	The GUI interface that gets exposed to the host application	??
AAX_IACFEffectParameters.h	The data model interface that is exposed to the host application	??
AAX_IACFFeatureInfo.h	??
AAX_IACFHostProcessor.h	The host processor interface that is exposed to the host application	??
AAX_IACFHostProcessorDelegate.h	??
AAX_IACFHostServices.h	??
AAX_IACFPageTable.h	??
AAX_IACFPageTableController.h	??
AAX_IACFPrivateDataAccess.h	Interface for the AAX host's data access functionality	??
AAX_IACFPropertyMap.h	Versioned interface for an AAX_IPropertyMap	??
AAX_IACFTransport.h	Interface for the AAX Transport data access functionality	??
AAX_IACFViewContainer.h	Interface for the AAX host's view of a single instance of an effect. Used by both clients of the AAXHost and by effect components	??
AAX_IAutomationDelegate.h	Interface allowing an AAX plug-in to interact with the host's automation system	??
AAX_ICollection.h	Interface to represent a plug-in binary's static description	??
AAX_IComponentDescriptor.h	Description interface for an AAX plug-in algorithm	??
AAX.IContainer.h	Abstract container interface	??
AAX_IController.h	Interface for the AAX host's view of a single instance of an effect	??
AAX_IDescriptionHost.h	??
AAX_IDisplayDelegate.h	Defines the display behavior for a parameter	??
AAX_IDisplayDelegateDecorator.h	The base class for all concrete display delegate decorators	??
AAX_IDma.h	Cross-platform interface for access to the host's direct memory access (DMA) facilities	??
AAX_IEffectDescriptor.h	Description interface for an effect's (plug-in type's) components	??
AAX_IEffectDirectData.h	Optional interface for direct access to alg memory	??
AAX_IEffectGUI.h	The interface for a AAX Plug-in's user interface	??
AAX_IEffectParameters.h	The interface for an AAX Plug-in's data model	??
AAX_IFeatureInfo.h	??
AAX_IHostProcessor.h	Base class for the host processor interface which is extended by plugin code	??
AAX_IHostProcessorDelegate.h	Interface allowing plug-in's HostProcessor to interact with the host's side	??

AAX_IHostServices.h	Various host services	??
AAX_IMIDINode.h	Declaration of the base MIDI Node interface	??
AAX_Init.h	AAX library implementations of required plug-in initialization, registration, and tear-down methods	??
AAX_IPageTable.h	??
AAX_IParameter.h	The base interface for all normalizable plug-in parameters	??
AAX_IPointerQueue.h	Abstract interface for a basic FIFO queue of pointers-to-objects	??
AAX_IPrivateDataAccess.h	Interface to data access provided by host to plug-in	??
AAX_IPropertyMap.h	Generic plug-in description property map	??
AAX_IString.h	An AAX string interface	??
AAX_ITaperDelegate.h	Defines the taper conversion behavior for a parameter	??
AAX_ITransport.h	The interface for query ProTools transport information	??
AAX_IViewContainer.h	Interface for the AAX host's view of a single instance of an effect	??
AAX_Map.h	??
AAX_MIDILogging.cpp	??
AAX_MIDILogging.h	Utilities for logging MIDI data	??
AAX_MIDIUtilities.h	Utilities for managing MIDI data	??
AAX_MiscUtils.h	Miscellaneous signal processing utilities	??
AAX_PageTableUtilities.h	??
AAX_PlatformOptimizationConstants.h	Constants descriptor..	??
AAX_PluginBundleLocation.h	Utilities for interacting with the host OS	??
AAX_PopStructAlignment.h	Resets (pops) the struct alignment to its previous value	??
AAX_Properties.h	Contains IDs for properties that can be added to an AAX_IPropertyMap	??
AAX_Push2ByteStructAlignment.h	Set the struct alignment to 2-byte. This file will throw an error on platforms that do not support 2-byte alignment (i.e. TI DSPs)	??
AAX_Push4ByteStructAlignment.h	Set the struct alignment to 4-byte	??
AAX_Push8ByteStructAlignment.h	Set the struct alignment to 8-byte	??
AAX_Quantize.h	Quantization utilities	??
AAX_RandomGen.h	Functions for calculating pseudo-random numbers	??
AAX_SampleRateUtils.h	Description	??
AAX_SliderConversions.h	Legacy utilities for converting parameter values to and from the normalized full-scale 32-bit fixed domain that was used for RTAS/TDM plug-ins	??

AAX_StringUtilities.h	Various string utility definitions for AAX Native	??
AAX_StringUtilities.hpp		??
AAX_UIDs.h	Unique identifiers for AAX/ACF interfaces	??
AAX_UtilsNative.h	Various utility definitions for AAX Native	??
AAX_VAutomationDelegate.h	Version-managed concrete AutomationDelegate class	??
AAX_VCollection.h	Version-managed concrete Collection class	??
AAX_VComponentDescriptor.h	Version-managed concrete ComponentDescriptor class	??
AAX_VController.h	Version-managed concrete Controller class	??
AAX_VDescriptionHost.h		??
AAX_VEffectDescriptor.h	Version-managed concrete EffectDescriptor class	??
AAX_Version.h	Version stamp header for the AAX SDK	??
AAX_VFeatureInfo.h		??
AAX_VHostProcessorDelegate.h	Version-managed concrete HostProcessorDelegate class	??
AAX_VHostServices.h	Version-managed concrete HostServices class	??
AAX_VPageTable.h		??
AAX_VPrivateDataAccess.h	Version-managed concrete PrivateDataAccess class	??
AAX_VPropertyMap.h	Version-managed concrete PropertyMap class	??
AAX_VTransport.h	Version-managed concrete Transport class	??
AAX_VViewContainer.h	Version-managed concrete ViewContainer class	??

Chapter 12

Module Documentation

12.1 AAX SDK Manual

12.1.1

12.1.2 Welcome to AAX

Select the "Manual" tab to see a full list of documentation pages, or choose from the topics below.

Note

The search function only includes indexing of code symbols and page titles. To search for specific text strings in the AAX SDK manual it is best to use a text search tool such as grep or FINDSTR on the AAX SDK directory or search for the desired text within the PDF version of the AAX SDK documentation.

12.1.2.1 The basics

- For a general overview of AAX and a walk-through of the DemoGain example plug-in, see [Getting Started with AAX](#)
- If you are new to Pro Tools then you may also want to read through the first few sections of the [Pro Tools Guide](#)
- The [sample plug-ins](#) provide examples of both basic and advanced AAX features
- If you want to find out more about the basic structure of AAX plug-ins, see [Core AAX Interface](#)
- To learn more about audio processing in AAX, see [Real-time algorithm callback](#)
- To learn about adding parameters and controls to your plug-in, see [Data model interface](#)
- For more information about plug-in configuration and initial set-up, see [Description callback](#)
- Ready to ship your new plug-in? See [Distributing Your AAX Plug-In](#) for information about finalizing and distributing your AAX products

12.1.2.2 More topics

- To see the interface that an AAX plug-in's host-based modules use to interact with the host, see [AAX_IController](#)
- To find out more about how different modules in an AAX plug-in communicate with one another, see [AAX communication protocols](#)

- For information about creating advanced non-real-time AAX plug-ins, see [Offline processing interface](#)
- For more information about implementing custom parameter and control behavior, see [Taper delegates](#) and [Display delegates](#)
- To find out how to create and optimize an AAX plug-in for Avid's HDX platform, see the [TI Guide](#)
- You can find signal processing utility classes and functions available for optimizing on Native and DSP in the /TI/SignalProcessing folder

12.1.2.3 Test tools and utilities

- For information about the tool for testing basic functionality of your plug-in, see [DSH Guide](#)
- To learn how to automate different test scenarios for DSH, see [DTT Guide](#)

12.1.2.4 Supplemental information

- [Example Plug-Ins](#)
- [Change Log](#)
- [Host Support](#)
- [Known Issues](#)

12.1.3 SDK Folder hierarchy

Documentation

SDK documentation

ExamplePlugins

Example plug-in projects [More information](#)

Extensions

Demonstrations of how to extend the AAX SDK, for example by incorporating third-party GUI frameworks into AAX plug-ins. [More information](#)

Interfaces

Interface headers and other resources required for use of the AAX SDK library

Libs

Source code for the AAX SDK library, a collection of default implementations and utility classes for use in all AAX plug-ins

TI

Various resources for use with TI's Code Composer Studio IDE and Avid's TI testing toolset

Utilities

Common SDK utilities and resources

12.1.4 Contacting Avid

Your personal [avid user account](#) is your hub for AAX Toolkit services and developer support.

Log in at [avid.com](#) for access to the full range of tools and services provided to AAX developers, including the AAX [developer forum](#). If you have any questions on the AAX SDK documentation or require support with AAX development, we encourage you to post them to the forum as your first line of inquiry.

If you have time-sensitive or critical support inquiries, contact the AAX development team directly at devservices@avid.com. Any AAX questions sent to this alias will be promptly addressed by the most appropriate contact here at Avid.

If you require NFR (Not For Resale) licenses to Avid software for AAX development please send an e-mail to devauth@avid.com with "License Request" in the subject.

If you require access to the digital signing toolkit from PACE Anti-Piracy, Inc. for compatibility with Pro Tools then please follow the instructions [here](#).

The following chart describes these and other ways of connecting with Avid to take advantage of the services provided to AAX developers:

Documents

- [Getting Started with AAX](#)

A brief introduction to AAX.

- [Core AAX Interface](#)

Main classes, callbacks, and format specification details for a standard AAX plug-in.

- [Additional AAX features](#)

How to use additional features and functionality supported by AAX.

- [AAX Library features](#)

AAX Library core support for the AAX interface

- [Additional Topics](#)

Additional information about the AAX design.

- [AAX Host Guides](#)

Documentation for specific AAX host environments.

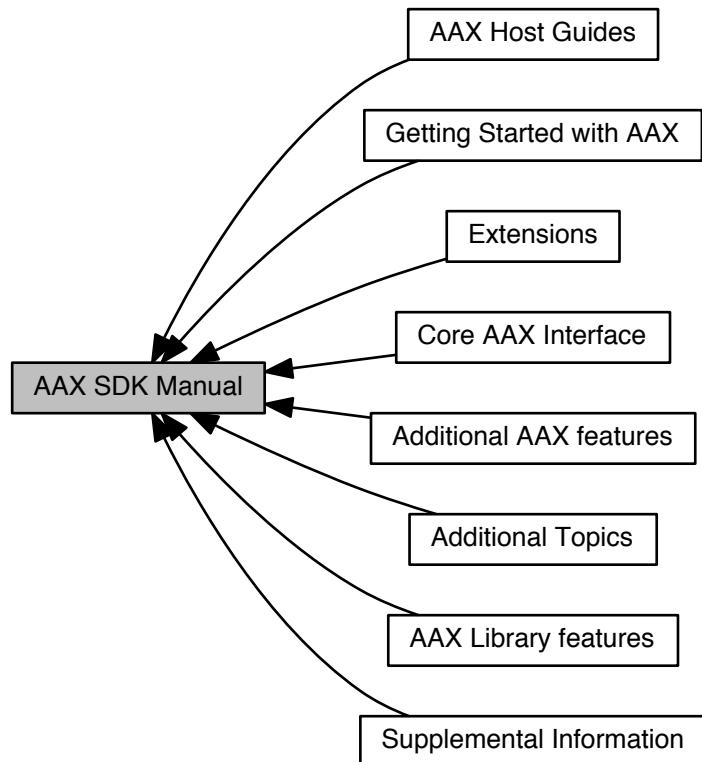
- [Extensions](#)

Extensions to the AAX SDK.

- [Supplemental Information](#)

Supplemental documents beyond the scope of the AAX SDK.

Collaboration diagram for AAX SDK Manual:



12.2 Getting Started with AAX

A brief introduction to AAX.

12.2.1 Contents

- [Welcome](#)
- [AAX Design Overview](#)
- [DemoGain Example](#)

12.2.2 Welcome

Welcome to AAX! This guide is designed to introduce you to the fundamental concepts of AAX. By the end of this guide you will understand:

- The purpose of AAX
- The basic components of an AAX plug-in
- The structure of the DemoGain example plug-in

If you would prefer to skip the documentation and jump straight to the code then we recommend jumping to [DemoGain Example](#) below. This section will walk you through the core parts of an AAX plug-in using the DemoGain plug-in as an example.

12.2.3 AAX Design Overview

12.2.3.1 Architecture Philosophy

The purpose of the AAX architecture is to provide an *easy-to-use* framework for the development of *high-performance audio plug-ins* that can run on a *variety of platforms*, both present and future, with a *high level of code re-use*.

12.2.3.2 Design Attributes

AAX incorporates a flexible, decoupled component hierarchy, including:

1. A relocatable C-style callback that performs the plug-in's real-time audio processing algorithm
2. A collection of supporting C++ objects that manage the plug-in's data, GUI, and any other non-real-time information
3. A "description" method that statically describes the plug-in's layout and compatibility requirements to the host.

This flexible design facilitates optimal performance and portability; AAX is 64-bit ready and is designed to work well in both host-based (Native), accelerated (DSP), and future (e.g. embedded) environments. Importantly, plug-ins running in each of these environments can use the same code base, and porting to new platforms should not require much more than a re-compile. To satisfy these portability requirements, AAX must be decoupled into three parts, the GUI, data model, and algorithm.

12.2.3.3 Component Structure

The core component structure of an AAX plug-in involves data model, GUI, description, and algorithm modules. The design involves a mixture of C++ objects (data model and GUI modules) and C-style callbacks (algorithm and description modules.)

Figure shows basic object ownership and composition in an AAX plug-in. The purple components are provided as part of the AAX SDK while the gray components are written as needed by the plug-in developer. The classes above the dotted line are pure virtual interfaces, while the classes below the dotted line are concrete implementations.

Figure 1: Core AAX interface design.

As you can see, the plug-in's [data model](#) and [GUI](#) are written as C++ objects inherited from SDK interfaces, while its [algorithm](#) and [Description](#) methods are implemented as simple callbacks. This basic model may be expanded by attaching additional modules, such as the [Host Processor](#) module used by advanced non-real-time plug-ins. In this section, however, we will be considering only the four core interfaces and modules.

12.2.3.4 Algorithm

The most fundamental, and most important, component of any audio plug-in is its processing algorithm. Due to the design requirements of an AAX plug-in, this component must meet several constraints in order to be compatible with accelerated platforms:

1. It must be possible to build the algorithm as a compatible component on a variety of platforms
2. It must be possible for the host to re-locate the algorithm in memory without affecting the algorithm's state
3. The algorithm must be separable from the rest of its plug-in, e.g. into a different memory space
4. The algorithm must be as efficient as possible to call.

To satisfy these constraints, AAX uses a decoupled C-style callback function. State information within the function is obtained through the context, a custom data structure that contains everything from the "outside world" that is relevant to the algorithm. The context includes information such as audio buffers and coefficient packets, which are provided according to a static set of data routing definitions that we will describe further in the next chapter. The host is responsible for ensuring that this structure is up-to-date whenever the algorithm callback is entered.

See also

[Real-time algorithm callback](#)

12.2.3.5 Data Model

The [AAX_IEffectParameters](#) interface represents the data model portion of your plug-in. The AAX host interacts with your plug-in's data model via this interface, which includes methods that store and update of your plug-in's internal data.

In your plug-in's data model implementation, you will be responsible for creating the plug-in's parameters and defining how the plug-in will respond when these parameters are changed via control updates or preset loads. In order for information to be routed from the data model to the plug-in's algorithm, the parameters that are created here must be registered with the host in the plug-in's Description callback (see below).

The data model is composed with [AAX_IController](#), an interface that provides access to the host. This interface provides a means of querying information from the host such as stem format or sample rate, and is also responsible for communication between the data model and the (decoupled) algorithm.

You will most likely inherit your custom data model's implementation from [AAX_CEffectParameters](#), a default implementation of the [AAX_IEffectParameters](#) interface. This class provides basic functionality such as adding custom parameters, setting control values, restoring state, generating coefficients, etc., which you can override and customize as needed.

See also

[Data model interface](#)

12.2.3.6 GUI Interface

The [AAX_IEffectGUI](#) interface contains the plug-in's GUI methods. This interface is decoupled from the plug-in's data model, allowing AAX to support distributed architectures that incorporate remote GUIs.

The GUI is also composed with [AAX_IController](#), from which references to the plug-in's other components, such as its data model interface, may be retrieved.

The AAX SDK includes a set of GUI extensions to help you get started implementing your plug-ins' GUIs. These extensions include both native drawing formats and suggested integrations with third-party graphics frameworks. Although the SDK does not include any actual graphics frameworks, these extensions provide examples of how you can incorporate your chosen GUI framework with [AAX](#).

See also

[GUI interface](#)

12.2.3.7 Describe

A plug-in's Describe callback ties all the plug-in's components together and registers the plug-in with the host. In this callback, your plug-in defines a set of algorithm callbacks, data connections, and static plug-in properties

In order to route data to the plug-in's algorithm, Describe includes a description of the algorithm's context structure. This description involves a set of port definitions, which can be "hard-wired" to receive data from the host (such as audio buffers,) from the plug-in's data model (such as packets of coefficients,) or even from past calls to the algorithm (private, persistent algorithm state.)

Once these connections are made, Describe passes the host a populated description interface and returns. In the next section we will demonstrate how all these interfaces interact with one another by examining a sample plug-in.

See also

[Description callback](#)

12.2.3.8 Controller

There is one additional core component to any AAX plug-in, the Controller. The plug-in does not implement this component - rather, the Controller is an interface that provides access to various facilities provided by the host, as well as to the plug-in's other modules.

12.2.4 DemoGain Example

The AAX SDK includes a basic example plug-in named DemoGain. In this section we will examine this plug-in to show how the various core modules in an AAX plug-in interact with one another. We will focus in particular on how the DemoGain's "gain" parameter is defined, routed to the algorithm, and used to apply a gain effect to audio samples.

For a description of all the example plug-ins included in the SDK, see the [Example Plug-Ins](#) page.

12.2.4.1 Getting Started

12.2.4.1.1 Build the AAX Library

If this is your first time opening the AAX SDK then your first step should be to build the AAX Library project. The AAX Library is a static library containing default implementations of the AAX interface and convenience classes

designed to make AAX development easy. All of the SDK example plug-ins link to this library, and your plug-ins should too.

Open the AAX Library project for your chosen IDE from the Libs/AAXLibrary directory and build the library. Now you are ready to build plug-ins!

12.2.4.1.2 Open and build DemoGain

The DemoGain project is located in ExamplePlugIns/DemoGain. Once you have built the AAX Library project you will be able to successfully build DemoGain.

Note

To run DemoGain or other example plug-ins in Pro Tools 10 you must change the `AAX_ePlugInCategory_Example` category to `AAX_ePlugInCategory_Dynamics` in the plug-in's `Describe` function. The "example" category is not supported in Pro Tools 10.

The DemoGain plug-in that you just built is not digitally signed, so it will only run in a developer build of Pro Tools or in another host that does not require digital signatures for AAX plug-ins. If you have a developer build of Pro Tools then you can now test the plug-in by installing it and launching Pro Tools. See [Install directories](#) section in the [Pro Tools Guide](#) document for more information about installing AAX plug-ins for Pro Tools.

12.2.4.1.3 Now what?

A good starting point for understanding a plug-in is to understand its parameters. In DemoGain, as in most AAX plug-ins, the plug-in's parameters define its data model state, and therefore the plug-in's parameters provide the fundamental connection between user interactions and audio processing.

In the remainder of this section we will examine the gain parameter in DemoGain to understand how this parameter is defined, hooked up, modified, and eventually applied to the plug-in algorithm. In this way we will "visit" each of the core design components in DemoGain.

12.2.4.2 Overview of Parameter Creation

To create a plug-in parameter, the following steps are required:

1. Data model: Create your parameter
 - (a) Create a new parameter object
 - (b) Register the parameter with the Packet Dispatcher
 - (c) Create an update callback that converts the raw parameter value into a packet of processed coefficients
2. Algorithm: Add new coefficients to the algorithm's context structure
 - (a) Create a new field for incoming coefficient packets
 - (b) Generate a *field ID* to reference the new member
 - (c) Add the new coefficient to the plug-in's algorithm code
3. Describe: Connect the parameter throughout the plug-in
 - (a) Add a new Data Input Port to the algorithm via the component descriptor interface
 - (b) Add a connection between the parameter's *packet ID* and its coefficients' *field ID*
4. GUI: Add a control
 - (a) Create a GUI widget that can update the parameter's state
 - (b) Add logic to notify the data model when the GUI is edited
 - (c) Add logic to update the GUI when the data model state changes

The following sections discuss these steps in greater detail, making use of the concrete implementation in [AAX_CEffectParameters](#).

12.2.4.3 Data Model: Create Your Parameter

DemoGain's data model inherits its functionality from [AAX_CEffectParameters](#) (the default implementation of [AAX_IEffectParameters](#)). The `DemoGain_Parameters.h` and `DemoGain_Parameters.cpp` source files comprise the source code for DemoGain's data model.

During plug-in initialization, `DemoGain_Parameters::EffectInit()` creates the plug-in's custom parameters. A look inside of the .cpp file shows how this is done via the creation of a new [AAX_CParameter](#) for the gainParameter: the [AAX_CParameter](#) constructor is parametrized with a set of arguments that define the gain parameter's behavior, such as its default value (1.0f), name ("Gain"), taper behavior (linear), etc.

After creating the parameter objects, a series of packets are registered with the host via the inherited [Packet Dispatcher](#), `mPacketDispatcher`, which is a helper object used by [AAX_CEffectParameters](#) to assist with the plug-in's packet management tasks.

```
mPacketDispatcher.RegisterPacket(
    DemoGain_GainID,
    eAlgPortID_CoefsGainIn,
    this,
    &DemoGain_Parameters::UpdatePacket_Gain);
```

Listing 1: Registration of DemoGain's "gain" packet handler

The last parameter in [AAX_CPacketDispatcher::RegisterPacket\(\)](#) takes a reference to a packet handler callback. This method will be called by the Packet Dispatcher whenever new parameter values need to be converted into coefficients that can be used by the algorithm. In DemoGain, a reference to `DemoGain_Parameters::UpdatePacket_Gain()` is used for the gain parameter's coefficient packet.

As a developer, you will choose how this portion of your data model gets handled; you can choose to use the default handler method, which simply forwards the raw parameter values to the algorithm, or you can define a custom handler. You can also choose to bypass the Packet Dispatcher completely (see the `DemoDist_GenCoef` plug-in for an example of this.)

Although the handling of DemoGain's gain parameter is trivial, for the sake of demonstration this plug-in uses both Packet Dispatcher approaches in the registration of its bypass and gain parameters.

12.2.4.4 Algorithm: Add coefficients to the algorithm's context structure

Your plug-in's context is nothing more than a data structure of pointers that is registered with the host during `Describe`. These pointers function as *ports* where host-managed data may be delivered or retrieved.

12.2.4.4.1 Context definition

Looking under the "Component context definitions" section within `DemoGain_Alg.h`, you will find DemoGain's `SDemoGain_Algo_Context` context. This is DemoGain's context structure, and its sole purpose is to parametrize DemoGain's algorithm with a set of ports. Note the definition of a port for the gain coefficients that are created by `DemoGain_Parameters::UpdatePacket_Gain()` and another port for the bypass value that is forwarded via the default packet update handler.

```
int32_t          * mCtrlBypassP;
SDemoGain_CoefsGain * mCoefsGainP;
```

Listing 2: Gain and bypass ports in the DemoGain algorithm's context structure

Once ports have been defined for the algorithm's coefficients and other algorithmic data, unique identifiers for each port in the context must be generated. This is accomplished through use of the [AAX_FIELD_INDEX](#) macro. The basic idea behind this macro is to generate IDs that the host can use to directly address the ports within their context:

```
, eAlgPortID_CoefsGainIn = AAX_FIELD_INDEX ( SDemoGain_Algo_Context , mCoef ),
, eAlgFieldID_AudioIn = AAX_FIELD_INDEX ( SDemoGain_Algo_Context , mInputPP ) )
```

Listing 3: Some port ID definitions for the DemoGain algorithm's context structure

Now the context's definition is complete. So far these are just fields in a struct; the host doesn't yet know how to route packets from the data model to these ports. That information will come later, in the plug-in's [description](#). Once this context is described to the host, the host will know to populate it and pass it to the processing function located in `DemoGain_AlgProc.cpp`.

12.2.4.4.2 Algorithm processing callback

```
void
AAX_CALLBACK
DemoGain_AlgorithmProcessFunction (
    SDemoGain_Alg_Context * const inInstancesBegin [],
    const void * inInstancesEnd )
```

Listing 4: The DemoGain algorithm's callback prototype

This is where the plug-in's audio buffer processing of the plug-in occurs. Note that this function is passed a pointer to an array of `SDemoGain_Alg_Contexts`. Each of these represents the state of a particular instance of DemoGain, and contains pointers to the applicable coefficient and audio buffer data.

Using the `SDemoGain_Alg_Context` array, instance-specific information and audio buffers are easily retrieved for processing. DemoGain accomplishes this by first iterating over each plug-in instance, then over each channel, and finally over each sample, at which point the gain coefficient is applied to each sample in the input audio buffer and the sample data is copied to the output audio buffer:

```
// ----- Iterate over plug-in instances -----
for (SDemoGain_Alg_Context * const * walk = inInstancesBegin ; walk <
     inInstancesEnd ; ++ walk )
{
    .

    // ----- Run processing loop over each input channel -----
    //
    for (int ch = 0; ch < kNumChannelsIn ; ch++)
    {
        // ----- Run processing loop over each sample -----
        //
        for (int t = 0; t < kAudioWindowSize ; t++)
        {

            .

            pdO [t] = gain * pdI [t];
            .

        } // Go to the next sample
    } // Go to next channel
} // End instance-iteration loop
```

Listing 5: Iterative loops in the DemoGain algorithm

12.2.4.5 Describe: Connect the parameter throughout the plug-in

As mentioned before, **Describe** provides a static description of all communication pathways between a plug-in's algorithm, host, and data model. In addition, various effect properties are defined that help the host determine how to handle the plug-in.

Communication paths between the various plug-in components are described as connections between source and destination ports. In order for these communication paths to be created, the algorithm must first define some destination ports by actually registering its previously defined context fields as communication destinations. DemoGain does this for its gain parameter through the following call to [AAX_IComponentDescriptor::AddDataInPort\(\)](#) in `DemoGain_Describe.cpp`'s `DescribeAlgorithmComponent()` method:

```
AAX_IComponentDescriptor * compDesc;
compDesc = outDescriptor->NewComponentDescriptor ();
err = compDesc->AddDataInPort (
    eAlgPortID_CoefsGainIn ,
    sizeof (SDemoGain_CoefsGain) );
```

Listing 6: Adding a data port to DemoGain's algorithm component descriptor

This registration process is required for both custom coefficients (Gain) and for data that all plug-ins need such as audio input and output fields.

That completes the connection, and now the plug-in is fully wired to receive parameter updates, convert raw parameter values to algorithmic coefficients, pack these coefficients into a packet, post the packet to the host for routing, receive the updated packet in the list of context structures that the host provides when calling the algorithm callback, and apply the updated coefficient data in the appropriate context structure to the plug-in's audio data. Whew!

12.2.4.6 GUI: Add a control

Although the specific steps for adding a GUI control to edit a plug-in parameter will vary depending on the GUI framework you choose, there are a few basic design principles that should always be followed.

The basic DemoGain plug-in does not include any GUI implementation. For practical GUI implementation examples, open the DemoGain_GUIExtensions sample plug-ins. DemoGain_Cocoa provides an example of a custom plug-in GUI using the native OS X SDK, while DemoGain_Win32 uses native Windows APIs. The other examples in this folder require common third-party libraries.

12.2.4.6.1 Control edits vs. parameter updates

The most important principle for AAX GUI design is that an edit in a plug-in's GUI should never directly set the state of the associated parameter or parameters. This is because there may be other controller clients of the plug-in's data model which will need to be notified of the edit, or which may need to override edits from the GUI.

- On control edit

When a control is edited in the plug-in GUI, call `AAX_IEffectParameters::SetParameterNormalizedValue()`. The implementation of this method should post a request to the host in order to trigger the parameter update. The GUI should not update its state until the corresponding parameter update notification is received.

- On parameter update

A parameter update can occur in response to a GUI edit, an edit from another attached controller, an update to a linked parameter, or any other event that affects the state of the data model. When the state of a parameter changes, `AAX_IEffectGUI::ParameterUpdated()` is called. The plug-in's GUI should be updated in this method in order to reflect the new state of the affected parameter.

For detailed information about the sequence of events and GUI responsibilities during a parameter update, see [Parameter updates](#)

12.2.4.6.2 Notifying the host of GUI events

Some GUI events must be handled by the host rather than by the plug-in. For example, in Pro Tools a user should be able to display a pop-up menu for controlling automation information by command-option-control-clicking (Mac) or alt-ctl-right clicking (Windows) a control. These events, and other direct communication between the GUI and the "container" in which the host creates the plug-in view, is accomplished via the `AAX_IViewContainer` interface. Be sure to always call the handler methods in this interface before handling a mouse event within the plug-in GUI, in order to maintain the expected host behavior.

Collaboration diagram for Getting Started with AAX:



12.3 Core AAX Interface

12.3.1

Main classes, callbacks, and format specification details for a standard AAX plug-in.

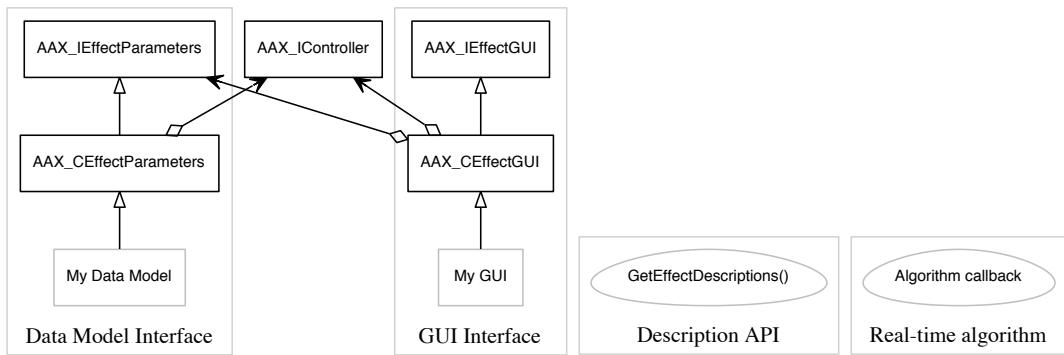


Figure 12.1: Main classes and callbacks for a standard AAX plug-in

These interfaces and components represent the standard interface between an AAX plug-in and the host. Default implementations for each interface are provided in the SDK. See the following modules for more information about each component.

See also

[Component Structure](#) in the [Getting Started Guide](#)

Documents

- [Description callback](#)

Static configuration for an AAX plug-in.

- [Real-time algorithm callback](#)

A plug-in's audio processing core.

- [Data model interface](#)

The interface for an AAX Plug-in's data model.

- [GUI interface](#)

The interface for a AAX Plug-in's user interface.

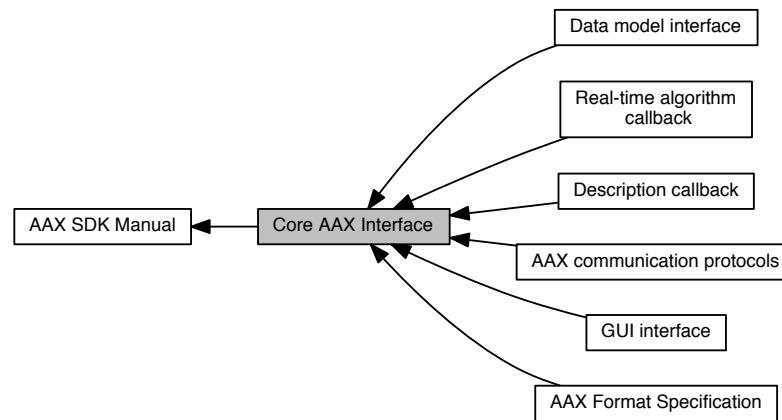
- [AAX communication protocols](#)

How to transfer data between different parts of an AAX plug-in.

- [AAX Format Specification](#)

Additional requirements for AAX plug-ins.

Collaboration diagram for Core AAX Interface:



12.4 Description callback

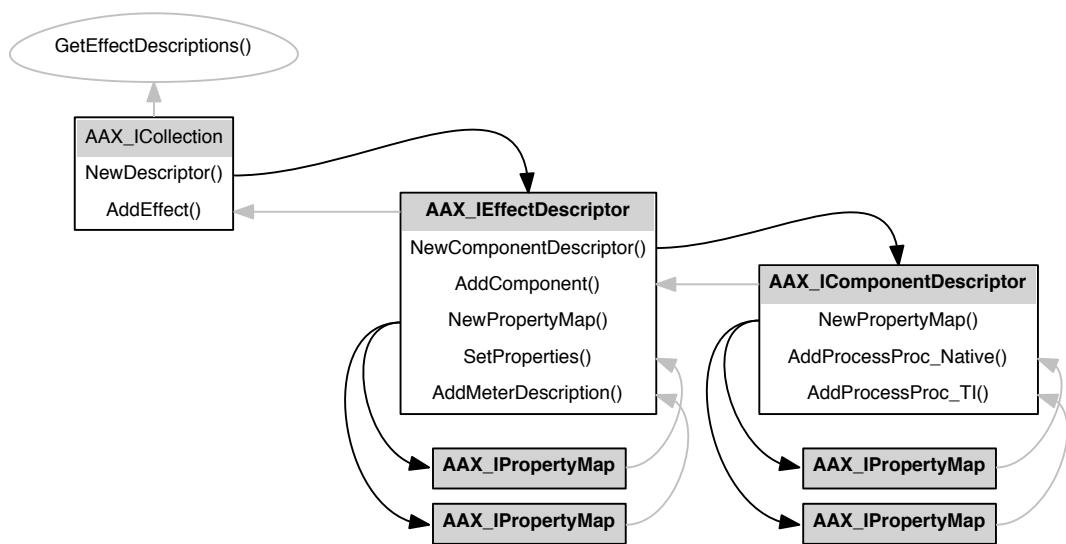
12.4.1

Static configuration for an AAX plug-in.

12.4.2 On this page

- [About the Describe callback and AAX descriptor interfaces](#)
- [Top level: Collection](#)
- [Middle level: Effects](#)
- [Lowest level: Algorithm components](#)
- [Checking Results](#)
- [Describe Validation](#)
- [Additional Topics](#)

12.4.3 About the Describe callback and AAX descriptor interfaces



In Describe, a plug-in declares its static (or default) configuration and any properties that the host will need in order to manage the plug-in.

A plug-in's Describe callback ties the Algorithm, GUI, and Data Model together. In this callback, the plug-in provides a description of its algorithm callbacks, data connections, and other static plug-in properties using a set of host-provided description interfaces. The plug-in uses a tiered hierarchy of these description interfaces to complete its description:

- At the top level, a single Collection interface represents properties that apply to the plug-in binary.
- The Collection interface is populated with one or more Effect descriptors, each of which represents a single kind of effect. For example, a single dynamics plug-in binary may include both single-band and multi-band Effects. Each Effect represents a different plug-in "product" available to users.

- Each Effect is registered with a set of one or more algorithm components that represent the specific processing configurations (e.g. stem formats, sample rates) that the plug-in supports. The set of components in a single Effect provide possible variations of the single plug-in "product", and as such these variations are mostly transparent to users.

The actual description callback entrypoint is [ACFRegisterPlugin\(\)](#), which is declared in [AAX_Exports.cpp](#). This method is implemented inside of the AAX Library, where it calls the plug-in's custom implementation of the [GetEffectDescriptions\(\)](#) callback.

```
// ****
// ROUTINE: GetEffectDescriptions
// ****
AAX_Result GetEffectDescriptions( AAX_ICollection *
    outCollection )
{
    AAX_Result          result = AAX_SUCCESS;
    AAX_IEffectDescriptor *  plugInDescriptor = outCollection->
        NewDescriptor();
    if ( plugInDescriptor )
    {
        result = DemoGain_GetPlugInDescription( plugInDescriptor );
        if ( result == AAX_SUCCESS )
            outCollection->AddEffect( kEffectID_DemoGain, plugInDescriptor );
    }
    else result = AAX_ERROR_NULL_OBJECT;

    outCollection->SetManufacturerName( "Avid, Inc." );
    outCollection->AddPackageName( "DemoGain Plug-In" );
    outCollection->AddPackageName( "DemoGain" );
    outCollection->AddPackageName( "DmGi" );
    outCollection->SetPackageVersion( 1 );

    return result;
}
```

[GetEffectDescriptions\(\)](#) from the DemoGain example plug-in

In general, the following procedure is used when describing an AAX plug-in:

1. Create an Effect description interface from the Collection interface provided by the host
2. Use the Effect description interface to create references to one or more component description interfaces.
3. Describe each algorithm component by populating the component descriptions.
4. Add the components to the Effect description
5. Add additional modules and properties to the Effect description.
6. Add the completed Effect to the Collection
7. Repeat for any additional Effects included in the plug-in binary.
8. Return the completed Collection interface to the host and exit.

Note

The host owns all memory associated with any descriptors that the plug-in returns via this callback.

12.4.4 Top level: Collection

The [AAX_ICollection](#) interface provides a creation function ([AAX_ICollection::NewDescriptor\(\)](#)) for new plug-in descriptors, which in turn provides access to the various interfaces necessary for describing a plug-in (see the [DemoGain_GetPlugInDescription\(\)](#) and [DescribeAlgorithmComponent\(\)](#) listings below).

When a plug-in description is complete, it is added to the collection via the [AAX_ICollection::AddEffect\(\)](#) method. The [AAX_ICollection](#) interface also provides some additional description methods that are used to describe the overall plug-in package. These methods can be used to describe the plug-in package's name, the name of the plug-in's manufacturer, and the plug-in package version. Once these have been described, the completed description interface is returned to the host and exits.

```

// ****
// ROUTINE: GetEffectDescriptions
// ****
AAX_Result GetEffectDescriptions( AAX_ICollection *
    outCollection )
{
    .

    .

    AAX_IEffectDescriptor *    plugInDescriptor = outCollection->
        NewDescriptor();

    .

    result = DemoGain_GetPlugInDescription( plugInDescriptor );

    .

    outCollection->AddEffect( kEffectID_DemoGain, plugInDescriptor );

    .

    outCollection->SetManufacturerName( "Avid, Inc." );
    outCollection->AddPackageName( "DemoGain Plug-In" );
    outCollection->AddPackageName( "DemoGain" );
    outCollection->AddPackageName( "DmGi" );
    outCollection->SetPackageVersion( 1 );

    .

    return result;
}

```

Populating the [AAX_ICollection](#) interface

12.4.5 Middle level: Effects

The [AAX_IEffectDescriptor](#) interface provides description methods that are used to describe the Effect, such as its name, category, associated page table, and, importantly, creation methods for its data model, GUI, and other AAX modules.

```

// ****
// ROUTINE: DemoGain_GetPlugInDescription
// ****
static AAX_Result DemoGain_GetPlugInDescription( AAX_IEffectDescriptor *
    outDescriptor )
{
    int             err;
    AAX_IComponentDescriptor * compDesc = outDescriptor->
        NewComponentDescriptor();
    if ( !compDesc )
        return AAX_ERROR_NULL_OBJECT;

    // Add empty component descriptors to the host, register a processing
    // entrypoint for each, and populate with description information.
    //
    // Alg component
    DescribeAlgorithmComponent( compDesc );
    err = outDescriptor->AddComponent( compDesc ); AAX_ASSERT( err == 0 );

    outDescriptor->AddPlugInName ( "Demo Gain AAX" );
    outDescriptor->AddPlugInName ( "Demo Gain" );
    outDescriptor->AddPlugInName ( "DemoGain" );
    outDescriptor->AddPlugInName ( "DmGain" );
    outDescriptor->AddPlugInName ( "DGpr" );
    outDescriptor->AddPlugInName ( "Dn" );
    outDescriptor->AddPlugInCategory ( AAX_ePlugInCategory_Dynamics );
    outDescriptor->AddProcPtr( (void *) DemoGain_Parameters::Create,
        kAAX_ProcPtrID_Create_EffectParameters );
    outDescriptor->AddResourceInfo ( AAX_eResourceType_PageTable,
        "DemoGainPages.xml" );

#if PLUGGUI != 0
    outDescriptor->AddProcPtr( (void *) DemoGain_GUI::Create,
        kAAX_ProcPtrID_Create_EffectGUI );
#endif

    return AAX_SUCCESS;
}

```

Populating an [AAX_IEffectDescriptor](#) interface

All components in an Effect must share the same AAX modules; for example, it is not possible to use one data model definition for one sample rate and another data model definition for a different sample rate. Therefore, a plug-in's AAX modules are defined in its Effect description.

12.4.5.1 Registering multiple Effects

A single plug-in package may include multiple Effects, which are added in turn in the description method. Once these connections are made, Describe passes the host a populated description interface and returns.

For example, consider an EQ plug-in that contains both one-band and four-band variations, each of which the user should see as a distinct plug-in. These Effects would be described and added separately to the collection object and would appear as separate products to the user.

```
AAX_Result GetEffectsDescriptions ( AAX_ICollection * outCollection )
{
    AAX_Result result = AAX_SUCCESS ;
    if ( result == AAX_SUCCESS )
    {
        AAX_IEffectDescriptor * aDesc1 = outCollection -> NewDescriptor () ;
        // ...
        // Populate aDesc1 with one - band EQ description
        // ...
        result = outCollection -> AddEffect ( kEffectID_MyOneBandEQ , aDesc1 ) ;
    }
    if ( result == AAX_SUCCESS )
    {
        AAX_IEffectDescriptor * aDesc4 = outCollection -> NewDescriptor () ;
        // ...
        // Populate aDesc4 with four - band EQ description
        // ...
        result = outCollection -> AddEffect ( kEffectID_MyFourBandEQ , aDesc4 ) ;
    }
    if ( result == AAX_SUCCESS )
    {
        outCollection -> SetManufacturerName ( "My Plug -Ins , Inc." );
        outCollection -> AddPackageName ( "MyEQ_Plug -In" );
        outCollection -> AddPackageName ( "MyQ" ); // Short name
        outCollection -> SetPackageVersion ( 1 );
    }
    return result ;
}
```

Registering multiple Effects in a single Collection

12.4.6 Lowest level: Algorithm components

In order to register an algorithm component, a plug-in must describe the component's external interface. This includes each of the component's ports and any other fields in its context structure, a reference to its processing function entrypoint (its "Process Procedure", or ProcessProc) and any other special properties that the host should know about.

The description of a context structure involves a set of port definitions, which can be "hard-wired" to receive data from the host (such as audio buffers), from the plug-in's data model (such as packets of coefficients), or even from past calls to the algorithm (private, persistent algorithm state). See [Real-time algorithm callback](#) for more information on an algorithm's context structure.

```
static void DescribeAlgorithmComponent( AAX_IComponentDescriptor * outDesc )
{
    .
    .
    .
    AAX_Result err;

    // Subscribe context fields to host-provided services or information
    err = outDesc->AddField ( eAlgFieldID_AudioIn, kAAX_FieldTypeAudioIn );
    AAX_ASSERT (err == 0);
    err = outDesc->AddField ( eAlgFieldID_AudioOut, kAAX_FieldTypeAudioOut );
    AAX_ASSERT (err == 0);
    err = outDesc->AddField ( eAlgFieldID_BufferSize, kAAX_FieldTypeAudioBufferLength );
    AAX_ASSERT (err == 0);
```

```

// Register context fields as communications destinations
err = outDesc->AddDataInPort ( eAlgPortID_BypassIn, sizeof (int32_t) );
AAX_ASSERT (err == 0);
err = outDesc->AddDataInPort ( eAlgPortID_CoefsGainIn, sizeof (SDemoGain_CoefsGain) );
AAX_ASSERT (err == 0);

}

.
.
```

Populating a single [AAX_IComponentDescriptor](#) interface

12.4.6.1 Algorithm callback properties

A set of callback properties is required when adding a Process Procedure to an algorithm component. This is done via the [AAX_IPropertyMap](#) interface. Using distinct property maps, a single component may register multiple versions of its callback. For example, an audio processing component might register mono and stereo callbacks, or Native and TI callbacks, assigning each the applicable property mapping. This allows the host to determine the correct callback to use depending on the environment in which the plug-in is instantiated.

```

static void DescribeAlgorithmComponent( AAX_IComponentDescriptor * outDesc )
{
    AAX_IPropertyMap *           properties = outDesc->
        NewPropertyMap();

    .
.

    properties->Clear ();
    properties->AddProperty ( AAX_eProperty_ManufacturerID,
        cDemoGain_ManufacturerID );
    properties->AddProperty ( AAX_eProperty_ProductID,
        cDemoGain_ProductID );
    properties->AddProperty ( AAX_eProperty_InputStemFormat,
        AAX_eStemFormat_Mono );
    properties->AddProperty ( AAX_eProperty_OutputStemFormat,
        AAX_eStemFormat_Mono );
    properties->AddProperty ( AAX_eProperty_CanBypass, true );

    // Native and AudioSuite versions
    properties->AddProperty ( AAX_eProperty_PlugInID_Native,
        cDemoGain_PlugInID_Native );
    properties->AddProperty ( AAX_eProperty_PlugInID_AudioSuite
        , cDemoGain_PlugInID_AudioSuite ); // Since this is a linear plug-in the RTAS version can also be an
        // AudioSuite version.
    properties->AddProperty ( AAX_eProperty_DSP_AudioBufferLength
        , kAAX_NativeAudioBufferLength_Default );
    err = outDesc->AddProcessProc_Native ( DemoGain_AlgorithmProcessFunction <1, 1, 1
        <<kAAX_NativeAudioBufferLength_Default>, properties ); AAX_ASSERT (err == 0);

    // TI DSP Version
    properties->AddProperty ( AAX_eProperty_PlugInID_TI,
        cDemoGain_PlugInID_TI );
    properties->AddProperty ( AAX_eProperty_DSP_AudioBufferLength
        , AAX_eAudioBufferLengthDSP_Default );
    properties->AddProperty ( AAX_eProperty_TI_InstanceCycleCount
        , 1055 );
    properties->AddProperty ( AAX_eProperty_TI_SharedCycleCount
        , 58 );
}

```

Adding properties to a component description

AAX does not require that every value in [AAX_IPropertyMap](#) be assigned by the developer. However, if a specific value is not assigned to one of an element's properties then the element must support any value for that property. For example, if an audio processing component does not provide any stem format properties then the host will assume that the callback will support any stem format.

12.4.7 Checking Results

12.4.7.1 Summary

- Use [AAX_CheckedResult](#) to store result values from all method calls in Describe.

- Use the `AAX_SWALLOW` and `AAX_SWALLOW_MULT` macros to encapsulate independent describe code, such as registration logic for separate Effects or for separate ProcessProc variations within a single Effect.

12.4.7.2 The Problem

With plain `AAX_Result` values it can be challenging to properly detect and handle error states. Each description method call returns an `AAX_Result` to indicate success or failure, and often problems in a plug-in's configuration can be addressed by properly detecting and resolving errors that occur here. However, adding a return value check after every method and providing conditional logic in the case of a failure is onerous, ugly, and difficult to maintain.

```

AAX_Result result = AAX_SUCCESS;

result = descriptor->SomeMethod();
result = descriptor->AnotherMethod(); // oops! might ignore an error

// -----
// Safer, but not a good solution:
// Information about the errors is lost, the
// merged error code is meaningless, and
// debugging to find the location of the
// failure is hard.
//
result |= descriptor->SomeMethod();
result |= descriptor->AnotherMethod();
// ...
if (AAX_SUCCESS != result)
{
    // handle the merged error code
}

// -----
// This is also not a good solution:
// There is no actual handling of errors
// (from the SDK example plug-ins)
//
result = descriptor->SomeMethod();
AAX_ASSERT(AAX_SUCCESS == result);
result = descriptor->AnotherMethod();
AAX_ASSERT(AAX_SUCCESS == result);

// -----
// This is correct but is too hard
//
AAX_Result result = AAX_SUCCESS;
result = descriptor->SomeMethod();
if (AAX_SUCCESS != result) {
    // logic to handle this error:
    // assert and/or log the failure?
    // return or continue execution?
}

result = descriptor->AnotherMethod();
if (AAX_SUCCESS != result) {
    // ditto
}

```

`AAX_Result` based error checking is awkward

12.4.7.3 The Solution

The `AAX_CheckedResult` class is designed to solve this problem. `AAX_CheckedResult` can be used just like a plain-old-data `AAX_Result`:

```

AAX_CheckedResult result = AAX_SUCCESS;
result = descriptor->SomeMethod();
result = descriptor->AnotherMethod();

```

Simpler result checking with `AAX_CheckedResult`

When a failure is encountered, `AAX_CheckedResult` will:

- Store the error value
- Log the error using AAX_TRACE_RELEASE
- Throw an exception of type `AAX_CheckedResult::Exception`

To make this safe to use in the Describe routine, the `AAX` Library includes a try/catch block around the call to the plug-in's `GetEffectDescriptions()` routine.

Warning

Do not use `AAX_CheckedResult` anywhere where an exception could escape to the host (`GetEffectDescriptions()` is OK)

12.4.7.4 Handling Errors and Managing Control Flow

With the basic approach shown above, any error which is encountered will throw an exception which will cancel the plug-in's registration and prevent the plug-in from being shown in the host. However, most errors can be safely handled without canceling the entire plug-in registration.

```
AAX_CheckedResult result = AAX_SUCCESS;

// effect 1 registration
result = DescribeMyEffect1( effect1Descriptor );
result = outCollection->AddEffect( myEffect1ID, effect1Descriptor );

// effect 2 registration
result = DescribeMyEffect2( effect2Descriptor );
result = outCollection->AddEffect( myEffect2ID, effect2Descriptor );
```

Example with no error handling

In this example, a failure when describing either individual effect will prevent the other effect from being registered. Registration of individual ProcessProcs within a single effect, e.g. for different stem formats, is similar.

To allow the registration of other effects to proceed in the event of a failure, any exceptions thrown during the registration of one effect should be caught and should only prevent the registration of that individual effect.

```
AAX_CheckedResult result = AAX_SUCCESS;

// effect 1 registration
try {
    result = DescribeMyEffect1( effect1Descriptor );
    result = outCollection->AddEffect( myEffect1ID, effect1Descriptor );
}
catch (const AAX_CheckedResult::Exception& ex) {
    // log the error using ex.What()
    // swallow the exception and proceed
}

// effect 2 registration
try {
    result = DescribeMyEffect2( effect2Descriptor );
    result = outCollection->AddEffect( myEffect2ID, effect2Descriptor );
}
catch (const AAX_CheckedResult::Exception& ex) {
    // ditto
}
```

Example of error handling with try/catch

This solves the problem fully, but it is still cumbersome - especially when registering a long list of separate ProcessProc variants!

The `AAX_SWALLOW_MULT` macro makes it easier to handle errors which are thrown by `AAX_CheckedResult`:

```
AAX_CheckedResult result = AAX_SUCCESS;

// effect 1 registration
AAX_SWALLOW_MULT (
    result = DescribeMyEffect1( effect1Descriptor );
```

```

result = outCollection->AddEffect( myEffect1ID, effect1Descriptor );
);

// effect 2 registration
AAX\_SWALLOW\_MULT (
    result = DescribeMyEffect2( effect2Descriptor );
    result = outCollection->AddEffect( myEffect2ID, effect2Descriptor );
);

```

Example of error handling with [AAX_SWALLOW_MULT](#)

Variations

- For single-line try/catch there is also [AAX_SWALLOW](#).
- If you need to reference the error value after the exception is caught, use [AAX_CAPTURE_MULT](#) (multi-line) or [AAX_CAPTURE](#) (single-line)
- If you know that a certain error code is OK and should not throw in a given situation then you can add it as an exception to the [AAX_CheckedResult](#) object with [AAX_CheckedResult::AddAcceptedResult\(\)](#).

For examples of [AAX_CheckedResult](#) in use, see the [DemoGain_Multichannel](#) and [DemoGain_UpMixer](#) plug-ins

12.4.8 Describe Validation

12.4.8.1 Validation with DSH

You can validate your plug-in's Describe routine using the [DigiShell](#) command-line tool. The validation command is available directly in the aaxh dish and is also available through an AAX Validator test module:

aaxh dish

```
dsh> load_dish aaxh
dsh> loadpi "/quoted/path/without escape chars/MyPlugIn.aaxplugin"
dsh> getdescriptionvalidationinfo 0
```

AAX Validator

```
dsh> load_dish aaxval
dsh> runtest [test.describe_validation, "/quoted/path/without escape chars/MyPlugIn.aaxplugin"]
```

12.4.8.2 Validation with Pro Tools

Beginning in Pro Tools 12.8.2, developer builds of Pro Tools will also check plug-in describe routines and will present an alert dialog when the plug-in is scanned if any aspect of the plug-in's describe code has failed the validation step.

Describe validation warning in a Pro Tools developer build

The specific kinds of errors which were encountered will be printed to the [DigiTrace](#) log file:

```
13033.502646,00307,0073: ERROR: Unknown target host for the plug-in.
13033.502662,00307,0073: ERROR: PlugInID property is missing for a ProcessProc (process, initialization, or ba
13033.502734,00307,0e0f: CMN_TRACEASSERT Sandbox.aaxplugin configuration contains 2 errors. See the DigiTrace
```

This check may be suppressed using the following [DigiOption](#):

```
TestPlugInDescriptions 0
```

12.4.9 Additional Topics

See also

[Plug-in meters](#)

Classes

- class [AAX_ICollection](#)
Interface to represent a plug-in binary's static description.
- class [AAX_IComponentDescriptor](#)
Description interface for an AAX plug-in component.
- class [AAX_IEffectDescriptor](#)
Description interface for an effect's (plug-in type's) components.
- class [AAX_IPropertyMap](#)
Generic plug-in description property map.

Functions

- **AAX_Result AAXRegisterPlugin (IACFUnknown *pUnkHost, IACFPluginDefinition **ppPluginDefinition)**
The main plug-in registration method.
- **AAX_Result GetEffectDescriptions (AAX_ICollection *inCollection)**
The plug-in's static Description entrypoint.

12.4.10 Function Documentation

12.4.10.1 AAX_Result AAXRegisterPlugin (IACFUnknown * pUnkHost, IACFPluginDefinition ** ppPluginDefinition)

The main plug-in registration method.

This method determines the number of components defined in the dll. The implementation of this method in the AAX library calls the following function, which must be implemented somewhere in your plug-in:

```
1 extern AAX_Result GetEffectDescriptions( AAX_ICollection * outCollection );
```

Wrapped by [ACFRegisterPlugin\(\)](#)

Referenced by [ACFRegisterPlugin\(\)](#).

Here is the caller graph for this function:



12.4.10.2 AAX_Result GetEffectDescriptions (AAX_ICollection * inCollection)

The plug-in's static Description entrypoint.

This function is responsible for describing an AAX plug-in to the host. It does this by populating an [AAX_ICollection](#) interface.

This function must be included in every plug-in that links to the AAX library. It is called when the host first loads the plug-in.

Parameters

out	<i>inCollection</i>	
-----	---------------------	--

Collaboration diagram for Description callback:



12.5 Real-time algorithm callback

A plug-in's audio processing core.

12.5.1 On this page

- [Algorithm definition](#)
- [Algorithm memory management](#)
- [Communicating with the algorithm](#)
- [Algorithm initialization](#)
- [Algorithm processing](#)
- [Persistent algorithm memory](#)
- [Example algorithm callback](#)
- [Port Types and Behavior](#)
- [Additional Information](#)

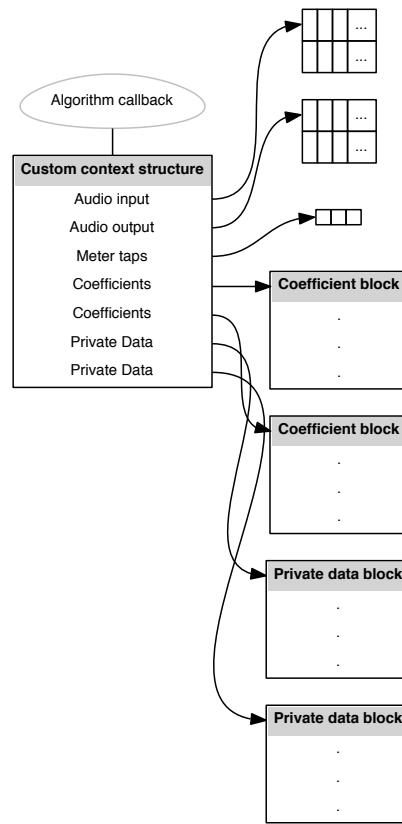
12.5.2 Algorithm definition

Algorithm processing in AAX plug-ins is handled via a C-style algorithm processing callback (see code below.) Each Effect variation in a plug-in must register an algorithm entrypoint in the plug-in [description](#), and the host will call this entrypoint to render a buffer of audio samples.

```
void AAX_CALLBACK MyPlugIn_AlgorithmProcessFunction(
    SMyPlugIn_Alg_Context * const    inInstancesBegin [],
    const void *                  inInstancesEnd)
{
    //
    // Processing code...
    //
}
```

12.5.3 Algorithm memory management

This callback pattern is designed such that plug-in algorithms may be easily loaded in remote memory spaces on a variety of devices and quickly re-compiled for different operating environments without significant changes to the code, and this design goal informs the algorithm's memory management techniques.



When the AAX host calls a plug-in's algorithm callback, it provides a block of memory describing the state of the plug-in. This block of memory, known as the algorithm's *context*, can be thought of as the algorithm's interface to the outside world: when another part of the plug-in interacts with the algorithm, it does so by posting information to the algorithm's context.

```
//=====
// Component context definitions
//=====

// Context structure
struct SDemoDist_Alg_Context
{
    int32_t           * mCtrlBypassP;          // Coefficient message destination
    float             * mCtrlMixP;            // Coefficient message destination
    float             * mCtrlInpGP;           // Coefficient message destination
    float             * mCtrlOutGP;           // Coefficient message destination
    SDemoDist_DistCoefs * mCoefsDistP;        // Coefficient message destination
    SDemoDist_FiltCoefs * mCoefsFiltP;        // Coefficient message destination

    CSimpleBiquad     * mBiquads;             // Private data

    float*             * mInputPP;              // Audio signal input
    float*             * mOutputPP;             // Audio signal output
    float*             * mMeterTapsPP;          // Meter signal output
};
```

It is important to note that, in most circumstances, algorithm callbacks do not own their own memory. The algorithm and its memory is managed entirely by the host or shell environment, and relies on the host-provided context structure for all state information.

If persistent memory is required, algorithms can register for block(s) of persistent state data via the [AAX_IComponentDescriptor::AddPrivateData\(\)](#) API (as in SDemoDist_Alg_Context::mBiquads above.) A plug-in may store state data in the resulting "private data" context fields and this data will be restored by the host when the algorithm is next called. See the [Persistent algorithm memory](#) section below for more information.

12.5.4 Communicating with the algorithm

Plug-ins communicate with their algorithms via a buffered, host-managed message system. The host guarantees that messages posted to this system will be delivered to the applicable context field and that the algorithm's context is up to date every time the component is entered.

This system utilizes a static data routing scheme that is defined in the plug-in's describe method. Once the routing scheme has been defined, the plug-in may post packets of data to its algorithm using [AAX_IController::PostPacket\(\)](#).

In order to reference the fields in its algorithm's context, the plug-in's host-side code uses unique identifiers generated with the [AAX_FIELD_INDEX](#) macro:

```
enum EDemoDist_Alg_PortID
{
    eAlgPortID_BypassIn          = AAX_FIELD_INDEX (SDemoDist_Alg_Context,
    mCtrlBypassP)
    ,eAlgPortID_MixIn           = AAX_FIELD_INDEX (SDemoDist_Alg_Context,
    mCtrlMixP)
    ,eAlgPortID_InpGIn          = AAX_FIELD_INDEX (SDemoDist_Alg_Context,
    mCtrlInpGP)
    ,eAlgPortID_OutGIn          = AAX_FIELD_INDEX (SDemoDist_Alg_Context,
    mCtrlOutGP)
    ,eAlgPortID_CoefsDistIn     = AAX_FIELD_INDEX (SDemoDist_Alg_Context,
    mCoefsDistP)
    ,eAlgPortID_CoefsFilterIn   = AAX_FIELD_INDEX (SDemoDist_Alg_Context,
    mCoefsFiltP)

    ,eAlgFieldID_Biquads        = AAX_FIELD_INDEX (SDemoDist_Alg_Context,
    mBiquads)

    ,eDemoDist_AlgFieldID_AudioIn = AAX_FIELD_INDEX (SDemoDist_Alg_Context,
    mInputPP)
    ,eDemoDist_AlgFieldID_AudioOut = AAX_FIELD_INDEX (SDemoDist_Alg_Context,
    mOutputPP)
    ,eAlgFieldID_MeterTaps      = AAX_FIELD_INDEX (SDemoDist_Alg_Context,
    mMeterTapsPP)
};
```

See [Description callback](#) for more information about registering context fields and defining a plug-in's message routing scheme.

12.5.5 Algorithm initialization

The following events occur before the AAX host begins calling a plug-in's algorithm:

- The Effect's [data model](#) is initialized
- An initial call to [AAX_IEffectParameters::ResetFieldData\(\)](#) is made for each private data block in the algorithm.
- An initial call to [AAX_IEffectParameters::GenerateCoefficients\(\)](#) is made and coefficient packets are dispatched to each of the algorithm's data ports based on the default model state.
- All packets are delivered and initial algorithm context state is set
- If one has been registered, the algorithm's optional initialization callback is called with the default context
- (Algorithmic processing begins)

12.5.5.1 Private data initialization

To initialize an algorithm's private data blocks, [AAX_IEffectParameters::ResetFieldData\(\)](#) is called on the host for each block in the algorithm. The host uses this method to acquire a default initialized memory block for each private data port, which is then copied into the algorithm's memory pool and provided to its context.

The default implementation of this method in [AAX_CEffectParameters](#) will initialize the data to zero.

See also

[Persistent algorithm memory](#)

12.5.5.2 Optional initialization callback

If any additional initialization or de-initialization steps are required for proper operation of the algorithm, an optional initialization routine may be registered and associated with the algorithm's processing callback. This initialization routine will be called in the same device / memory space as the algorithm's processing context. The initialization callback is provided with the algorithm's default context and is called both before every new instance of the Effect begins its algorithm render callbacks and before every instance is destroyed.

This initialization routine is provided in [Describe](#) as an argument to the platform's `AddProcessProc` registration method:

- [AAX_IComponentDescriptor::AddProcessProc_Native\(\)](#)
- [AAX_IComponentDescriptor::AddProcessProc_TI\(\)](#)

Host Compatibility Notes As of Pro Tools 10.2.1 an algorithm's initialization callback routine will have up to 5 seconds to execute.

See also

[AAX_CInstanceInitProc](#)

12.5.6 Algorithm processing

Once the algorithm has been initialized and processing begins, the algorithm function is called regularly by the host audio engine. The algorithm may read the context data provided by the host and is responsible for writing data to all of the samples in its output buffers each time it is executed.

Note

The data in an algorithm's output buffers is not initialized before the algorithm is called, thus the algorithm must always write data into all output samples. This is to ensure equivalent behavior between all platforms, some of which do not have the resource budget to pre-initialize output data buffers.

12.5.7 Persistent algorithm memory

An AAX plug-in algorithm may contain one or more *private data* ports in its context. These are the only context fields in which an algorithm may store persistent state data.

12.5.7.1 Private memory characteristics

Each private data port is a pointer to a preallocated block of memory. The size of each port is defined during [Describe](#) when the port is registered. On DSP systems, the plug-in may request that the data block be placed in the chip's external memory.

Once private data is allocated by the plug-in host or DSP shell, it will not be relocated or re-allocated until the algorithm is destroyed (see [Optional initialization callback](#))

12.5.7.2 Private data port registration

Private data ports are registered during [Describe](#) via [AAX_IComponentDescriptor::AddPrivateData\(\)](#). This method defines the size of the data block that will be allocated as well as an initialization callback with format [AAX_CInitPrivateDataProc](#).

12.5.7.3 Private data initialization

[AAX_IEffectParameters::ResetFieldData\(\)](#) is called on the host for both Native and DSP plug-ins. For DSP plug-ins, the initialized data block is copied to the DSP by the AAX host following the initialization callback. The initialization callbacks for a plug-in's private data blocks are called after all host modules have been initialized and before the algorithm's optional initialization callback.

See also

[Algorithm initialization](#)

12.5.7.4 Private data communication

It is possible to transfer data to and from the algorithm's private data blocks using the [AAX_IPrivateDataAccess](#) interface, which is available in a TimerWakeup context through the [AAX_IEffectDirectData](#) interface. For more information about this API, see auxinterface_directdata_privatedataaccess.

12.5.8 Example algorithm callback

As a final example, the code below describes a simple audio processing component. The component's context contains one message pointer to receive incoming "gain" parameter values, as well as one audio data input, "pdI", and one audio data output, "pdO". Additionally there is a message pointer to receive "bypass" on/off values. The host calls the component each time a new input sample buffer must be processed, and each time the component is called the host ensures that all context fields are up-to-date.

```
void AAX_CALLBACK MyPlugIn_AlgorithmProcessFunction(
    SMyPlugIn_Alg_Context * const inInstancesBegin [],
    const void * const inInstancesEnd)
{
    // Get a pointer to the beginning of the memory block table
    SMyPlugIn_Alg_Context* AAX_RESTRICT instance = inInstancesBegin [0];

    //----- Iterate over plug-in instances -----
    for (SMyPlugIn_Alg_Context * const * walk = inInstancesBegin; walk < inInstancesEnd; ++walk)
    {
        instance = *walk;

        //----- Retrieve instance-specific information -----
        //
        const SMyPlugIn_CoefsGain* const AAX_RESTRICT coefsGainP = instance->mCoefsGainP; // Input (const)
        const int32_t bypass = *instance->mCtrlBypassP;
        const float gain = coefsGainP->mGain;

        //----- Run processing loop over each input channel -----
        //
        for (int ch = 0; ch < kNumChannelsIn; ch++) // Iterate over all input channels
        {

            //----- Run processing loop over each sample -----
            //
            for (int t = 0; t < kAudioWindowSize; t++)
            {
                float* const AAX_RESTRICT pdI = instance->mInputPP [ch];
                float* const AAX_RESTRICT pdO = instance->mOutputPP [ch];

                if ( pdI && pdO )
                {
                    pdO [t] = gain * pdI [t];
                    if (bypass) { pdO [t] = pdI [t]; }
                }
                } // Go to the next sample
            } // Go to next channel
        } // End instance-iteration loop
    }
}
```

12.5.9 Port Types and Behavior

In this section, we will examine the various kinds of ports that can be used by the algorithm component in an AAX plug-in:

1. Standard message input
2. Internal state storage
3. Metering output
4. Environment variable retrieval
5. Other functionality enhancement

12.5.9.1 Standard message input

Most ports will function as pointers to incoming data. This data can have any type. For example, an algorithm's context may include a port of type `float*` to receive incoming float data and another port of type `SMyCustomStructure*` to receive incoming `SMyCustomStructure` data.

Like all registered context fields, input ports are managed by the hosting environment such that they always point to the most recently received data at the time that the algorithm callback is entered. The algorithm may not store or alter data in a standard message input port: this data is available as read-only input. If data is stored in the space allocated for the port's data then the result will be undefined behavior.

To define a standard message input port, a plug-in should call [AAX_IComponentDescriptor::AddDataInPort\(\)](#).

12.5.9.2 Internal state storage

Most plug-ins require local data to be accessible to their algorithms. These may be static data, such as lookup tables, or dynamic data, such as coefficient smoothing history or delay lines. In the DemoDist sample plug-in, `SDemoDist_Alg_Context::mBiquads` is an example of this type of port: it is not modified by any other component and `DemoDist_AlgorithmProcessFunction()` relies on the `mBiquads` data persisting between processing calls.

A component that has registered a private data field is given access to a block of private data. Although the memory in this block will be allocated by the host, its data is fully owned by the component. Because this data is considered private to its parent component, other components cannot overwrite or target this data. Plug-ins that need to transmit data directly between their algorithms' private data ports and their other modules may use the [AAX_IEffectDirectData](#) interface, which provides an API for reading from or writing to this data from outside of the algorithm callback.

The plug-in's data model includes an initialization function that is called by the AAX host at the time of plug-in instantiation and "reset" events. This initialization method is called on the host for both Native and DSP plug-ins. Since this method is part of the plug-in's data model, it has direct access to plug-in state information.

12.5.9.3 Metering output

Plug-in metering ports are populated with an array of float values, or 'taps'. One tap is provided per plug-in meter. The algorithm writes per-buffer peak values to this port and the AAX host applies standardized ballistics to these values. Both raw and processed meter values are available to the plug-in's GUI.

12.5.9.4 Environment variable retrieval

Another use of ports is to receive data from the AAX host describing the execution environment. For example, an algorithm may include a port to receive the number of samples in its processing window or the sample rate. These services are provided automatically by the host once the component registers ports for them.

12.5.9.5 Other functionality enhancement

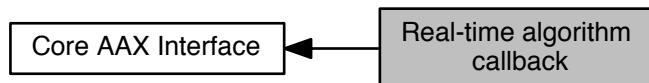
An algorithm component may use ports to gain additional functionality that is provided by the host. For example, an algorithm that will be compiled for accelerated environments may take advantage of the TI chip's Direct Memory Access functionality by registering a DMA port. The host will then allow this port to access memory directly using AAX's DMA APIs.

12.5.10 Additional Information

For information about optional features for the algorithm processing callback, see the following [Additional AAX features](#) documentation:

- [Direct Memory Access](#)
- [Background processing callback](#)

Collaboration diagram for Real-time algorithm callback:



12.6 Data model interface

12.6.1

The interface for an AAX Plug-in's data model.

:Implemented by the Plug-In

The interface for an instance of a plug-in's data model. A plug-in's implementation of this interface is responsible for creating the plug-in's set of parameters and for defining how the plug-in will respond when these parameters are changed via control updates or preset loads. In order for information to be routed from the plug-in's data model to its algorithm, the parameters that are created here must be registered with the host in the plug-in's [Description callback](#).

At [initialization](#), the host provides this interface with a reference to [AAX_IController](#), which provides access from the data model back to the host. This reference provides a means of querying information from the host such as stem format or sample rate, and is also responsible for communication between the data model and the plug-in's (decoupled) algorithm. See [Real-time algorithm callback](#).

You will most likely inherit your implementation of this interface from [AAX_CEffectParameters](#), a default implementation that provides basic data model functionality such as adding custom parameters, setting control values, restoring state, generating coefficients, etc., which you can override and customize as needed.

The following tags appear in the descriptions for methods of this class and its derived classes:

- **CALL:** Components in the plug-in should call this method to get / set data in the data model.

Note

- This class always inherits from the latest version of the interface and thus requires any subclass to implement all the methods in the latest version of the interface. The current version of [AAX_CEffectParameters](#) provides a convenient default implementation for all methods in the latest interface.
- Except where noted otherwise, the parameter values referenced by the methods in this interface are normalized values. See [Parameter Manager](#) for more information.

Legacy Porting Notes In the legacy plug-in SDK, these methods were found in CProcess and CEfectProcess. For additional CProcess methods, see [AAX_IEffectGUI](#).

12.6.2 Related classes

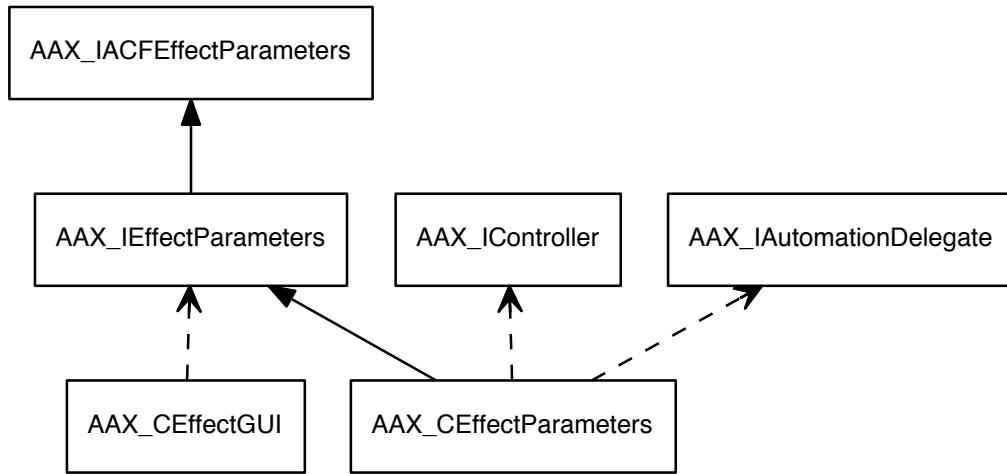


Figure 12.2: Classes related to `AAX_IEffectParameters` by inheritance or composition

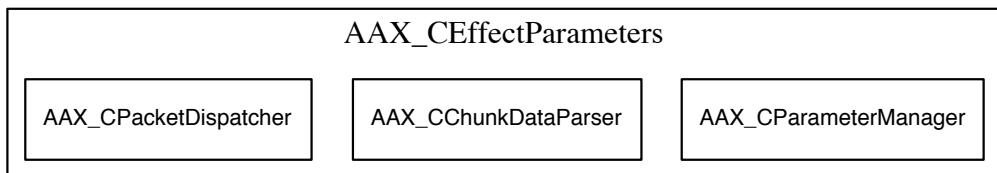


Figure 12.3: Classes owned as member objects of `AAX_CEffectParameters`

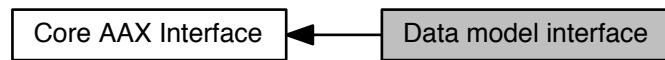
Classes

- class [AAX_CEffectParameters](#)
Default implementation of the `AAX_IEffectParameters` interface.
- class [AAX_IACFEFFECTPARAMETERS](#)
The interface for an AAX Plug-in's data model.
- class [AAX_IACFEFFECTPARAMETERS_V2](#)
Supplemental interface for an AAX Plug-in's data model.
- class [AAX_IACFEFFECTPARAMETERS_V3](#)
Supplemental interface for an AAX Plug-in's data model.
- class [AAX_IACFEFFECTPARAMETERS_V4](#)
Supplemental interface for an AAX Plug-in's data model.

- class [AAX_IEffectParameters](#)

The interface for an AAX Plug-in's data model.

Collaboration diagram for Data model interface:



12.7 GUI interface

12.7.1

The interface for a AAX Plug-in's user interface.

The [GUI interface](#) includes methods for handling the plug-in's GUI window and events.

Accessing the window

In AAX, the plug-in's window is provided as a native window pointer through the [AAX_IViewContainer](#) interface. The plug-in may also use this interface to forward events in its window back to the host for handling.

Default implementation

A default implementation of the GUI interface, [AAX_CEffectGUI](#), is compiled in to the AAX library. This class includes a few helper methods and other extensions to the base interface. Of particular note are several additional pure virtual methods that are used by this class to extend the GUI API, and which must be overridden by any inheriting class.

Extensions

The AAX SDK includes several examples of how the basic GUI interface may be extended to support native or third-party GUI frameworks. These examples are not a core part of the SDK, but are provided to developers as a convenience when incorporating their own chosen GUI framework.

Classes

- class [AAX_CEffectGUI](#)
Default implementation of the [AAX_IEffectGUI](#) interface.
- class [AAX_IACFEffectGUI](#)
The interface for a AAX Plug-in's GUI (graphical user interface).
- class [AAX_IEffectGUI](#)
The interface for a AAX Plug-in's user interface.
- class [AAX_IViewContainer](#)
Interface for the AAX host's view of a single instance of an effect. Used both by clients of the AAX host and by effect components.

Collaboration diagram for GUI interface:



12.8 AAX communication protocols

How to transfer data between different parts of an AAX plug-in.

AAX is a highly modular architecture. This section describes the various means by which AAX plug-in modules may communicate with one another and with the host.

There are two fundamental categories of communication in [AAX](#):

1. [Communication with the C++ interface objects](#)
2. [Communication with the real-time algorithm](#)

12.8.1 Communication with the C++ interface objects

12.8.1.1 Direct host communication

Most communication between the AAX host and the plug-in is accomplished via the [AAX_IController](#) interface. This interface contains methods for such things as:

- Retrieving environment information such as the current [sample rate](#)
- Getting and setting Effect parameters such as the Effect's [algorithmic delay](#)
- Accessing host-managed information such as [Plug-in meters](#) and MIDI
- Accessing other host-managed communications protocols like [Data packets](#) and MIDI

In addition, the GUI uses a separate interface for managing view and event details with the host. This interface, [AAX_IVViewContainer](#), includes methods for:

- Retrieving information like the raw view and the currently held modifier keys
- Requesting changes to view parameters (e.g. size)
- Passing GUI events on to the host.
 - This is an important function because the host may require its own specific behavior for certain events. For example, a command-control-option click in Pro Tools should bring up the parameter's automation menu.

12.8.1.2 Custom data blocks

Often it is necessary to transmit arbitrary blocks of custom plug-in data between different plug-in modules. In AAX, this is accomplished by "pushing" data to and "pulling" it from the plug-in's [data model](#).

The abstract data model interface includes two custom data methods for this:

- [AAX_IEffectParameters::GetCustomData\(\)](#)
- [AAX_IEffectParameters::SetCustomData\(\)](#)

It is the data model's job to act as a go-between when custom data must be transmitted between a plug-in's other modules.

For example, a plug-in may wish to send analysis data from its [direct data module](#) to its [GUI](#). In this situation, the Direct Data object would call [SetCustomData\(\)](#) to update the data model whenever new data was available, while the GUI would "pull" the most up-to-date data via [GetCustomData\(\)](#) whenever an update was required.

Note that the default implementations of these methods are empty and thus all implementation details, including thread safety guards, are left to the plug-in.

12.8.1.3 Notifications

The [data model](#) and [GUI](#) interfaces include [notification hook](#) methods. These methods used for [host-to-Effect notifications](#) by default, but may also be called with custom notification IDs in order to create custom notifications within a plug-in.

12.8.1.4 Direct pointer sharing

If co-location is guaranteed, plug-in modules may directly share data pointers. For example, a non-real-time plug-in's [Host Processor](#) object may share a `this` pointer with its [data model](#) object.

To guarantee co-location between modules that could normally be placed into different memory spaces by the host, use "constraint" properties:

- [AAX_eProperty_Constraint_Location](#)
- [AAX_eProperty_Constraint_Topo](#)

To help avoid forwards-compatibility issues with future devices that support AAX, these constraints should be set whenever a plug-in requires co-location of its components. Note, however, that using a design that relies on co-location will prevent the plug-in from running in distributed environments and should therefore be avoided when possible.

12.8.2 Communication with the real-time algorithm

An AAX plug-in's algorithm is essentially a stateless callback and, therefore, all of its state data must at some level be managed by the host. This model is fundamentally different from the other plug-in modules, which are each objects with their own memory and state.

Most algorithmic data management is performed via the algorithm's context structure. More information about memory management in AAX real-time algorithms can be found [here](#).

12.8.2.1 Data packets

The most common form of communication with a plug-in's real-time algorithm callback is the transmission of read-only data from the data model to the context structure.

AAX includes a dedicated API for this task that provides buffered, optimized delivery of read-only data packets to the algorithm. For more information, see [Communicating with the algorithm](#).

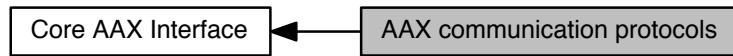
12.8.2.2 Host-managed context fields

Algorithms can also send data to the host and receive environment information through dedicated context fields. For example, the host can provide access to DMA facilities through an object accessed via a DMA field, and a plug-in can report meter values to the host via a dedicated meter field. For more information, see [Communicating with the algorithm](#).

12.8.2.3 Direct data transfers

When other modules in the plug-in must interact directly with the algorithm's state information this is accomplished via the [Direct Data](#) interface. This interface provides an idle-time context in which the plug-in may read from or write to the algorithm's private data memory. These transfers are unbuffered and therefore the plug-in must handle any

appropriate thread-safety considerations. Collaboration diagram for AAX communication protocols:



12.9 AAX Format Specification

Additional requirements for AAX plug-ins.

This document describes aspects of the AAX plug-in format specification that are beyond the scope of the [common interface classes and callbacks](#) that the plug-in must implement.

12.9.1 .aaxplugin Directory Structure

AAX uses a bundle packaging format. On OS X, AAX plug-ins are built as standard OS bundles, while on Windows they are simple directories. All AAX plug-in bundles must use the .aaxplugin extension and the following directory structure:

- /Contents
 - /Resources
 - * *This directory contains all of the additional resource files that will be needed by the plug-in at run time such as DSP algorithm DLLs, XML page tables, and image files for the plug-in's GUI*
 - /MacOS
 - * *Contains the plug-in's OS X binary (Mach-O)**
 - /Win32
 - * *Contains the plug-in's Windows x86 binary**
 - /x64
 - * *Contains the plug-in's Windows x64 binary**
 - /Factory Presets (optional)
 - * *This directory includes built-in plug-in presets. For more information, see [Presets and settings management](#) in the [Pro Tools Guide](#) documentation*
 - PkgInfo (OS X only)
 - * *This file must include the concatenation of the plug-in's CFBundlePackageType (TDMw) and CFBundleSignature (PTul)*
 - Info.plist (OS X only)
 - * *The plug-in's property list*
- desktop.ini (Windows only)
 - *The .aaxplugin directory's view resource file, used to set its custom icon in Windows Explorer*
- PlugIn.ico (Windows only)
 - *Custom plug-in icon file*

*See the following compatibility notes

Host Compatibility Notes

- The plug-in's binary filename must be the same as the outer .aaxplugin bundle name
- On Windows, the plug-in binary (DLL) must use the ".aaxplugin" suffix; i.e. the DLL must use exactly the same name as the outer .aaxplugin folder. On OS X, the plug-in binary does not require a specific suffix.
- On Windows, the plug-in's binary filename (and therefore also the outer .aaxplugin file name) must not contain any spaces. There is a bug in AAE that will prevent binaries with spaces from being loaded properly. This is logged as PTSW-189928.

Note

This directory structure is also used for plug-in installer directories in the VENUE plug-in installer system. See [VENUE Plug-in installer specification](#) for more information.

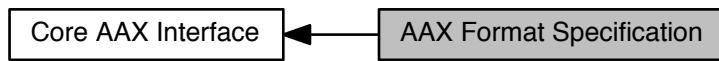
12.9.2 Required Symbols

The following symbols are required in any AAX plug-in and must not be stripped from the binary:

- `_ACFRegisterPlugin`
- `_ACFRegisterComponent`
- `_ACFGetClassFactory`
- `_ACFCanUnloadNow`
- `_ACFStartup`
- `_ACFSshutdown`
- `_ACFGetSDKVersion` *

Host Compatibility Notes *`_ACFGetSDKVersion` is required for 64-bit AAX plug-ins only

Collaboration diagram for AAX Format Specification:



12.10 Additional AAX features

12.10.1

How to use additional features and functionality supported by AAX.

Documents

- [Direct data access interface](#)

A host interface providing direct access to a plug-in's algorithm memory.

- [Offline processing interface](#)

Advanced offline processing features.

- [Hybrid Processing architecture](#)

An architecture combining low-latency and high-latency audio processing.

- [MIDI](#)

How to route and process MIDI in AAX plug-ins.

- [Plug-in meters](#)

How to manage metering data for AAX plug-ins.

- [Sidechain Inputs](#)

Routing custom audio streams to a plug-in.

- [Auxiliary Output Stems](#)

Routing custom audio streams from a plug-in.

- [Direct Memory Access](#)

DMA support for AAX DSP plug-ins, with emulation for AAX Native.

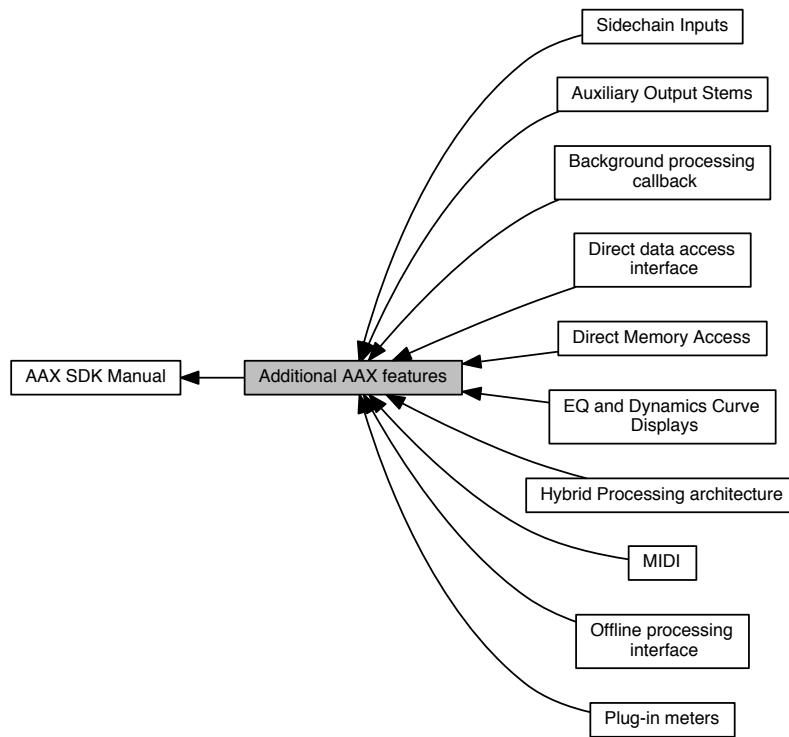
- [Background processing callback](#)

Background processing support for AAX DSP and Native plug-in algorithms.

- [EQ and Dynamics Curve Displays](#)

Displaying EQ and Dynamics curves in Pro Tools, control surfaces, and other auxiliary graphical interfaces.

Collaboration diagram for Additional AAX features:



12.11 Direct data access interface

12.11.1

A host interface providing direct access to a plug-in's algorithm memory.

This interface represents an optional component that you can add to your plug-in in order to support extended features of the AAX SDK.

Some plug-ins require the host to retrieve non-meter data from the decoupled algorithm module to display on a GUI or perform additional computation. For example, the result of computing the audio spectrum or pitch data in the algorithm can be delivered to the host to display on-screen. This is the purpose of the [AAX_IEffectDirectData](#) interface.

The [Direct Data interface](#) provides facilities for directly accessing a plug-in's algorithm memory. This interface may be used to transfer private data from the algorithm to other plug-in components, such as the [GUI](#). It may also be used as an alternative to [PostPacket\(\)](#) to perform direct writes to the algorithm's private data memory.

To set up Direct Data, the module must be registered with the host in the plug-in's Description callback like other process pointers. To add this interface to your plug-in at describe time, call [AAX_IEffectDescriptor::AddProcPtr\(\)](#) using the [kAAX_ProcPtrID_Create_EffectDirectData](#) selector.

The DirectData module works for all plug-in types, including AAX Native, AAX DSP, and AAX AudioSuite.

12.11.2 Convenience class

[AAX_CEffectDirectData](#), the concrete implementation of [AAX_IEffectDirectData](#), consists of a [TimerWakeup←PrivateDataAccess\(\)](#) function that you subclass in order to access an algorithm's private state data. This timer wakes up at a periodic interval. In this function you can read the algorithm's private data port to pull the state of an algorithm. Note that the wakeup period is variable depending on the plug-in's buffer size and running context (real time processing, AudioSuite, offline bounce, etc.) Care must be taken to ensure that any data retrieved from the algorithm is either buffered to handle the thread callback periods for the various running contexts or that the plug-in does not depend on the Direct Data timer catching every state update.

[AAX_CEffectDirectData](#) also includes convenience accessors to the Controller and Data Model in order to help facilitate common access scenarios. Using these, you can do any computation necessary to handle the incoming algorithm state data and send results on to the Data Model and/or the GUI interface.

12.11.3 Private data access interface

The Direct Data API provides a [TimerWakeup](#) callback with access to [AAX_IPrivateDataAccess](#). This reference is only valid within the context of the wakeup callback and cannot be stored to provide private data access in other contexts.

The Private Data Access interface can be used to directly read from and write to an algorithm's private data. These operations are not synchronized with the algorithm's processing callback, which may asynchronously pre-empt the read or write operations. Plug-ins that use this interface should buffer all access to their private data to ensure data integrity.

12.11.4 Communicating with other modules

The Direct Data API does not include any facilities for inter-module communication. In order to transfer data between a plug-in's [AAX_IEffectDirectData](#) object and its other objects, dedicated custom data methods in those objects' interfaces should be used. For example, to communicate with the plug-in's data model, use [AAX_IEffect←Parameters::GetCustomData\(\)](#) and [AAX_IEffectParameters::SetCustomData\(\)](#)

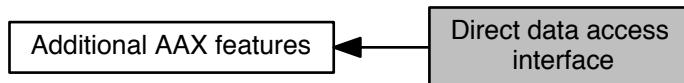
See also

[Hybrid Processing architecture](#) for another approach to transferring large amounts of (audio) data between the algorithm callback and the plug-in's data model.

Classes

- class [AAX_CEffectDirectData](#)
Default implementation of the [AAX_IEffectDirectData](#) interface.
- class [AAX_IACFEffectDirectData](#)
Optional interface for direct access to a plug-in's alg memory.
- class [AAX_IACFPrivateDataAccess](#)
Interface for the AAX host's data access functionality.
- class [AAX_IEffectDirectData](#)
The interface for a AAX Plug-in's direct data interface.
- class [AAX_IPrivateDataAccess](#)
Interface to data access provided by host to plug-in.

Collaboration diagram for Direct data access interface:



12.12 Offline processing interface

12.12.1

Advanced offline processing features.

This interface represents an optional component that you can add to your plug-in in order to support extended features of the AAX SDK.

The HostProcessor interface provides offline plug-ins with useful offline processing features such as random-access facilities and a non-processing analysis callback. For documentation, see the following classes:

- [Host processor module](#)
- [Host processor delegate](#)

To add this interface to your plug-in at describe time, register a [ProcPtr](#) using the [kAAX_ProcPtrID_Create_HostProcessor](#) selector.

Note

If your plug-in does not require the specific offline processing features provided by this interface then it should not register a host processor. Instead, register an offline version of the plug-in's real-time algorithm using the [AAX_eProperty_PlugInID_AudioSuite](#) property.

Classes

- class [AAX_CHostProcessor](#)

Concrete implementation of the [AAX_IHostProcessor](#) interface for non-real-time processing.
- class [AAX_IACFHostProcessor](#)

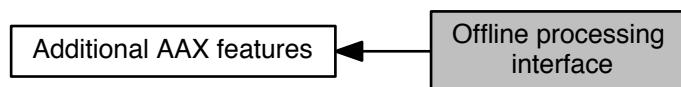
Versioned interface for an AAX host processing component.
- class [AAX_IHostProcessor](#)

Base class for the host processor interface.
- class [AAX_IHostProcessorDelegate](#)

Versioned interface for host methods specific to offline processing.
- class [AAX_VHostProcessorDelegate](#)

Version-managed concrete Host Processor delegate class.

Collaboration diagram for Offline processing interface:



12.13 Hybrid Processing architecture

12.13.1

An architecture combining low-latency and high-latency audio processing.

12.13.2 Overview of Hybrid

Hybrid processing is an optional feature that allows a single plug-in to simultaneously render data on the host's low- and high-latency signal networks. In many large plug-ins this can be very useful. For example, consider a reverb algorithm with both early reflection and tail processing. With AAX Hybrid, this plug-in can process the early reflections at low latency while allowing the tail algorithm to be handled at higher latency (and thus higher efficiency.) Other kinds of algorithms that could benefit from Hybrid processing are noise reductions, analyzers, multi-effect suites, and instruments.

Because the low-latency AAX signal network may be run on DSP hardware, AAX DSP plug-ins that incorporate Hybrid processing can split audio processing between the DSP and the host. This provides the benefits of low latency, highly deterministic DSP-based processing while also allowing the plug-in to leverage the high-latency power of the Intel core where appropriate.

AAX Hybrid is an internal feature and is not exposed to users, except in terms of better plug-in performance and more efficient DSP usage.

Note

AAX Hybrid may be protected by one or more U.S. and non-U.S. patents. Details are available at www.avid.com/patents.

12.13.3 Implementing Hybrid processing

For an example of Hybrid processing, see the [DemoDelay_Hybrid](#) example plug-in

To register for Hybrid processing, a plug-in should add values for [AAX_eProperty_HybridInputStemFormat](#) and [AAX_eProperty_HybridOutputStemFormat](#) to the associated ProcessProc property map. Once these values have been registered, both the ProcessProc callback and the Hybrid render function in the plug-in's data model will be invoked during processing.

Hybrid processing context information is provided via a dedicated [Hybrid processing context structure](#). It is not possible to register additional fields on this context. However, unlike a normal algorithm ProcessProc, the Hybrid render method is implemented directly within the plug-in's effect parameters object and has direct access to the data model memory. This is possible since the render method will always run on the host, and makes it easier to implement algorithms that require access to the data model, e.g. for direct access to impulse responses, etc.

The AAX host provides dedicated audio buffers in both the ProcessProc context and the Hybrid processing context. These buffers can be used to pass audio data between the low-latency ProcesProc and the Hybrid render contexts.

- The plug-in may pass output from the low-latency ProcessProc to the Hybrid render method using additional audio buffers that are added at the end of the ProcessProc context's normal output buffer array. The ProcessProc may perform any pre-processing that is desired before passing audio to the Hybrid render context via these buffers. The [AAX_eProperty_HybridOutputStemFormat](#) property defines how many buffers will be sent from the ProcessProc to the Hybrid render method.
- Similarly, the plug-in may pass samples from the Hybrid processing callback to the low-latency ProcessProc using additional audio buffers that are added at the end of the ProcessProc context's normal input buffer array. The [AAX_eProperty_HybridInputStemFormat](#) property defines how many buffers will be sent from the Hybrid render method to the ProcessProc.

Samples which are sent from the ProcessProc to the Hybrid processing callback and back to the ProcessProc are delayed by a fixed amount relative to the normal input samples that are processed directly by the ProcessProc to

its output buffers. The number of samples of delay that are added in this round-trip is available to the plug-in via [AAX_IController::GetHybridSignalLatency\(\)](#).

12.13.4 Additional information

12.13.4.1 Parameter update timing

Because updates are not passed to the [Hybrid processing context](#) using the normal AAX port infrastructure, any parameter updates from automation will be reflected in this context a little bit ahead of time (~21 ms at 44.1 kHz.) See the [Parameter update timing](#) page for a discussion of parameter timing accuracy and some suggestions of how you can maintain accurate parameter update timing.

12.13.4.2 Host support and alternatives

Not all [AAX](#) hosts support [AAX](#) Hybrid processing. See the [Host Support](#) page for additional information.

See also

[Direct data access interface](#) for another approach for transferring non-audio data between the algorithm callback and the plug-in's data model.

Classes

- struct [AAX_SHybridRenderInfo](#)
Hybrid render processing context.

Functions

- virtual [AAX_Result AAX_IController::GetHybridSignalLatency](#) (int32_t *outSamples) const =0
CALL: Returns the latency between the algorithm normal input samples and the inputs returning from the hybrid component.

Hybrid audio methods

- virtual [AAX_Result AAX_IACFEffectorParameters_V2::RenderAudio_Hybrid](#) ([AAX_SHybridRenderInfo](#) *ioRenderInfo)=0
Hybrid audio render function.

12.13.5 Function Documentation

12.13.5.1 virtual [AAX_Result AAX_IACFEffectorParameters_V2::RenderAudio_Hybrid](#) ([AAX_SHybridRenderInfo](#) **ioRenderInfo*) [pure virtual]

Hybrid audio render function.

This method is called from the host to render audio for the hybrid piece of the algorithm.

Note

To use this method plug-in should register some hybrid inputs and outputs in "Describe"

Implemented in [AAX_CEffectParameters](#).

```
12.13.5.2 virtual AAX_Result AAX_IController::GetHybridSignalLatency( int32_t * outSamples ) const [pure  
virtual]
```

CALL: Returns the latency between the algorithm normal input samples and the inputs returning from the hybrid component.

This method provides the number of samples that the AAX host expects the plug-in to delay a signal. The host will use this value when accounting for latency across the system.

Note

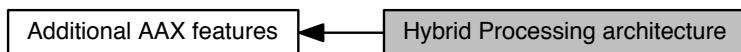
This value will generally scale up with sample rate, although it's not a simple multiple due to some fixed overhead. This value will be fixed for any given sample rate regardless of other buffer size settings in the host app.

Parameters

out	<i>outSamples</i>	The number of samples of hybrid signal delay
-----	-------------------	--

Implemented in [AAX_VController](#).

Collaboration diagram for Hybrid Processing architecture:



12.14 MIDI

How to route and process MIDI in AAX plug-ins.

12.14.1 Midi Overview

DirectMidi is Avid's protocol for communication of MIDI and other timing-critical plug-in information. It is a cross-platform solution to tightly integrate the host application, audio engine, and plug-ins.

12.14.2 MIDI node types

There are four kinds of nodes an AAX plug-in can create. See [AAX_EMIDINodeType](#) for additional details about these node types:

- [AAX_eMIDINodeType_LocalInput](#)
- [AAX_eMIDINodeType_LocalOutput](#)
- [AAX_eMIDINodeType_Global](#)
- [AAX_eMIDINodeType_Transport](#)

12.14.3 Adding MIDI functionality to a plug-in

Plug-in may access MIDI data in its algorithm or data model. If plug-in needs MIDI in both places or just in the algorithm, it should add a MIDI node to the algorithm context, i.e. call [AAX_IComponentDescriptor::AddMIDINode\(\)](#) with the appropriate node type.

```
//=====
// Algorithm context definitions
//=====

// Context structure
struct SMy_Alg_Context
{
    [...]
    AAX_IMIDINode * mMIDIInNodeP;           // Local input MIDI node pointer
    AAX_IMIDINode * mMIDINodeOutP;          // Local output MIDI node pointer
    AAX_IMIDINode * mMIDINodeTransportP;    // Transport node
    [...]
};

enum EDemoMIDI_Alg_PortID
{
    [...]
    //
    // Add the MIDI node as a physical address within the context field
    ,eAlgPortID_MIDINodeIn      = AAX_FIELD_INDEX (SDemoMIDI_Alg_Context, mMIDIInNodeP)
    ,eAlgPortID_MIDINodeOut     = AAX_FIELD_INDEX (SDemoMIDI_Alg_Context, mMIDINodeOutP)
    ,eAlgPortID_MIDINodeTransport = AAX_FIELD_INDEX (SDemoMIDI_Alg_Context,
        mMIDINodeTransportP)
    [...]
};

// *****
// ROUTINE: DescribeAlgorithmComponent
// Algorithm component description
// *****
static void DescribeAlgorithmComponent( AAX_IComponentDescriptor * outDesc )
{
    AAX_Result err;

    [...]
    // Register MIDI nodes
    err = outDesc->AddMIDINode(eAlgPortID_MIDINodeA,
        AAX_eMIDINodeType_LocalInput, "DemoMIDI", 0xffff);
    AAX_ASSERT (err == 0);
    err = outDesc->AddMIDINode(eAlgPortID_MIDINodeOut,
        AAX_eMIDINodeType_LocalOutput, "DemoMIDIOut", 0xffff);
}
```

```

    AAX_ASSERT (err == 0);
    err = outDesc->AddMIDINode(eAlgPortID_MIDINodeTransport,
        AAX_eMIDINodeType_Transport, "DemoMIDITrnsprt", 0xffff);
    AAX_ASSERT (err == 0);
}
[...]
}

```

If MIDI data is needed in the plug-in's data model only, plug-in should describe MIDI node with [AAX_IEffectDescriptor::AddControlMIDINode\(\)](#).

```

// ****
// ROUTINE: GetPlugInDescription
// ****
static AAX_Result GetPlugInDescription( AAX_IEffectDescriptor *
    outDescriptor )
{
    AAX_Result             err;
    [...]
    // Register MIDI nodes
    err = outDesc->AddControlMIDINode('linp', AAX_eMIDINodeType_LocalInput, "
        DemoMIDI", 0xffff);      AAX_ASSERT (err == 0);
    err = outDesc->AddControlMIDINode('lout', AAX_eMIDINodeType_LocalOutput,
        "DemoMIDIOut", 0xffff); AAX_ASSERT (err == 0);
    err = outDesc->AddControlMIDINode('tran', AAX_eMIDINodeType_Transport, "
        DemoMIDITrnsprt", 0xffff); AAX_ASSERT (err == 0);
    [...]

    return err;
}

```

Note

These two types of MIDI nodes can't be used together in the same plug-in's effect.

12.14.4 Using MIDI in a plug-in algorithm

Like with other algorithm context ports, data in MIDI nodes is directly available in the plug-in's algorithm process function. Here is an example from the DemoMIDI_NoteOn sample plug-in:

```

template<int kNumChannelsIn, int kNumChannelsOut>
void
AAX_CALLBACK
DemoMIDI_AlgorithmProcessFunction (
    SDemoMIDI_Alg_Context * const    inInstancesBegin [],
    const void *                      inInstancesEnd)
{
    [...]
    // Setup MIDI In node pointers
    AAX_IMIDINode* midiNodeIn = instance->mMIDINodeP;
    AAX_CMidiStream* midiBufferIn = midiNodeIn->GetNodeBuffer();
    AAX_CMidiPacket* midiBufferInPtr = midiBufferIn->mBuffer;
    uint32_t packets_count_in = midiBufferIn->mBufferSize;

    // Setup MIDI Out node pointers
    AAX_IMIDINode* midiNodeOut = instance->mMIDINodeOutP;
    AAX_CMidiStream* midiBufferOut = midiNodeOut->GetNodeBuffer();
    AAX_CMidiPacket* midiBufferOutPtr = midiBufferOut->mBuffer;
    uint32_t packets_count_out = midiBufferOut->mBufferSize;

    // Setup MIDI Transport node pointers
    AAX_IMIDINode* midiTransport = instance->mMIDINodeTransportP;
    AAX_ITransport * transport = midiTransport->GetTransport();
    bool transport_is_playing = false;
    if (transport)
        transport->IsTransportPlaying(&transport_is_playing);

    if (transport_is_playing)
    {
        //
        // While there are packets in the node
        while (packets_count_in > 0)
        {
            midiBufferOutPtr = midiBufferInPtr;           // Copy the packet from the input MIDI node
                                                        // to the output MIDI node
            midiBufferOutPtr->mTimestamp = timeStamp;    // Set the MIDI time stamp
    }
}

```

```

        midiNodeOut->PostMIDIPacket (midiBufferOutPtr);      // Post the MIDI packet
        midiBufferOut->mBufferSize = packets_count_in;

        midiBufferInPtr++;
        packets_count_in--;
    }
}

[...]
}

```

Also data from the MIDI nodes that were described with [AAX_IComponentDescriptor::AddMIDINode\(\)](#) can be accessed via [AAX_CEffectParameters::UpdateMIDINodes\(\)](#) method. This method provides an [AAX_CMidiPacket](#). Because the MIDI packet structure does not identify the associated MIDI stream's type (input, output, global, or transport) this method also provides an index into the plug-in's algorithm context structure which can be used to identify the semantics of the MIDI packet.

12.14.5 Accessing MIDI in the plug-in data model

A plug-in may access MIDI data in its data model via the [AAX_CEffectParameters::UpdateMIDINodes\(\)](#) or [AAX_CEffectParameters::UpdateControlMIDINodes\(\)](#) methods. Both of these methods provide an [AAX_CMidiPacket](#). Because the MIDI packet structure does not identify the associated MIDI stream's type (input, output, global, or transport) UpdateMIDINodes method also provides an index into the plug-in's algorithm context structure which can be used to identify the semantics of the MIDI packet, while UpdateControlMIDINodes provides MIDI node ID for the same reason.

```

AAX_Result DemoMIDI_Parameters::UpdateMIDINodes ( AAX_CFieldIndex inFieldIndex,
                                                AAX_CMidiPacket& inPacket )
{
    if (eAlgPortID_MIDINodeIn == inFieldIndex)
    {
        if ( (inPacket.mData[0] & 0xF0) == 0x90 )
        {
            if ( inPacket.mData[2] == 0x00 )
            {
                // Note Off
            }
            else
            {
                // Note On
            }
        }
    }

    return AAX_SUCCESS;
}

```

Note

Only one of the UpdateMIDINodes and UpdateControlMIDINodes can be used in the single plug-in's effect at a time. If plug-in uses MIDI nodes described with AddMIDINode function, then only UpdateMIDINodes method can be used to receive MIDI messages. Otherwise UpdateControlMIDINodes should be used.

Collaboration diagram for MIDI:



12.15 Plug-in meters

How to manage metering data for AAX plug-ins.

12.15.1 Overview of metering in AAX

AAX provides a host-managed metering system for plug-ins. The host buffers, thins, and applies ballistics to each of the plug-in's meters. When the plug-in GUI retrieves this processed data, it receives the exact same information that is displayed on control surfaces and other metering devices.

12.15.2 Adding meters to an Effect

Meters are added to an algorithm Component in [Describe](#) using [AAX_IComponentDescriptor::AddMeters\(\)](#). The resulting meter context field will be populated with an array of meter "tap" values, one for each of the Component's meters.

12.15.2.1 Customizing meter behavior

Using the [Effect Descriptor](#), each meter in the Effect may optionally be associated with a [property map](#) that applies a particular set of display properties to the meter. These are the properties that may be set on a meter:

- [AAX_EMeterOrientation](#)
- [AAX_EMeterBallisticType](#)
- [AAX_EMeterType](#)

Note that, because meter properties are added at the Effect level, it is not possible to describe different meter property configurations for different algorithms in the same Effect.

12.15.3 Reporting meter values

Meter values are reported by the algorithm using one "tap" per channel per buffer. For each tap, the algorithm must report the maximum metered sample value for each processing buffer.

Meter tap values can be interpreted as the maximum value of the meter per buffer, on a scale of [0.0 1.0]. In all cases the plug-in's meter position should be normalized between 0 and 1, where 0 is no gain reduction. For example:

- An input meter should report the maximum absolute sample value that is present in the input audio buffer for the appropriate channel
- An output meter should report the maximum absolute sample value that is present in the output audio buffer for the appropriate channel
- A gain-reduction meter (CL or EG types) should report the largest amount of gain reduction in the current buffer for the appropriate channel. If no gain reduction occurred for a buffer then a value of 0.0 should be reported. If a full-scale signal was reduced to silence then a value of 1.0 should be reported.

Gain-reduction meter values should report peak gain reduction, not RMS or other algorithms, and may use any normalization mapping (e.g. linear, exponential) which is desired. Ideally the gain-reduction metering UI in the host and on attached control surfaces will match the Peak gain reduction metering in the plug-in's GUI.

Legacy Porting Notes The gain-reduction meter handling for AAX plug-ins is different from that for RTAS/TD↔M plug-ins. AAX plug-ins must invert their gain-reduction meter values manually before reporting these values from the audio processing callback. The AAX host will always thin reported meter data using a "max" operation, and will later invert gain-reduction meter values before they are available to the plug-in GUI or to control surfaces.

12.15.4 Displaying meter values

The meter values that are reported to the system from the algorithm are available, in buffered and (optionally) ballistics-smoothed form, from [AAX_IController](#). The meter values returned from methods such as [GetCurrentMeterValue\(\)](#) and [GetMeterPeakValue\(\)](#) are the same values used by the system when displaying plug-in meters on control surfaces, and when a plug-in clears the peak value using [ClearMeterPeakValue\(\)](#) this change will likewise be reflected throughout the system.

The literal values provided by these methods can be interpreted as the distance from "rest" that the meter must travel to represent the current value, again on a scale of [0.0 1.0]. Note that this is not necessarily equivalent to the semantics of the meter's reported values in the algorithm:

- For "standard" meters such as input meters, this corresponds to the value provided by the algorithm, since a maximum metered sample value (1.0) corresponds to a meter that should be drawn "furthest from rest" (1.0), i.e. at the top of a standard bottom-to-top meter graphic, or at the far right of a standard left-to-right graphic.
- For "inverted" meters, such as gain-reduction meters, these semantics are reversed: a maximum metered sample value (1.0) corresponds to a meter drawn "at rest" (0.0), i.e. at the bottom of a bottom-to-top meter graphic or at the far left of a left-to-right graphic.

These values are independent of [meter orientation](#): an input or output meter that is oriented with [AAX_eMeterOrientation_TopRight](#) will still use 0.0 as its "at rest" position, and likewise a gain-reduction meter that is oriented with [AAX_eMeterOrientation_BottomLeft](#) will still use 1.0.

12.15.5 Alternatives

For advanced metering applications a single tap value may not be sufficient. To transmit more detailed information from the algorithm to its other components, a plug-in must use the [Direct Data](#) interface. Collaboration diagram for Plug-in meters:



12.16 Sidechain Inputs

Routing custom audio streams to a plug-in.

12.16.1 Overview of Sidechain Inputs

If applicable, plug-ins may choose to enable sidechain inputs. If a sidechain is enabled, a menu is added to the plug-in's header that allows the user to choose an interface or bus as the sidechain, or "key input". Once enabled, the plug-in will be able to access sidechain input just like any other input signal. Currently, DAE is limited to mono sidechain inputs.

12.16.2 Adding a Sidechain Input to an Effect

Setting up a sidechain input is fairly straight forward. You will want to add a physical address within your context structure, and then "describe" the sidechain in `Describe`.

Context Structure:

```
//=====
// Component context definitions
//=====

// Context structure
struct SMyPlugIn_Alg_Context
{
    [...]
    int32_t * mSideChainP;
    [...]
};

// Physical addresses within the context
enum EDemoDist_Alg_PortID
{
    [...]
    ,MyPlugIn_AlgFieldID_SideChain = AAX_FIELD_INDEX (SDemoDist_Alg_Context, mSideChainP)
    [...]
};
```

`Describe`:

```
// ****
// ROUTINE: DescribeAlgorithmComponent
// Algorithm component description
// ****
static void DescribeAlgorithmComponent( AAX_IComponentDescriptor * outDesc )
{
    AAX_Result           err = AAX_SUCCESS;

    [...]
    err = outDesc.AddSideChainIn(eDemoDist_AlgFieldID_SideChain);
    [...]
    properties->AddProperty ( AAX_eProperty_SupportsSideChainInput
        , true );
    [...]
}
```

Todo Is `properties->AddProperty (AAX_eProperty_SupportsSideChainInput, true)` even necessary?!?! I believe I saw a p.i. that does not declare this...

In order to tell whether there is sidechain information available to your plug-in, check for a null pointer within your algorithm's process function. The sidechain channel will show up as an additional stem from the original stem format you declare. That is to say, for a stereo plug-in, the sidechain channel will be the third channel passed in.

```
//=====
// Processing function definition
//=====

void
```

```
AAX_CALLBACK  
MyPlugIn_AlgorithmProcessFunction (  
    SMyPlugIn_Alg_Context * const     inInstancesBegin [],  
    const void *                  inInstancesEnd)  
{  
    [...]  
    int32_t sideChainChannel = *instance->mSideChainP;  
    float * AAX_RESTRICT sideChainInput = 0;  
    if ( sideChainChannel )  
        sideChainInput = instance->mInputPP [sideChain]Channel;  
    [...]  
}
```

Collaboration diagram for Sidechain Inputs:



12.17 Auxiliary Output Stems

Routing custom audio streams from a plug-in.

12.17.1 Overview of Auxiliary Output Stems in AAX

Pro Tools has the capability to show and route multiple "auxiliary" outputs from a plug-in to other tracks. These are known as Auxiliary Output Stems (AOS), a stem referring to one set of outputs. A stereo stem contains two outputs, left and right, and a mono stem contains one output. The outputs will appear in the input assignment pop-up menu of each track under the category "plug-in".

Your plug-in is responsible for the definition of valid aux output paths. This definition includes the total number of outputs and the desired order of stereo and mono paths. Pro Tools will query each plug-in for available valid paths and populate its track input selector popup menus accordingly.

Plug-ins must define the lowest available aux output number. In other words, the port number of an aux output needs to be the lowest available port number after the main outputs of the track the plug-in is instantiated on. For example, the first available aux output for the plug-in residing on a 5.1 surround track would have a port number of 7, since there are 6 main outputs for the track.

Additionally, port numbers must be declared sequentially and in the order aux output stems are added. For example, a stem cannot be added with the port number 10 if it precedes a stem with the port number 4.

12.17.2 Implementing Auxiliary Output Stems

The Auxiliary Output Stems API has a specific descriptor associated with it that needs to be added in `Describe` : `AAX_IComponentDescriptor::AddAuxOutputStem()`. Make sure this method is called for each component that supports a different stem format. For example, a mono aux output would be defined as follows:

```
// ****
// ROUTINE: DescribeAlgorithmComponent
// Algorithm component description
// ****
static void DescribeAlgorithmComponent( AAX_IComponentDescriptor * outDesc )
{
    AAX_Result           err = AAX_SUCCESS;

    [...]
    err = outDesc->AddAuxOutputStem(0 /* first parameter is not used */,
                                    AAX_eStemFormat_Mono,
                                    "My Auxiliary Output Channel");
    AAX_ASSERT( err == AAX_SUCCESS );
    [...]
}
```

The auxiliary output buffers for the plug-in will be appended to the normal output buffer array in the plug-in algorithm.

Warning

Some hosts, such as Media Composer, do not support Auxiliary Output Stems. You must clearly document that your plug-ins are not supported on these hosts; attempts by the plug-in to write data beyond the end of the audio output buffer may cause crashes and other bugs in these hosts. See [Host Support](#) for more information.

In your plug-in's algorithm, you will simply need to account for the extra outputs when it processes the audio. Pro Tools will not automatically route your processed audio to all the extra outputs. As with main outputs, make sure the processed audio samples are placed in the auxiliary outputs' buffers as well. Collaboration diagram for Auxiliary Output Stems:



12.18 Direct Memory Access

DMA support for AAX DSP plug-ins, with emulation for AAX Native.

12.18.1 On this page

- [DMA facility overview](#)
- [DMA transfer modes](#)
- [Registering for DMA transfers](#)
- [DMA restrictions](#)
- [Additional information](#)

12.18.2 DMA facility overview

AAX provides an [abstract interface](#) for accessing the host environment's DMA or other memory-transfer facilities. All platform-specific details are handled by the AAX host environment, allowing plug-ins that use this interface to be re-targeted to Native or DSP environments without changing their memory transfer implementation.

12.18.3 DMA transfer modes

AAX hosts may support the following DMA modes, as listed in [AAX_IDma::EMode](#) :

- In [Scatter](#) mode, data is transferred from a linear buffer to a series of offset segments in a circular buffer. This mode is most often used to transfer data from linear internal memory to a large external memory buffer.
- In [Gather](#) mode, data is collected from a series of offset segments in a circular buffer and concatenated in a linear buffer. This mode is most often used to transfer data from an external memory buffer to an internal memory buffer.
- In [Burst](#) mode, data is written linearly from one location to another. Burst mode transfers may be used for linear transfers of data to or from external memory. During the transfer, the source data is broken into a series of individual bursts. This mode is included for completeness, though the Scatter/Gather modes are expected to be more appropriate for the vast majority of real-world DMA use cases.

12.18.4 Registering for DMA transfers

Algorithm Components register for DMA transfers by adding one or more DMA fields to their context via [AAX_IComponentDescriptor::AddDmaInstance\(\)](#). At runtime, each field will be populated with a valid [DMA interface](#) for the specified DMA mode.

12.18.5 DMA restrictions

The following restrictions apply to DMA transfers on all AAX platforms:

- The maximum burst size for any DMA transfer is 64B. The minimum burst size is 1B.
- Only one DMA transfer request may be posted per [AAX_IDma](#) object per processing callback.
- Scatter and Gather requests each require that the circular memory buffer be padded by at least the size of one burst

12.18.6 Additional information

TI Guide

- DMA support
- DMA and background thread performance reporting

Collaboration diagram for Direct Memory Access:



12.19 Background processing callback

Background processing support for AAX DSP and Native plug-in algorithms.

12.19.1 On this page

- [Background thread description](#)
- [Restrictions and limitations of background threads](#)
- [Background thread performance characteristics on DSP systems](#)
- [Background thread memory management](#)
- [Additional information](#)

12.19.2 Background thread description

Each algorithm render callback may optionally be associated with a background processing callback. This background callback will be triggered regularly in an idle context on a separate thread, and can be used to perform any background task required by the algorithm.

Background thread processing is supported for both [AAX](#) DSP and [AAX](#) Native plug-ins.

12.19.3 Restrictions and limitations of background threads

- An [AAX](#) DSP Effect that registers for background processing will not share a DSP with any other Effect type. It may share a DSP with multiple instances of its own type, but only if its resource requirements allow for this.
- The frequency of background thread executions relative to render thread executions will vary depending on the processing situation. For example, a host may pre-process a series of audio buffers as quickly as possible during an offline render. In this case there would be many executions of the render thread callback for each execution of the background thread callback. Be sure to consider this when using the background thread feature in plug-ins that support an AudioSuite processing type.

12.19.4 Background thread performance characteristics on DSP systems

The background processing callback is called from a true idle thread context. On DSP accelerated platforms, this means that the callback will be triggered continuously whenever the chip is not executing an interrupt, i.e. the algorithm render callback. Since the render callback's resource requirements are well-defined (or at least strictly bounded,) the background thread's available cycles are also deterministically bounded.

However, the background thread itself has a lower priority than the DSP shell. While the background callback's execution will not be interrupted by shell operations, it will be blocked in the event of a contention for memory resources with the shell. As a result, the number of memory operations that may be performed in this callback will be less well-defined when the host is consuming memory resources, e.g. when delivering a very large coefficient block to the DSP.

If your TDM plug-in does not perform any resource-intensive memory operations then you can assume a guaranteed performance level for its DSP background thread. Development tools are available that will test a plug-in by refreshing its entire context memory at every interrupt, and the background thread performance characteristics measured by these tools, plus an additional buffer to account for any pathological cases that may be missed by the performance check, should provide a guaranteed performance baseline for the background thread that will be completely safe for any Pro Tools operation scenario.

12.19.5 Background thread memory management

The background processing callback is not provided with any data pointers and does not have access to any facilities for managed communication with the rest of the plug-in. Therefore, the background process must use shared global data structures to interact with the render callback. Your plug-in will need to manually synchronize access to this data.

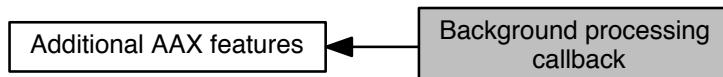
Usually the background callback will want to interact with the render callback via the algorithm's private data blocks. Therefore, private data blocks that are provided to an algorithm's context will not be relocated by the host between calls to the render callback, and background processes can reliably access this data once provided with a pointer. The same is not true for audio buffers, meters, coefficient ports, etc. - this data can all be relocated by the host when the render callback is not executing.

12.19.6 Additional information

TI Guide

- [Background processing](#)
- [DMA and background thread performance reporting](#)

Collaboration diagram for Background processing callback:



12.20 EQ and Dynamics Curve Displays

12.20.1

Displaying EQ and Dynamics curves in Pro Tools, control surfaces, and other auxiliary graphical interfaces.

About

Pro Tools, control surfaces, and other auxiliary displays connected to the AAX host may provide a curve data display to enhance the graphical representation of the plug-in's state.

A "bouncing ball" meter may also be overlaid within the curve data presentation for Dynamics plug-ins.

Pro Tools Mix Window displaying EQ plug-in instances

Pro Tools | S6 MTM display showing a Dynamics plug-in instance with bouncing-ball metering

Requirements

Host Compatibility Notes For S6 control surface displays, see [PT-226228](#) and [PT-226227](#) in the [Known Issues](#) page for more information about the requirements listed in this section.

These are the requirements for supporting the AAX curve data display features:

- To support EQ curve data displays, a plug-in must support [AAX_IEffectParameters::GetCurveData\(\)](#)
- To support Dynamics curve data displays, a plug-in must also support [AAX_IEffectParameters::GetCurveDataDisplayRange\(\)](#) for the Dynamics curve data types.

The AAX host will only query and display curve data for plug-ins of the applicable [Category](#), as specified in the [AAX_ECurveType](#) documentation.

In order to present a bouncing-ball metering display, Dynamics plug-ins must also support [AAX_IEffectParameters::GetCurveDataMeterIds\(\)](#) in addition to the two other curve data methods. This feature is always optional: a Dynamics plug-in may present a curve only without support for a bouncing-ball meter overlay.

Pro Tools Implementation

There are three different kinds of calls that Pro Tools will make when querying EQ plug-ins for curve data:

- An initial query with a small set of points. This query is used only to determine whether the plug-in supports the EQ Curve display feature. The result data is not used. If the plug-in does not support the feature it must return an error value from [GetCurveData\(\)](#)
- A normal full-curve query to get the base curve data across the full display range
- One or more targeted queries around any detected inflection points. These queries are used to increase display resolution for plug-ins with very narrow Q settings.

Pro Tools will call [GetCurveData\(\)](#) from a thread in a low-priority thread pool. Most other Pro Tools operations will not be blocked by the execution of this method, though note that if a control surface is also issuing queries then the method may be called concurrently from multiple host threads.

In Pro Tools, curve updates are triggered by incrementing the plug-in's change counter. This is the counter value returned from the [GetNumberOfChanges\(\)](#) method in the plug-in. This counter is updated automatically when any plug-in parameter changes.

Testing

There are several ways to test a plug-in's S6 curve implementation:

View the plug-in using Pro Tools or another client app or control surface with curve display support

The best way to test your plug-in's curve data display support is to use a real application or control surface to test the behavior of your plug-in with real user workflows. For EQ plug-ins you can use the EQ Curve display in the Pro Tools Mix Window.

View the plug-in using the S6 Surfulator software to emulate an S6 control surface

You can view the plug-in's curve data on the emulated MTM module display. The emulator runs the same software as a true S6 system, so the curve representation in the emulator will be accurate to what would be displayed on S6 hardware.

Test EQ and Dynamics curves at high resolution within Pro Tools using the "TestGetCurveData" DigiOption

See [Using DigiOptions](#) for more information about this option.

TestGetCurveData EQ graph in plug-in header

Use the aaxh dish which is included in the DSH Test Tools package to check your plug-in's curve data

```
dsh> load_dish aaxh
dsh> loadapi "/path/to/your/plug-in.aaxplugin"
dsh> listeffects
View the list of effects in your plug-in and determine which effect you want to test, if there are more than one
dsh> instantiateforcontext {plugin: 0, effect: 0, plat: native, in: mono, out: mono, rate: 48000, alg: true}
Use whatever context parameterization is appropriate for the plug-in configuration you want to test
dsh> getcurvedata {plugin: 0, inst: 0, display: graph}
Experiment with other display values to get other kinds of information
You can edit the instance's parameter values within DSH using the "setparameter" command to see the effects of
```

The "getcurvedata" command will check all three curve data types and will present information for each supported type.

Here is an example of the command output using Avid's Dynamics III plug-in:

```

instanceID: 0
pluginID: 0
message_type: cmd_result
...

```

Enumerations

- enum [AAX_ECurveType](#) { [AAX_eCurveType_None](#) = 0, [AAX_eCurveType_EQ](#) = 'AXeq', [AAX_eCurveType_Dynamics](#) = 'AXdy', [AAX_eCurveType_Reduction](#) = 'AXdr' }

Different Curve Types that can be queried from the Host.

Auxiliary UI methods

- virtual [AAX_Result AAX_IACFEffectParameters::GetCurveData](#) ([AAX_CTypeID](#) iCurveType, const float *iValues, uint32_t iNumValues, float *oValues) const =0
Generate a set of output values based on a set of given input values.

Auxiliary UI methods

- virtual [AAX_Result AAX_IACFEffectParameters_V3::GetCurveDataMeterIds](#) ([AAX_CTypeID](#) iCurveType, uint32_t *oXMeterId, uint32_t *oYMeterId) const =0
Indicates which meters correspond to the X and Y axes of the EQ or Dynamics graph.
- virtual [AAX_Result AAX_IACFEffectParameters_V3::GetCurveDataDisplayRange](#) ([AAX_CTypeID](#) iCurveType, float *oXMin, float *oXMax, float *oYMin, float *oYMax) const =0
Determines the range of the graph shown by the plug-in.

12.20.2 Enumeration Type Documentation

12.20.2.1 enum [AAX_ECurveType](#)

Different Curve Types that can be queried from the Host.

Note

All 'AX__' IDs are reserved for host messages

See also

[AAX_IEffectParameters::GetCurveData\(\)](#)
[AAX_IEffectParameters::GetCurveDataMeterIds\(\)](#)
[AAX_IEffectParameters::GetCurveDataDisplayRange\(\)](#)

Enumerator

[AAX_eCurveType_None](#)

[AAX_eCurveType_EQ](#) EQ Curve, input values are in Hz, output values are in dB.

Host Compatibility Notes Pro Tools requests this curve type for [EQ](#) plug-ins only

[AAX_eCurveType_Dynamics](#) Dynamics Curve showing input vs. output, input and output values are in dB.

Host Compatibility Notes Pro Tools requests this curve type for [Dynamics](#) plug-ins only

[AAX_eCurveType_Reduction](#) Gain-reduction curve showing input vs. gain reduction, input and output values are in dB.

Host Compatibility Notes Pro Tools requests this curve type for [Dynamics](#) plug-ins only

12.20.3 Function Documentation

12.20.3.1 virtual AAX_Result AAX_IACFEffectorParameters::GetCurveData(AAX_CTypeID *iCurveType*, const float * *iValues*, uint32_t *iNumValues*, float * *oValues*) const [pure virtual]

Generate a set of output values based on a set of given input values.

This method is used by the host to generate graphical curves. Given a set of input values, e.g. frequencies in Hz, this method should generate a corresponding set of output values, e.g. dB gain at each frequency. The semantics of these input and output values are dictated by *iCurveType*. See [AAX_ECurveType](#).

Plug-ins may also define custom curve type IDs to use this method internally. For example, the plug-in's GUI could use this method to request curve data in an arbitrary format.

Note

- This method may be called by the host simultaneously from multiple threads with different *iValues*.

Note

- *oValues* must be allocated by caller with the same size as *iValues* (*iNumValues*).

Host Compatibility Notes Versions of S6 software which support the [GetCurveDataDisplayRange\(\)](#) method will not display a plug-in's curve data unless both [GetCurveData\(\)](#) and [GetCurveDataDisplayRange\(\)](#) are supported by the plug-in.

Warning

S6 currently polls this method to update a plug-in's EQ or dynamics curves based on changes to the parameters mapped to the plug-in's EQ or dynamics center section page tables. Parameters that are not included in these page tables will not trigger updates to the curves displayed on S6. ([GWSW-7314](#), [PTSW-195316 / PT-218485](#))

Parameters

in	<i>iCurveType</i>	One of AAX_ECurveType
in	<i>iValues</i>	An array of input values
in	<i>iNumValues</i>	The size of <i>iValues</i>
out	<i>oValues</i>	An array of output values

Returns

This method must return [AAX_ERROR_UNIMPLEMENTED](#) if the plug-in does not support curve data for the requested *iCurveType*

Implemented in [AAX_CEffectParameters](#).

12.20.3.2 virtual AAX_Result AAX_IACFEffectorParameters_V3::GetCurveDataMeterIds(AAX_CTypeID *iCurveType*, uint32_t * *oXMeterId*, uint32_t * *oYMeterId*) const [pure virtual]

Indicates which meters correspond to the X and Y axes of the EQ or Dynamics graph.

These meters can be used by attached control surfaces to present an indicator in the same X/Y coordinate plane as the plug-in's curve data.

Parameters

in	<i>iCurveType</i>	One of AAX_ECurveType
out	<i>oXMeterId</i>	Id of the X-axis meter
out	<i>oYMeterId</i>	Id of the Y-axis meter

Returns

This method should return [AAX_ERROR_UNIMPLEMENTED](#) if the plug-in does not implement it.

Implemented in [AAX_CEffectParameters](#).

```
12.20.3.3 virtual AAX_Result AAX_IACFEffectorParameters_V3::GetCurveDataDisplayRange ( AAX_CTypeID iCurveType,  
float * oXMin, float * oXMax, float * oYMin, float * oYMax ) const [pure virtual]
```

Determines the range of the graph shown by the plug-in.

Min/max arguments define the range of the axes of the graph.

Parameters

in	<i>iCurveType</i>	One of AAX_ECurveType
out	<i>oXMin</i>	Min value of X-axis range
out	<i>oXMax</i>	Max value of X-axis range
out	<i>oYMin</i>	Min value of Y-axis range
out	<i>oYMax</i>	Max value of Y-axis range

Returns

This method should return [AAX_ERROR_UNIMPLEMENTED](#) if the plug-in does not implement it.

Implemented in [AAX_CEffectParameters](#).

Collaboration diagram for EQ and Dynamics Curve Displays:



12.21 AAX Library features

12.21.1

AAX Library core support for the AAX interface

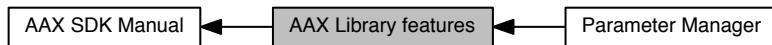
The AAX Library includes several built-in features that are designed to facilitate plug-in development and to make it easy to create plug-ins with correct and consistent behavior. Although these features are not a part of the AAX API, they are a core part of the SDK.

Documents

- [Parameter Manager](#)

Optional (but recommended) system for managing AAX plug-in parameters.

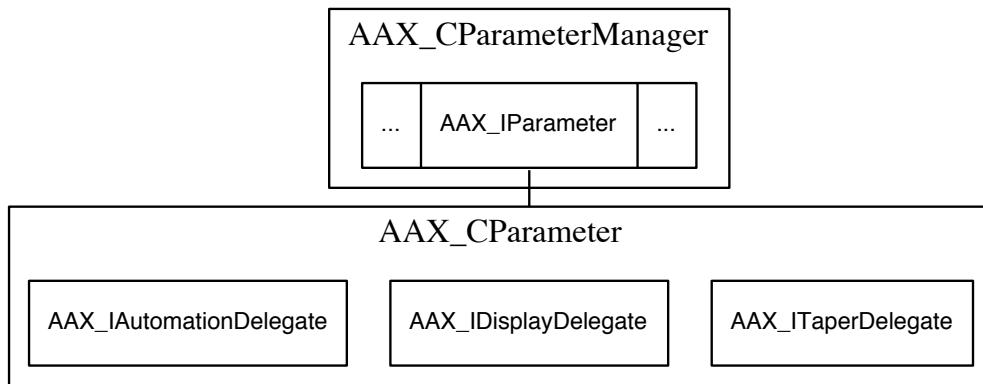
Collaboration diagram for AAX Library features:



12.22 Parameter Manager

12.22.1

Optional (but recommended) system for managing AAX plug-in parameters.



The Parameter Manager is a generic container for a plug-in's parameters, which constitute the complete externally-facing state of a plug-in's data model. Additional internal state data may be stored via settings chunks. The Parameter Manager is owned and operated by the plug-in's [Data model interface](#).

The Parameter Manager provides a convenient and consistent interface by which a plug-in's data model implementation may access its parameters. Other plug-in components that require access to the data model may also use this interface, or a proxy of it, to view the current state of the plug-in.

In the Parameter Manager, implementation-specific parameter behaviors such as taper and display formatting are modular and are applied through delegation. Because of this model, it is possible to easily create a wide variety of behavior combinations without additional subclassing; any display behavior may be combined with any taper behavior, and a newly written behavior can be quickly "mixed in" to many parameters.

12.22.2 Parameter concepts

- [Parameter value domains](#)
- [Taper](#)
- [Delegates](#)
- [Model-View-Controller](#)

12.22.2.1 Parameter value domains

In AAX, parameter values can be represented in one of two "domains". Developers work with parameters in the *real* domain, while the host handles parameters in a scaled, *normalized* format.

Real (or "logical") domain

AAX plug-ins and parameter controllers work with typed parameter values that represent the *real* (logical) state

of the parameter. The type, form, and meaning of this value is dependent on the parameter's implementation and is unknown to the host.

Normalized domain

The AAX host works with parameter values that have been scaled (*normalized*) to a type-agnostic format. Although normalized values make little logical sense, they provide the host with a consistent means of handling, storing, and communicating parameters' values without having to worry about the actual implementation or meaning of the parameters. Normalized parameter values are 64-bit floating point and are scaled to a range of [0, 1].

For more information about conversion between parameter domains, see [AAX_IParameter](#).

Note

The [AAX_IEffectParameters](#) interface currently utilizes a secondary normalization to full-scale int32_t values. In the future, this will be unified with the double precision floating point normalization documented above.

12.22.2.2 Taper

A *taper* is the conversion function that translates a parameter's value between its real and normalized forms.

For example, a taper could be created that converts between a normalized value ([0, 1]) and a real frequency value ranging from [20 2000]. The conversion between these two ranges could be linear or logarithmic, or could use any other desired mapping. This mapping, as well as the specific range of the possible logical values, is defined by the taper.

For more information about tapers in AAX, see [AAX_ITaperDelegate](#).

12.22.2.3 Delegates

In AAX, individual parameters achieve their own unique behavior by being associated with behavioral delegates.

For example, when [AAX_CParameter::SetNormalizedValue\(\)](#) is called on a particular parameter through its [AAX_IParameter](#) interface, the [AAX_CParameter](#) calls into a [AAX_ITaperDelegate](#) that it owns in order to convert the normalized value to its real equivalent. This real value is then set as the parameter's new state.

For more information about how delegates are used to create a parameter's behavior see [AAX_CParameter](#)

12.22.2.4 Model-View-Controller

AAX adheres roughly to a Model-View-Controller pattern. The Parameter Manager functions within the context of [AAX_IEffectParameters](#), which in turn acts as an AAX plug-in's Data Model in an MVC sense. Views, such as the plug-in's GUI, attached control surfaces, or the automation facilities in the AAX host, are given access to the Data Model via a central Controller, which is represented by the [AAX_IController](#) interface.

For more information about how MVC applies to AAX, see the [Data model interface](#) documentation page.

Classes

- class [AAX_CParameter< T >](#)
Generic implementation of an AAX_IParameter.
- class [AAX_CParameterManager](#)
A container object for plug-in parameters.
- class [AAX_IParameter](#)
The base interface for all normalizable plug-in parameters.

Documents

- [Taper delegates](#)
Classes for conversion to and from normalized parameter values.
- [Display delegates](#)
Classes for parameter value string conversion.

Enumerations

- enum [AAX_CParameter< T >::Type](#) {

AAX_CParameter< T >::eParameterTypeUndefined = 0, AAX_CParameter< T >::eParameterTypeBool = 1,

AAX_CParameter< T >::eParameterTypeInt32 = 2, AAX_CParameter< T >::eParameterTypeFloat = 3,

AAX_CParameter< T >::eParameterTypeCustom = 4 }
- enum [AAX_CParameter< T >::Defaults](#) { *AAX_CParameter< T >::eParameterDefaultNumStepsDiscrete = 2, AAX_CParameter< T >::eParameterDefaultNumStepsContinuous = 128 }* }

12.22.3 Enumeration Type Documentation

12.22.3.1 template<typename T> enum AAX_CParameter::Type

Enumerator

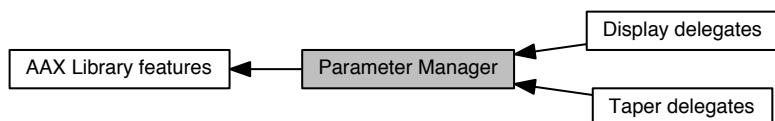
eParameterTypeUndefined
eParameterTypeBool
eParameterTypeInt32
eParameterTypeFloat
eParameterTypeCustom

12.22.3.2 template<typename T> enum AAX_CParameter::Defaults

Enumerator

eParameterDefaultNumStepsDiscrete
eParameterDefaultNumStepsContinuous

Collaboration diagram for Parameter Manager:



12.23 Taper delegates

12.23.1

Classes for conversion to and from normalized parameter values.

Taper delegates are used to convert real parameter values to and from their normalized representations. All taper delegates implement the `AAX_ITaperDelegate<T>` interface template, which contains two conversion functions:

```
virtual T      NormalizedToReal(double normalizedValue) const = 0;
virtual double RealToNormalized(T realValue) const = 0;
```

In addition, tapers may incorporate logical value constraints via the following interface methods:

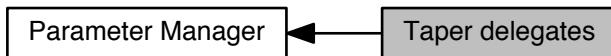
```
virtual T      GetMaximumValue() const = 0;
virtual T      GetMinimumValue() const = 0;
virtual T      ConstrainRealValue(T value) const = 0;
```

For more information, see the [AAX_ITaperDelegate](#) class documentation.

Classes

- class [AAX_ITaperDelegateBase](#)
Defines the taper conversion behavior for a parameter.
- class [AAX_ITaperDelegate< T >](#)
Classes for conversion to and from normalized parameter values.

Collaboration diagram for Taper delegates:



12.24 Display delegates

12.24.1

Classes for parameter value string conversion.

Display delegates are used to convert real parameter values to and from their formatted string representations. All display delegates implement the [AAAX_IDisplayDelegate](#) interface, which contains two conversion functions:

```
virtual bool    ValueToString(T value, std::string& valueString) const = 0;
virtual bool    StringToValue(const std::string& valueString, T& value) const = 0;
```

12.24.2 Display delegate decorators

The AAX SDK utilizes a decorator pattern in order to provide code re-use while accounting for a wide variety of possible parameter display formats. The SDK includes a number of sample display delegate decorator classes.

Each concrete display delegate decorator implements [AAAX_IDisplayDelegateDecorator](#) and adheres to the decorator pattern. The decorator pattern allows multiple display behaviors to be composited or wrapped together at run time. For instance it is possible to implement a dBV (dB Volts) decorator, by wrapping an [AAAX_CDecibelDisplayDelegateDecorator](#) with an [AAAX_CUnitDisplayDelegateDecorator](#).

12.24.2.1 Display delegate decorator implementation

By implementing [AAAX_IDisplayDelegateDecorator](#), each concrete display delegate decorator class implements the full [AAAX_IDisplayDelegate](#) interface. In addition, it retains a pointer to the [AAAX_IDisplayDelegateDecorator](#) that it wraps. When the decorator performs a conversion, it calls into its wrapped class so that the wrapped decorator may apply its own conversion formatting. By repeating this pattern in each decorator, all of the decorator subclasses call into their "wrapper" in turn, resulting in a final string to which all of the decorators' conversions have been applied in sequence.

Here is the relevant implementation from [AAAX_IDisplayDelegateDecorator](#):

```
template <typename T>
AAAX_IDisplayDelegateDecorator<T>::AAAX_IDisplayDelegateDecorator
    (const AAAX_IDisplayDelegate<T>& displayDelegate) :
    AAAX_IDisplayDelegate<T>(),
    mWrappedDisplayDelegate(displayDelegate.Clone())
{
}

template <typename T>
bool        AAAX_IDisplayDelegateDecorator<T>::ValueToString(
    T value, AAAX_CString* valueString) const
{
    return mWrappedDisplayDelegate->ValueToString(value, valueString);
}

template <typename T>
bool        AAAX_IDisplayDelegateDecorator<T>::StringToValue(
    const AAAX_CString& valueString, T* value) const
{
    return mWrappedDisplayDelegate->StringToValue(valueString, value);
}
```

12.24.2.2 Decibel decorator example

Here is a concrete example of how a decibel decorator might be implemented

```
template <typename T>
bool    AAAX_CDecibelDisplayDelegateDecorator<T>::ValueToString
    (T value, AAAX_CString* valueString) const
{
    if (value <= 0)
    {
```

```

    *valueString = AAX_CString("--- dB");
    return true;
}

value = 20*log10(value);
bool succeeded = AAX_IDisplayDelegateDecorator<T>::ValueToString(
    value, valueString);
*valueString += AAX_CString("dB");
return succeeded;
}

```

Notice in this example that the `ValueToString()` method is called in the parent class, `AAX_IDisplayDelegateDecorator`. This results in a call into the wrapped class' implementation of `ValueToString()`, which converts the decorated value to a redecorated string, and so forth for additional decorators.

Classes

- class `AAX_IDisplayDelegateBase`
Defines the display behavior for a parameter.
- class `AAX_IDisplayDelegate< T >`
Classes for parameter value string conversion.

Documents

- [Display delegate decorators](#)
Classes for adapting parameter value strings.

Collaboration diagram for Display delegates:



12.25 Display delegate decorators

12.25.1

Classes for adapting parameter value strings.

The AAX parameter display strategy uses a decorator pattern for parameter value formatting. This approach allows developers to maximize code re-use across display delegates with many different kinds of varying formatting, all without creating interdependencies between the different display delegates themselves.

For more information, see [Display delegate decorators](#). For even more information, about the Decorator design pattern, please consult the GOF design patterns book.

Classes

- class [AAX_CDecibelDisplayDelegateDecorator< T >](#)
A percent decorator conforming to [AAX_IDisplayDelegateDecorator](#).
- class [AAX_CPercentDisplayDelegateDecorator< T >](#)
A percent decorator conforming to [AAX_IDisplayDelegateDecorator](#).
- class [AAX_CUnitDisplayDelegateDecorator< T >](#)
A unit type decorator conforming to [AAX_IDisplayDelegateDecorator](#).
- class [AAX_CUnitPrefixDisplayDelegateDecorator< T >](#)
A unit prefix decorator conforming to [AAX_IDisplayDelegateDecorator](#).
- class [AAX_IDisplayDelegateDecorator< T >](#)
The base class for all concrete display delegate decorators.

Collaboration diagram for Display delegate decorators:



12.26 Additional Topics

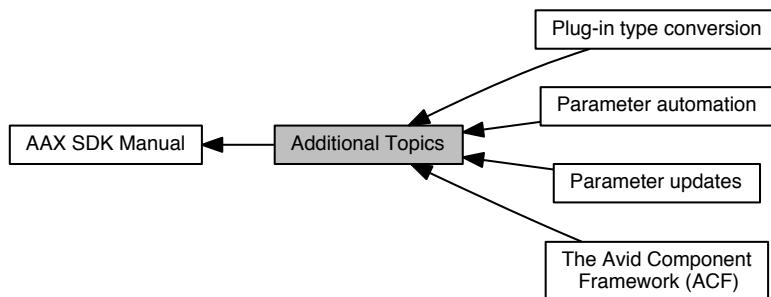
12.26.1

Additional information about the AAX design.

Documents

- [Parameter automation](#)
Information about parameter automation.
- [Parameter updates](#)
The anatomy of a parameter update.
- [Plug-in type conversion](#)
Specification for valid conversions between plug-in types.
- [The Avid Component Framework \(ACF\)](#)
How the AAX C++ interfaces work.

Collaboration diagram for Additional Topics:



12.27 Parameter automation

Information about parameter automation.

12.27.1 On this page

12.27.2 Overview

The term "automation" can mean two things in AAX:

1. A host feature allowing users to record and play back plug-in parameter changes. In this documentation, this data is referred to as **automation data**, and it is stored in **automation lanes** in the host.
2. A system for arbitrating between changes from different parameter editors such as the plug-in GUI, control surfaces, and pre-recorded automation values. In this documentation, this is referred to as the **event system** for parameters.

Here are some examples of how these two different meanings are used in AAX:

- The [AAX_IAutomationDelegate](#) provides methods for interacting with the host's parameter event system.
- [AAX_IACFEEffectParameters::GetParameterIsAutomatable\(\)](#) and the `automatable` parameter in the [AA_X_CParameter](#) constructor reflect whether a parameter can have automation written and read by the host.
- [AAX_IController::GetCurrentAutomationTimestamp\(\)](#) gets the timestamp for pre-recorded automation data when it is received by the plug-in during playback

For more information about the parameter event system, see the [Parameter updates](#) pages, and particularly the information on the [Token protocol](#)

12.27.3 Plug-in elements used for automation

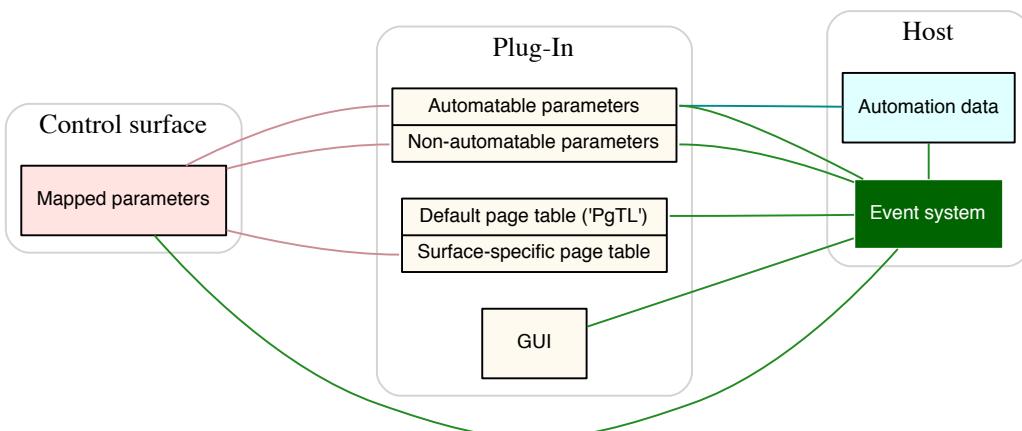


Figure 12.4: Plug-in elements used for events and automation

12.27.3.1 Defining automatable parameters

In order for a parameter to be available for automation recording, editing, and playback, the plug-in must meet the following criteria:

- It must provide `true` when the host calls `GetParameterIsAutomatable()` for the parameter. In nearly all plug-ins, this means providing `true` to the `automatable` parameter in the parameter's `AAX_CParameter` constructor.
- It must expose the parameter to the parameter event system (see below.)

In order for a parameter to be exposed to the event system, the plug-in must meet the following criteria:

- It must respond to all parameter methods in the `AAX_IEffectParameters` interface, particularly `GetNumberOfParameters()` and `GetParameterIDFromIndex()`. Generally this is accomplished by adding an `AAX_CParameter` object for each parameter to the plug-in's `Parameter Manager`.
- It must include the parameter in its one-parameter-per-page 'PgTL' (default) page tables. See [Implementing Page Tables](#) in the [Page Table Guide](#) for more information about defining this page table type.

All plug-in parameters must be registered with the host's event system in order for editors, including the plug-in's GUI, to work properly. Therefore a plug-in should always define a complete 'PgTL' (default) page table including all of its parameters, even the parameters that are not "automatable".

12.27.4 Advanced automation topics

- [Linked parameters](#)

Collaboration diagram for Parameter automation:



12.28 Parameter updates

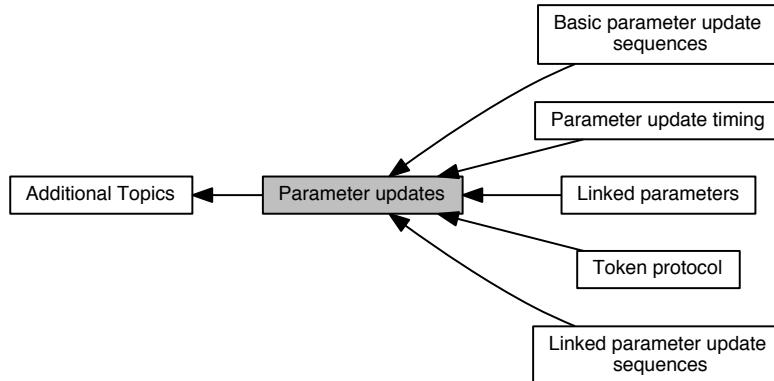
12.28.1

The anatomy of a parameter update.

Documents

- [Parameter update timing](#)
Details about parameter timing and how to keep parameter updates in sync.
- [Token protocol](#)
Communicating parameter state with the host.
- [Basic parameter update sequences](#)
Sequence diagrams for some common parameter update scenarios.
- [Linked parameters](#)
How to link parameters.
- [Linked parameter update sequences](#)
Sequence diagrams for some common linked parameter update scenarios.

Collaboration diagram for Parameter updates:



12.29 Parameter update timing

Details about parameter timing and how to keep parameter updates in sync.

12.29.1 On this page

- [Timeline Locations](#)
- [Coordinating the data model and algorithm](#)
- [Fixing timing issues due to shared data](#)
- [Determining the absolute timestamp for a parameter update](#)

12.29.2 Timeline Locations

At any given moment, a plug-in may be asked to handle events from multiple locations on the timeline. Each module in an AAX plug-in may be updated using a different timeline position. For example:

- During automation playback the host may choose to send parameter updates in advance, while the algorithm is still processing audio from earlier in the timeline.
- When a processing chain involves a significant amount of latency, the host may delay the metering data which is available to the plug-in's GUI until the point in time when the corresponding processed audio is actually being played back to the user.

In this article, we will refer to the following timeline locations:

- *Automation time*: The location that corresponds to the state of the plug-in's data model
- *Playhead*: The location where the audio engine is currently gathering samples for processing
- *Render time*: The location of the audio samples currently being processed by the plug-in's algorithm
- *Presentation time*: The location that corresponds to the playback presentation to the user (i.e. the sound coming out of the speakers)

Figure 1: Timeline locations

12.29.3 Coordinating the data model and algorithm

As an AAX plug-in developer, you don't usually need to worry about the fact that your plug-in's data model and algorithm may each represent a different point in the timeline; the [AAX packet system](#) handles all of the necessary synchronization between these two locations.

This works seamlessly in a normal AAX plug-in because the real-time algorithm is fully decoupled from the plug-in's data model. Since all of the state information for the algorithm is delivered through its [context structure](#), the host can simply swap in the correct context data for each call to the processing callback. The plug-in does not require any special handling code to synchronize between the two timeline locations, and, as a bonus, AAX plug-ins can achieve deterministic, accurate automation playback without doing any extra work to handle time-stamped parameter update queues or other overhead.

Figure 2: Synchronization through the AAX packet system

12.29.3.1 A closer look at the AAX packet delivery system

Adding new packets for automation events

When playing back automation, the AAX host calls [UpdateParameterNormalizedValue\(\)](#) to update the data model state, then calls [GenerateCoefficients\(\)](#) to trigger the generation of new packets. See [Basic parameter update sequences](#) for a full description of this sequence.

Before the host calls [GenerateCoefficients\(\)](#) to generate packets for an automation breakpoint, it records the timeline position of the breakpoint ([AAIX_IController::GetCurrentAutomationTimestamp\(\)](#) provides this value as a sample offset from the beginning of playback.) Every packet that is posted during execution of [GenerateCoefficients\(\)](#) is tagged with this timestamp when it is queued for delivery.

Packet delivery for AAX Native plug-ins

As the playhead advances and sample buffers are queued for processing, the host tracks the location of the next time-stamped packet in the packet queue. As the render time location for a Native plug-in processing chain approaches the next packet time-stamp for a plug-in in the chain, the host divides the plug-in's processing buffers into smaller buffers. When the render time location is as close as possible to the packet's time-stamp, the host delivers the packet. The packet data is available to the algorithm in its context the next time it is executed.

Because the host may divide native processing buffers down to a minimum size of [AAIX_eAudioBufferLengthNative_Min](#) - 32 samples - the host can guarantee that all automation playback will be effected within 32 samples of the actual automation breakpoint location. In addition, with the help of some extra internal bookkeeping, AAX hosts also guarantee that the exact sample where an automation breakpoint is applied will be deterministic and will not change between different playback passes.

Packet delivery for AAX DSP plug-ins

The packet delivery system for AAX DSP plug-ins works similarly to the system for AAX Native plug-ins. AAX DSP plug-ins use a fixed buffer size, so the host is not able to divide their playback buffers into smaller units: the plug-in will receive each data packet in the fixed-size playback buffer which most closely corresponds to the location of the automation event which triggered the packet.

An AAX DSP plug-in which declares an [AAIX_eProperty_DSP_AudioBufferLength](#) value of N will be guaranteed to receive data packets within N/2 samples of the actual automation event position on the timeline. Since the default buffer size for an AAX DSP plug-in is 4 samples, this yields extremely accurate automation playback with no extra work required in the plug-in algorithm.

12.29.4 Fixing timing issues due to shared data

The packet system works perfectly to synchronize the states of the plug-in data model and algorithm, *but only when the plug-in algorithm is fully decoupled from the data model*. If the algorithm directly shares data with the data model then the algorithm will immediately start using any new data model state without waiting for the corresponding coefficient delivery.

Figure 3 shows one kind of problem that can arise when a plug-in uses the same state for both its data model and its algorithm. In this case, the plug-in applied a volume trim (shown in the automation lane at the top of the image) to its algorithm as soon as the parameter update was applied to its data model, even though the algorithm was not yet processing the audio at the Automation time location. As a result, the audio trim was applied several hundred samples too early.

Figure 3: Offset automation playback due to lack of timeline location synchronization in a monolithic plug-in

12.29.4.1 Monolithic plug-ins

Plug-ins that share data directly between their data model and algorithm are referred to as *monolithic*. All plug-ins that inherit from the [AAIX_CMonolithicParameters](#) helper class are monolithic.

Note

Monolithic plug-ins must always set the [AAX_eProperty_Constraint_Location](#) property to include [AAX_eConstraintLocationMask_DataModel](#) in order to avoid being loaded into incompatible AAX hosts.

All monolithic plug-ins must include special handling code to reconcile the plug-in's automation time state with its render time state.

12.29.4.2 How to resolve timing errors

There are many possible solutions for the timing errors that arise when a plug-in combines data from different time locations. Ultimately, the plug-in must separate the state that is represented at different time locations.

In most cases, this requires deferring data model state changes from being applied to the algorithm until the relevant samples are being processed in the render callback. One easy way to accomplish this separation is to take advantage of the synchronization provided by the AAX packet delivery system. This approach benefits from the fact that it emulates the design of a normal, decoupled AAX plug-in.

After a packet is queued with a call to [PostPacket\(\)](#), the packet delivery system will wait to update the algorithm's context structure with the packet's data until the Render time location is very close to the automation event (see [above](#).) This provides an appropriate mechanism for deferring state changes in the plug-in's data model until the Render time location has "caught up" to the correct sample.

Figure 4 shows the same scenario as Figure 3, but now the plug-in has been updated to defer data model updates from the automation time location so that they are applied as coefficients in the algorithm when the render time location has reached the correct point on the timeline.

Figure 4: Deferring a data model update in a monolithic plug-in using the packet queue

Here is one way to use the packet delivery system to defer changes to the data model state:

```
AAX_Result
MyEffectParameters::UpdateParameterNormalizedValue(
    AAX_CParamID iParamID,
    double aValue,
    AAX_EUpdateSource inSource)
{
    // Call inherited
    AAX_Result result =
        AAX_CMonolithicParameters::UpdateParameterNormalizedValue
        (
            iParamID,
            aValue,
            inSource);
    if (AAX_SUCCESS != result) { return result; }

    // Do whatever additional work is required to note that the
    // parameter has been updated - for example, set a "dirty"
    // flag for the parameter.

    return result;
}

AAX_Result
MyEffectParameters::GenerateCoefficients()
{
    // Call inherited
    AAX_Result result = AAX_CMonolithicParameters::GenerateCoefficients
        ();
    if (AAX_SUCCESS != result) { return result; }

    const uint32_t stateNum = mMyStateCounter++; // member uint32_t

    // Do whatever additional work is required to capture the current
    // parameter state and associate it with stateNum, for example
    // check for "dirty" parameters and create a list of these
    // parameters with their values, add this list to a map using
    // stateNum as a key, and clear the "dirty" flags.

    result = Controller()->PostPacket(
        kCurrentStateFieldIndex,
        &stateNum,
        sizeof(uint32_t));

    return result;
}
```

```

struct MyContextStructure
{
    int32_t * mCurrentStateNum; // Private data
    // ...
};

void
MyAudioRenderCallback(
    MyContextStructure* const inInstancesBegin [],
    const void* inInstancesEnd)
{
    /* For each instance... */
    const uint32_t stateNum = instance->mCurrentStateNum;

    // Update the custom plug-in object state based on stateNum
    // and the additional data that was cached during
    // GenerateCoefficients().
}

```

Figure 5: One specific solution for deferring a data model update in a monolithic plug-in using the packet queue

This approach is incorporated directly into the design of [AAX_CMonolithicParameters](#). If your plug-in data model is a subclass of [AAX_CMonolithicParameters](#) then you can follow these steps to ensure accurate parameter update timing in your plug-in:

1. After creating an automatable parameter, call [AAX_CMonolithicParameters::AddSynchronizedParameter\(\)](#) to add the parameter to an internal list of parameters to synchronize using the deferred-update system
2. In the plug-in's [RenderAudio\(\)](#) implementation, iterate through the incoming queue of deferred parameter values
3. Update the coefficients used by the plug-in's algorithm or other processing components

NOTES

- Remember to use the deferred parameter values, not values of the plug-in's [AAX_IParameter](#) objects, when setting the state of the plug-in's coefficients
- The deferred parameter values are delivered in the real-time thread, so all synchronized updates should follow the basic principles of real-time operation such as avoiding memory allocation/free, thread synchronization, access to shared resources, or any other actions which could block the real-time thread

For reference, see [DemoMIDI_Synth](#) and the other example instrument plug-ins. All of the instrument examples in the AAX SDK use these facilities to achieve deterministic, accurate playback for automated parameters.

One benefit of this approach is that it provides a compatible interface with monolithic plug-in objects which are designed to work across multiple plug-in formats. For example, the set of parameter updates provided to [AAX_CMonolithicParameters::RenderAudio\(\)](#) "RenderAudio" can be provided to plug-in objects which require a queue of time-stamped parameter updates for each audio render callback.

12.29.4.3 Additional considerations

Of course, the approach described in this section is just one possible solution. The [timestamp](#) section below provides some alternatives to using the packet queue system for synchronization. Ultimately, the best design for your plug-in will depend on the facilities that are available in the plug-in's monolithic state object, the size of this object, its interface, the number of parameters representing its state, and other internal details.

Here are some additional factors to consider when using the packet queue system for time location synchronization of parameter updates:

- The algorithm callback / [RenderAudio\(\)](#) method is called from a real-time thread, and may be called concurrently with data model methods. You should use a synchronization strategy that is optimized for high performance in this thread.

- If a parameter is not automatable then you should probably ignore these additional steps and directly update the plug-in's monolithic state object from within [UpdateParameterNormalizedValue\(\)](#) when that parameter is changed. Updates for non-automatable parameters can always be applied to the algorithm "as soon as possible".
- Depending on your plug-in's design you may not need or want to apply this solution to some automatable parameters either. For example, parameters that are unlikely to be automated or which require CPU-intensive changes in your instrument object should probably be updated on the object directly from within [Update←ParameterNormalizedValue\(\)](#), and not from within the real-time thread

12.29.5 Determining the absolute timestamp for a parameter update

The AAX packet queue provides a host-managed system for applying parameter updates at the correct location without requiring any special knowledge about the timeline. However, In some situations a plug-in may need to know the absolute sample position of a parameter change.

For example, a plug-in that synchronizes parameter changes to some external system, and which wants to forward these changes over to the external system as early as possible, would want to know the sample position for a coefficient update when the update is first triggered by a call to [GenerateCoefficients](#).

In these situations it is not suitable to simply use a method like [AAX_ITransport::GetCurrentNativeSampleLocation\(\)](#) which returns the current position of the audio render thread. The parameter update may be occurring at a different location on the timeline from the current render position, so using the current render position for the update would result in timeline offset problems similar to those described above.

12.29.5.1 Obtaining timeline information

AAX provides a variety of information that can be used for timeline synchronization. This information is provided through a combination of [AAX_ITransport](#), [AAX_IController](#), and MIDI beat clock data. Here is a summary of the relevant ways that a plug-in can get information about the timeline and timing synchronization data:

- [AAX_ITransport::GetCurrentNativeSampleLocation\(\)](#) Provides the absolute sample position of the first sample in the audio buffer that is currently being processed by the plug-in's worker chain
- [AAX_IController::GetTODLocation\(\)](#) Provides the current "time of day" value, which is a counter within the audio engine that counts the number of samples that the playhead has traversed since playback start
- [AAX_IController::GetCurrentAutomationTimestamp\(\)](#) Must be called from within [GenerateCoefficients](#). Provides the timestamp for the beginning of the hardware audio buffer during which the generated coefficients will be applied to the algorithm. This timestamp is provided in terms of the "time of day" counter, i.e. the number of samples since playback started.
- MIDI Beat Clock Sends transport start/continue/stop events to plug-ins that register global MIDI nodes

12.29.5.2 Determining the timeline position of a parameter update

Each of the available methods for getting information about the timeline position has a particular purpose. No single interface method can be used to directly determine the sample location for a parameter update, but it is possible to determine this value by combining information from a few of the available methods.

Here are some possible approaches for determining the timeline position of a parameter update

Note

Remember that these are not strict recipes; the specific requirements for what kinds of timeline information are needed will vary from plug-in to plug-in. You may be able to refine these approach to better match the needs of your specific plug-in.

12.29.5.2.1 1. Defer the update to the real-time thread

1. Queue state updates using a plug-in design similar to the one described [above](#)
2. When a state update is received on the real-time thread, call [GetCurrentNativeSampleLocation](#) to get the sample location for the start of the current render buffer
3. Perform all necessary update handling using this value as the sample location

NOTES

- This approach yields a sample location value which is accurate within 32 samples
- Event handling must be performed on the real-time render thread, which may not be viable depending on the types of operations that the plug-in must perform
- Event handling cannot be performed in advance to reduce overall system latency

12.29.5.2.2 2. Compute the timestamp as a TOD offset

1. Add a queue for update events which will be used internally within the plug-in's [AAX_IEffectParameters](#) object
2. In [UpdateParameterNormalizedValue](#), enqueue an update event
3. In [GenerateCoefficients](#), call [AAX_IController::GetTODLocation\(\)](#) and [AAX_IController::GetCurrentAutomationTimestamp\(\)](#)
4. Subtract the current TOD value from the automation timestamp to find the number of samples currently lie between the data model location and the render audio location on the timeline
5. Call [AAX_ITransport::GetCurrentNativeSampleLocation\(\)](#) and add the resulting value to the sample offset that was determined in the last step. The sum of these two values is the approximate absolute sample location for the coefficient update.
6. Once this sample location has been calculated, dequeue all pending update events and handle them using the calculated timestamp

The reason that this approach yields an approximate value is that the TOD location and current playback location are both given in terms of the real-time audio workers, and these values continue to progress simultaneously with execution of methods on the automation update thread. As a result, this approach will yield an absolute timestamp that is "late" by between zero and one hardware buffer.

NOTES

- Using this approach it is possible to handle parameter updates in advance to reduce overall system latency
- This approach yields a sample location value which is accurate within one hardware buffer
- This approach uses AAX interface methods that are not supported in older AAX hosts such as Pro Tools 10

12.29.5.2.3 3. Compute the timestamp with improved accuracy using MIDI Beat Clock

You can refine the approach described above by using MBC events to detect the location of playback start.

1. Register a global MIDI node in your plug-in using [AAX_IEffectDescriptor::AddControlMIDINode\(\)](#) with [AAX_eMIDINodeType_Global](#) and the appropriate event mask bitfield for MBC events
2. Override [AAX_IEffectParameters::UpdateControlMIDINodes\(\)](#) to receive MBC data
3. When an MBC Start or Continue event is received, call [AAX_ITransport::GetCurrentNativeSampleLocation\(\)](#) to get the current render location. Cache this value. This value should represent the absolute playback start sample since audio render will not have started before the MBC event dispatch.

4. As in the previous solution, queue relevant update events in `UpdateParameterNormalizedValue`
5. In `GenerateCoefficients`, call `GetCurrentAutomationTimestamp` and add the resulting value to the cached playback start sample location
6. Dequeue all pending update events and handle them using the calculated absolute sample timestamp

NOTES

- Using this approach it is possible to handle parameter updates in advance to reduce overall system latency
- This approach will yield timestamps within a few samples of the actual automation event location on the Pro Tools timeline
- This approach uses AAX interface methods that are not supported in older AAX hosts such as Pro Tools 10

Collaboration diagram for Parameter update timing:



12.30 Token protocol

Communicating parameter state with the host.

12.30.1 On this page

- [An Introduction to Tokens](#)
- [Basic Token Operation](#)

12.30.2 An Introduction to Tokens

One way in which a plug-in can communicate with the "outside world" is through Shared Data Services, also known as the Token System. This is a mechanism that allows Pro Tools to share parameter information with external hardware and software modules. While the AAX SDK only uses the Token System indirectly, knowing how it works will provide a good understanding of how linked parameters should operate.

12.30.2.1 Touch

Touch tokens inform the system of user interaction with a parameter. When a parameter is being touched the system knows to stop sending automation data to the plug-in and just use the SET value of the parameter. It is also used to tell the system when to start/stop recording new automation data.

In AAX, the touch message is sent to the host by [AAx_IAutomationDelegate::PostTouchRequest\(\)](#). The most common way to call this method is via the following methods:

```
class AAX_IEffectParameters
{
    virtual AAX_Result TouchParameter ( AAX_CParamID inParameterID );
    virtual AAX_Result ReleaseParameter ( AAX_CParamID inParameterID
    );
};

class AAX_IParameter
{
    virtual void Touch ();
    virtual void Release ();
};
```

However, AAX plug-ins will rarely need to call these methods directly since the [AAx_CParameter](#) and [AAx_CEffectParameters](#) implementations will automatically handle parameter touch and release tokens whenever a new value is set on the parameter by the plug-in.

Other clients besides the plug-in may touch a parameter. Since the TOUCH token can come from a control surface the touch state will actually come back to the plug-in via:

```
class AAX_IEffectParameters
{
    virtual AAX_Result UpdateParameterTouch (
        AAX_CParamID iParameterID, AAX_CBoolean iTouchState );
};
```

This method is mainly important for [linked parameters](#).

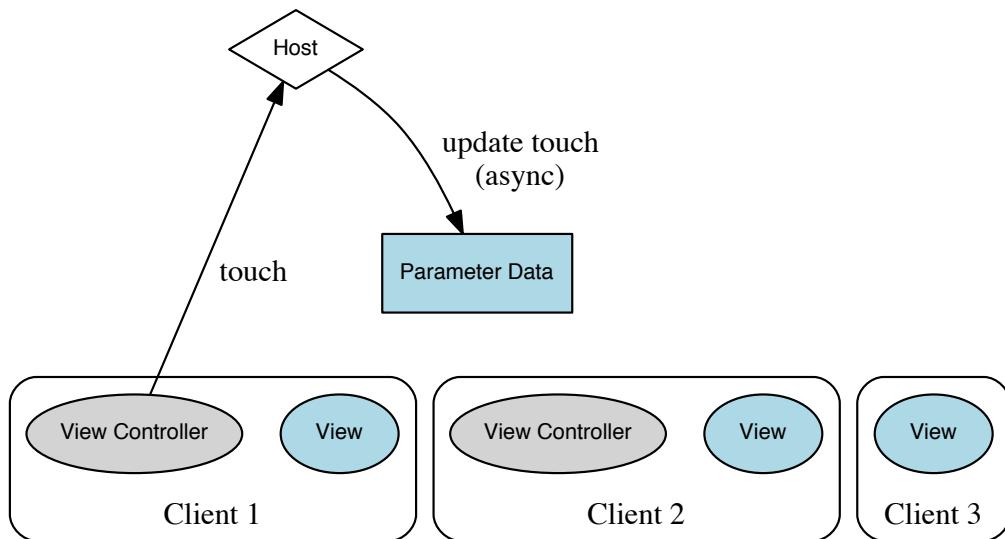


Figure 12.5: Touch request from a view controller, with resulting async touch update

12.30.2.2 Set

SET tokens can come from many different locations: the plug-in GUI, a control surface, loading a chunk or automation playback. Eventually the value of a SET token comes into the plug-in and that's when the internal value of the parameter gets updated. In AAX the SET token will be sent as a result of calling the following method:

```
class AAX_CParameter<T>
{
    void SetValue ( T newValue );
};
```

which will be called from many other supporting methods:

```
class AAX_CParameter<T>
{
    bool SetValueWithBool ( bool value );
    bool SetValueWithInt32 ( int32_t value );
    bool SetValueWithFloat ( float value );
    bool SetValueWithDouble ( double value );
    void SetToDefaultValue ();
    void SetNormalizedValue ( double normalizednewValue );
    bool SetValueFromString ( const AAX_CString & newValueString );
};
```

When a SET token enters the system from the GUI, control surface or automation the value comes back to the plug-in via the following method:

```
class AAX_CEFFECTPARAMETERS
{
    AAX_Result UpdateParameterNormalizedValue (
        AAX_CParamID iParameterID, double aValue, AAX_EUpdateSource inSource);
};
```

At this point the internal contents of the plug-in are set.

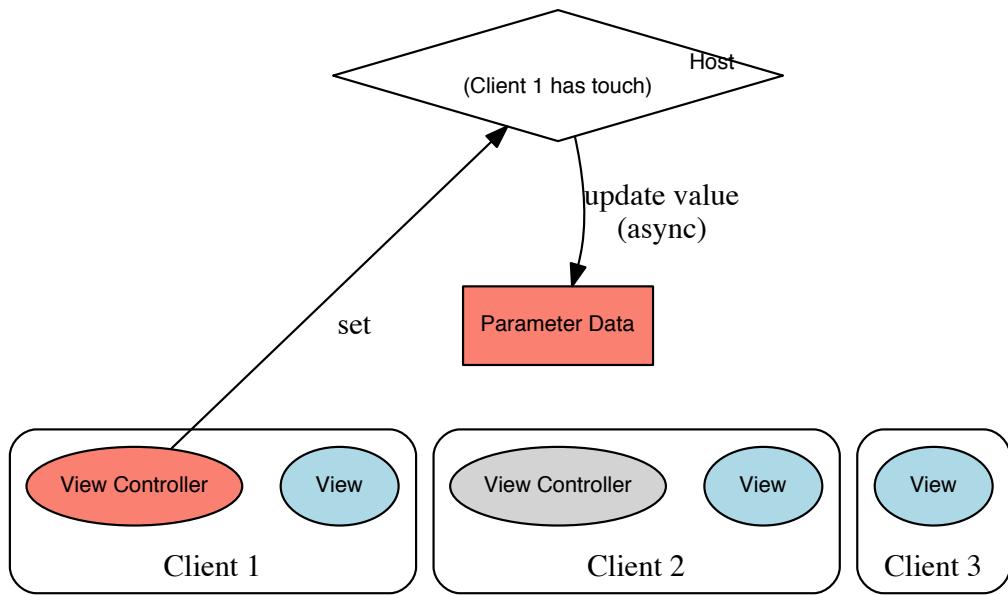


Figure 12.6: Set token asynchronously changes state of the parameter data

12.30.2.3 Update

An update token is generated when the internal value of a parameter has been set. GUIs and control surfaces listen for UPDATE tokens to update the displayed values. In AAX the UPDATE token is sent by calling the following method:

```

class AAX_CParameter<T>
{
    void UpdateNormalizedValue ( double newNormalizedValue );
};

```

All views of the parameter are then asynchronously notified that the value has changed. The plug-in GUI is notified via a call to [AAx_IEffectGUI::ParameterUpdated\(\)](#).

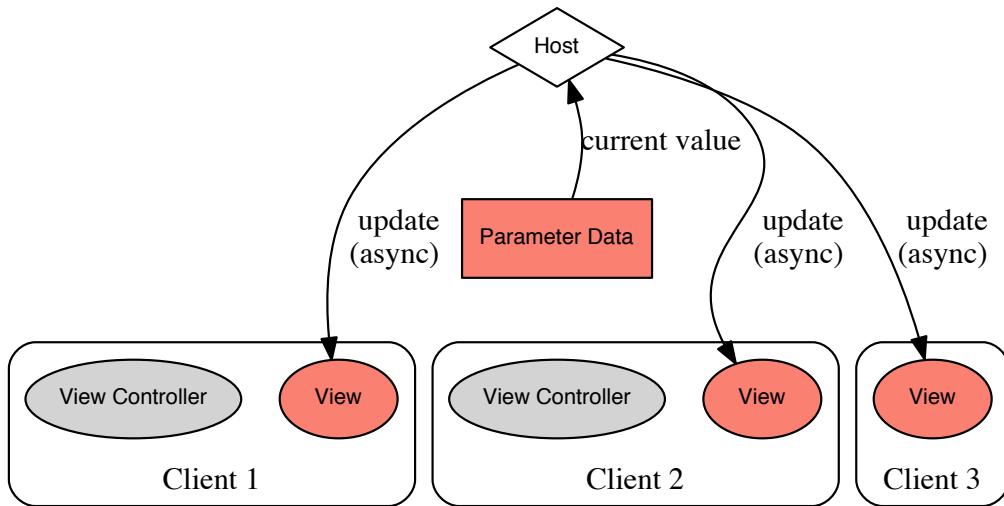


Figure 12.7: Update token triggers async updates to all views

12.30.3 Basic Token Operation

The lists below indicate how the system works in a few different standard update scenarios. To enable logging for these events set `DTF_AUTOMATION=file@DTP_LOW` in the [DigiTrace](#) configuration file. For more detailed information about the sequence of calls used to update parameters in different situations, see [Basic parameter update sequences](#).

12.30.3.1 User Editing

1. User clicks on a parameter in the GUI or grabs a parameter on the controls surface. A TOUCH token should be sent at this point.
2. The user changes the parameter from the GUI or controls surface. A SET token should be sent at this point.
3. The SET token goes into the system and comes back to the plugin via `UpdateParameterNormalizedValue()`.
4. The plug-in updates its internal state and sends an UPDATE token.
5. Repeat steps 2-4 while changing the parameter.
6. The user lets go of the GUI or controls surface. A TOUCH token with the released state should be sent.

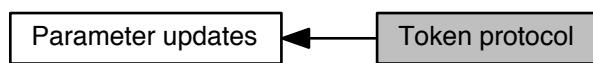
12.30.3.2 Automation Playback

1. The SET token comes from the automation system and enters the plugin via `UpdateParameterNormalizedValue()`.
2. The plug-in updates its internal state and sends an UPDATE token.
3. Repeat steps 1-2 while playing back automation.

12.30.3.3 Chunk Restoring

1. Plug-in loads the chunk.
2. The plug-in sets every parameters value. Another thing to note is that the
3. SetValue() method also contains Touch() and Release() calls. So, while setting every parameter there is a combination of TOUCH and SET tokens sent to the system.
4. The SET tokens comes back to the plugin via UpdateParameterNormalizedValue().
5. The plug-in updates it's internal state and sends out UPDATE tokens.

Collaboration diagram for Token protocol:



12.31 Basic parameter update sequences

Sequence diagrams for some common parameter update scenarios.

12.31.1 On this page

- [User-generated update](#)
- [Automation playback](#)
- [Initialization](#)

Note

To enable logging for these events at run time set DTF_AUTOMATION=file@DTP_LOW in the [DigiTrace](#) configuration file.

12.31.1.1 Notes on threading for these sequences

- Calls from the host into [AAX_IEffectParameters](#) may occur on any thread. In general, the only synchronization that is guaranteed for data model calls in these diagrams is that the call will follow whatever event is indicated as its trigger.
- Calls from the host into [AAX_IEffectGUI](#) will occur on the main application thread unless indicated otherwise in the [AAX_IEffectGUI](#) documentation.
- Host-driven updates to the algorithm context are always synchronized with the real-time processing thread

12.31.2 User-generated update

This is the sequence of calls for a basic, unlinked parameter update triggered by the user. For this sequence, we assume that the edit was triggered by a GUI event.

12.31.2.1 High-level interface calls and events

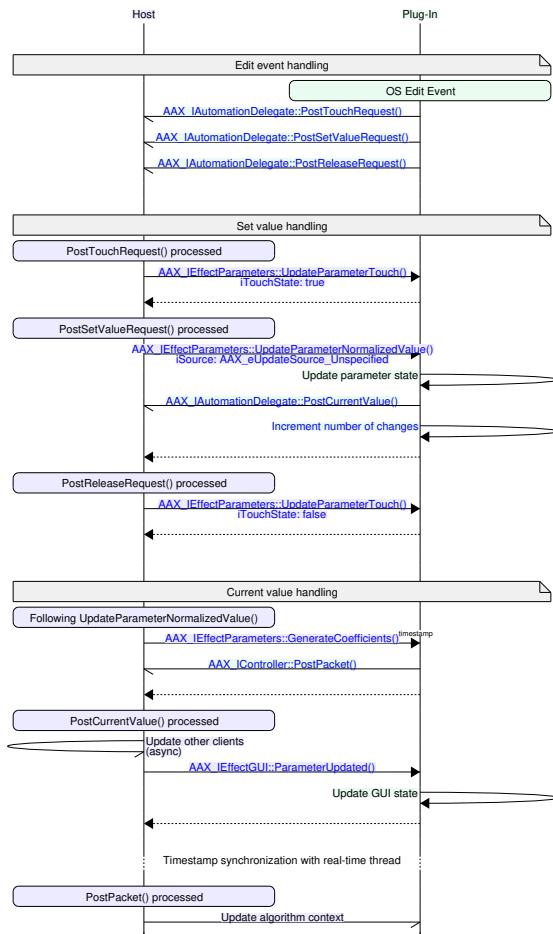


Figure 12.8: High-level sequence of interface calls and events for a parameter update following a user-generated edit

12.31.2.2 Detailed sequence for default implementation

Note that this diagram assumes a GUI implementation that uses `SetParameterNormalizedValue()`. The implementation could also use other parameter set methods, either in `AAX_IEffectParameters` or directly on an `AAX_IParameter`. The overall sequence would remain the same.

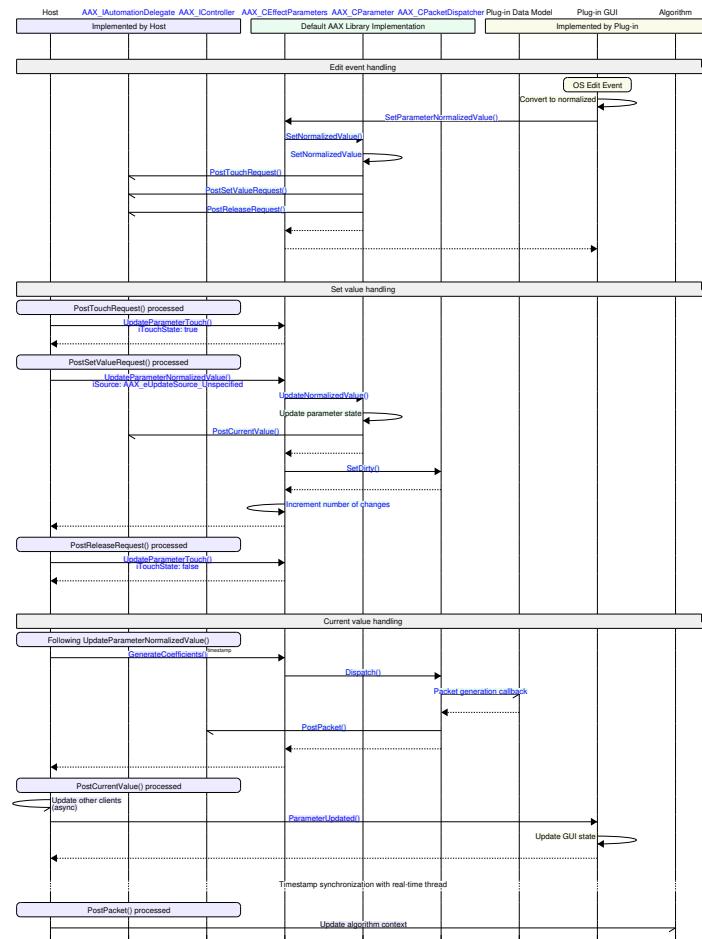
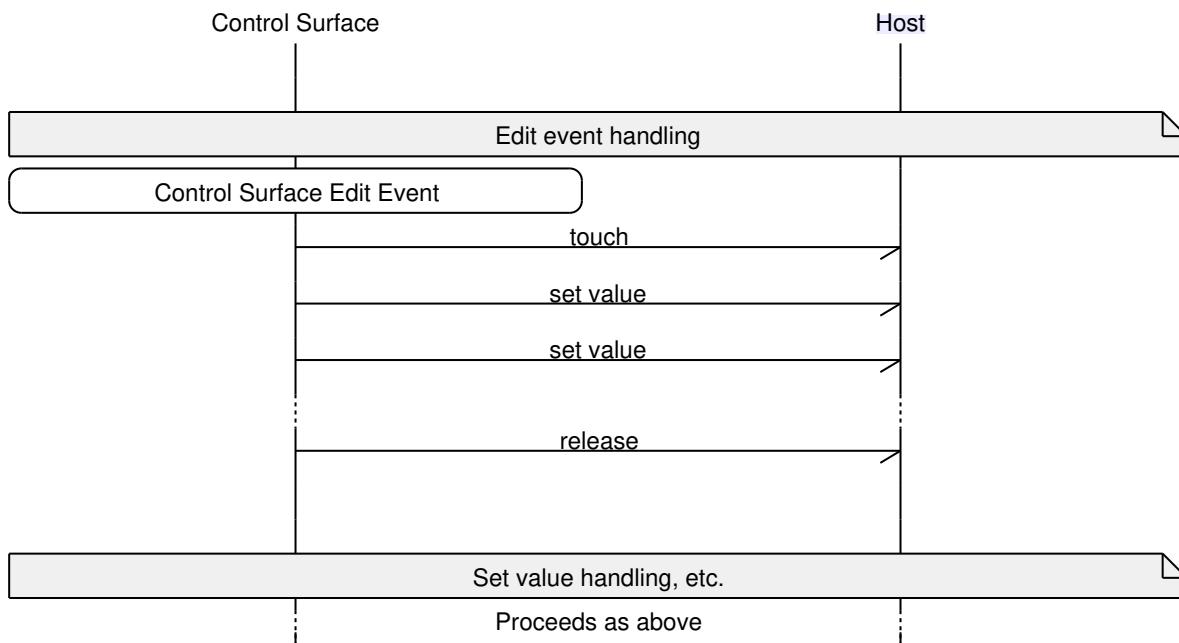


Figure 12.9: Detailed sequence of method calls and events for a parameter update following a user-generated edit on the plug-in GUI

12.31.2.3 Updates from control surfaces

Updates from control surfaces are handled in exactly the same way. In this case, though, the parameter touch, set value, and release tokens are generated by the control surface.



12.31.3 Automation playback

Automation playback handling is similar to the handling for user-generated parameter updates. However, parameters are never touched/released during automation playback. This allows touches from other clients, such as the GUI or control surfaces, to override the automation playback.

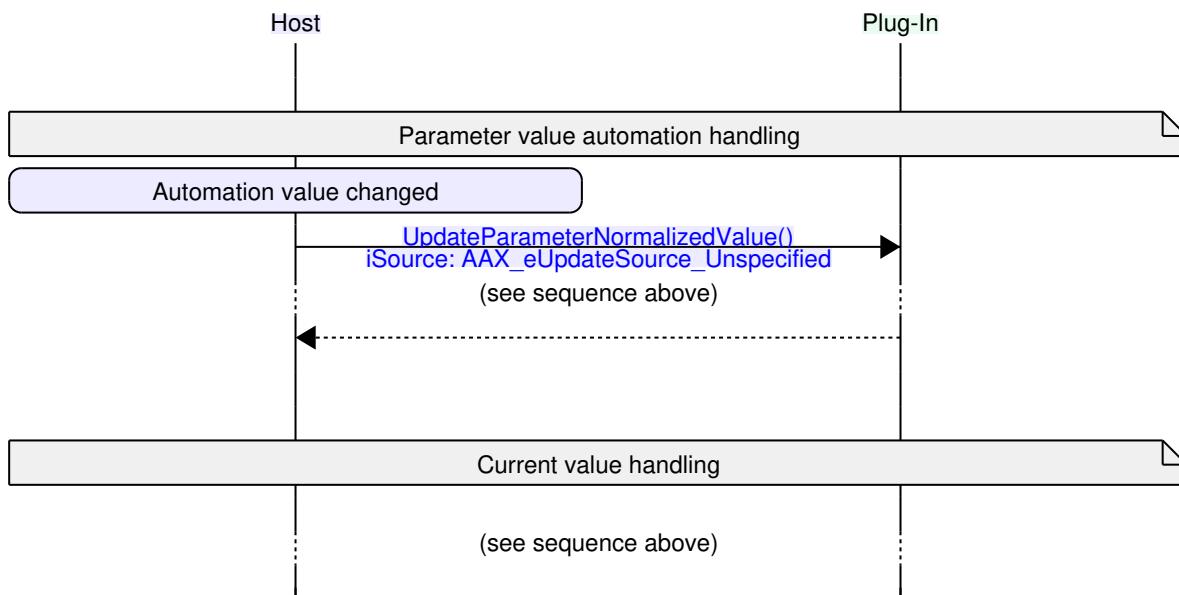


Figure 12.10: Sequence of method calls and events for playback of parameter automation

12.31.4 Initialization

This is the sequence of calls for the initial parameter updates made during data model initialization. Steps that are redundant with sections of the [standard user-generated update sequence](#) are elided.

Todo Update this section with information about default chunk setting, which is a separate step following the pro-

cedure described below.

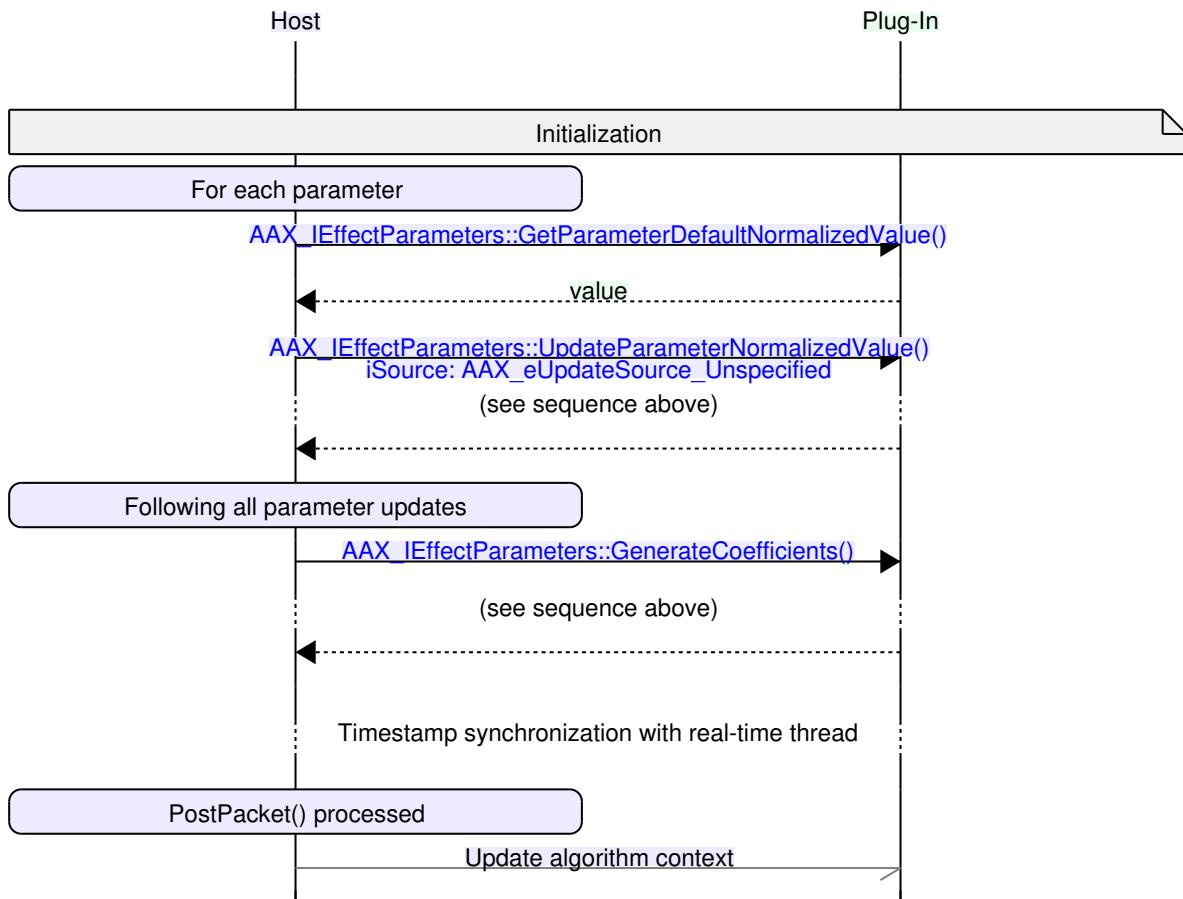
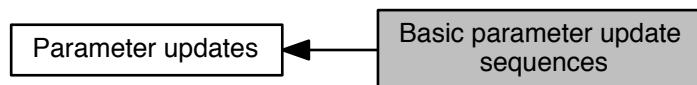


Figure 12.11: Sequence of method calls and events for parameter updates at plug-in initialization

Collaboration diagram for Basic parameter update sequences:



12.32 Linked parameters

How to link parameters.

12.32.1 On this page

- [Basics of Linked Parameters](#)
- [Linked Parameter Operation](#)
- [Changing Tapers](#)

12.32.2 Basics of Linked Parameters

A "linked" parameter can be defined as any parameter whose state is somehow dependent on another parameter. Within this general definition, there are various different kinds of parameter linking:

- Linking behavior can operate one-way or parameters can be reciprocally linked
- Linking between parameters can be one-to-one or one-to-many

12.32.2.1 Basic considerations for parameter linking

Although the concept of parameter linking is simple, implementing linked parameter behavior that is intuitive and consistent can require careful design.

- Parameter interdependencies and constraints can become complex, especially when handling multiple sets of linked parameters.
- Linked parameters that update other parameters during playback can result in subtle timing inconsistencies
- Automated parameters may contain arbitrary and conflicting automation data
- A user may attempt to edit multiple linked parameters simultaneously during playback, e.g. using multiple encoders on a control surface
- A plug-in may contain dependency cycles between interdependent parameters. These cycles can cause undesired behavior that is difficult to debug, especially if it only occurs in certain circumstances such as when loading particular presets.

In all of these cases, a plug-in should provide consistent linked parameter behavior: every automated playback pass should be identical, parameters should never "fight" one another or trigger rapid and unexpected changes in other parameters, parameters should not become "stuck" in a particular state, etc.

Here comes trouble

12.32.2.2 Defining proper linked parameter behavior

A good way to approach parameter linking is to start with an understanding of exactly what behavior you desire.

Here are some behaviors that you probably *don't* want in your plug-in:

- BAD: Parameters are only linked when edited from the plug-in GUI. Users may attempt to edit linked parameters from attached control surfaces or using the host's automation features. The parameters should behave the same way regardless of which method is used to edit them.

- BAD: Parameters try to match all automation data Automation data can be written arbitrarily: Pro Tools doesn't have any restrictions that a user with a pencil tool must draw inside the lines, or a user may attempt to edit multiple parameters on an attached control surface simultaneously. Any parameter that attempts to match both its own automation data and the automation data of another parameter, or any parameter that attempts to set another automatable parameter's state based on its own automation data, will lead to "fighting" during playback of non-conformant automation.
- BAD: Automation data is only written to one lane at a time One approach to parameter linking may be to only write automation data to a single parameter at a time. This could be the parameter that is currently being touched and edited, or it could be a dedicated "master" parameter within the linked group. While this approach can be used to solve some types of conflicts, it can still lead to unnecessarily complex or inconsistent behavior in certain situations: for example, arbitrary automation data can still be written to multiple parameters' automation lanes, or a user can choose to record automation for only one parameter in a set but can skip the "master" parameter. Furthermore, it is difficult if not impossible to properly handle parameters that can be dynamically linked or un-linked using this approach.

With those potential problems in mind, here is a description of how parameter linking *should* behave.

12.32.2.2.1 Correct behavior for linked parameters

Notes

- In this proposal (and throughout the rest of this page) the term "linker" will refer to a parameter that initiates a change and the term "linked" will refer to a dependent parameter that receives the change.
- The following discussion will focus on *automatable* parameters that are *reciprocally linked*. This case tends to be the most complex, with the greatest need for consistency of implementation.

During user-generated real-time edits (from the plug-in GUI or a control surface) both the linker and the linked parameters should be updated. Without this requirement, there would be no parameter linking. In order for this requirement to be enforced consistently the following behaviors must be maintained:

- The linked parameter should not jump to a new value if the user attempts to edit both parameters simultaneously using a control surface.
- To ensure proper automation playback, automation should be written to both the linker and the linked parameters.

When playing back automation, parameters should operate independently and should not attempt to force dependent parameters to a new state. This prevents fighting in the presence of incompatible automation and ensures deterministic automation playback with every playback pass. As above, there are some more subtle behaviors that must be maintained for this to work properly:

- If the user begins a real-time edit during automation playback then the parameter linking behavior should resume as described above
- If the plug-in's algorithm cannot support certain parameter configurations then its automatable parameters should be decoupled from the algorithm using a set of coefficients that is aware of the algorithm's constraints. In this way every combination of parameter states can map to a particular coefficient state, maintaining determinism, and incompatible parameter combinations can simply resolve to the "closest" match in the possible coefficient space during playback of edited parameter automation data.
- Another, simpler approach for plug-ins that do not support arbitrary parameter configurations is to ensure that the problematic parameters are not automatable. Handling non-automatable parameter linking is much easier in general, so consider this approach if automation is not a requirement for some of your plug-in's parameters.

When handling preset changes and plug-in initialization, a similar approach should be taken as with plug-in automation playback. In these cases it is very unlikely that the plug-in's parameters will be left in an incompatible state and attempts at linking may result in unwanted update cycles between inter-dependent parameters or unnecessary coefficient churn. This latter concern can be a real problem for AAX DSP plug-ins that initialize internal algorithmic state based on initial coefficient data.

12.32.2.2.2 Compatibility caveat

This behavior was not possible under the RTAS/TDM format, and many RTAS and TDM plug-ins reverted to workarounds such as writing automation to only one parameter at a time and linking the parameters during playback. Therefore, plug-ins that previously supported linked automatable parameters under the RTAS/TDM format may not be able to both implement this recommended parameter linking behavior and maintain compatibility with automation in saved sessions.

Most of Avid's plug-ins that were available in the RTAS and TDM formats fall into this category and should not be used as examples of proper parameter linking behavior. Instead, use the SDK's DemoGain_LinkedParameters example plug-in as an example of proper linked parameter operation.

12.32.3 Linked Parameter Operation

As described above, the key rule for linked parameters is to link during real-time user edits *only*, and should operate the parameters independently (without linked behavior) during automation playback and preset restore. This rule will simplify many issues: it will prevent conflicts with automation data, avoid potentially strange behaviors when restoring presets, and more.

Here is how the system works WITH linked parameters, using code snippets from the DemoGain_LinkedParameters example plug-in:

12.32.3.1 User Editing

1. User clicks on a parameter in the GUI or grabs a parameter on the controls surface. A TOUCH token should be sent at this point.
 - The touched parameter status comes back to the plug-in. If the parameters are linked the other linked parameter should have a TOUCH token sent. This really should only be done for linked continuous parameters. This is done by overriding the [AAECEffectParameters::UpdateParameterTouch\(\)](#) method.
2. The user changes the parameter from the GUI or controls surface. A SET token should be sent at this point.

```
// ****
// METHOD: UpdateParameterTouch
// ****
AAX_Result DemoGain_Parameters::UpdateParameterTouch ( AAX_CParamID inParameterID,
                                                       AAX_CBoolean inTouchState )
{
    if ( inTouchState )
    {
        AAX_CParamID linkedControl = this->GetLinkedControl ( inParameterID );
        if ( linkedControl )
        {
            this->TouchParameter ( linkedControl );
            mLinkTouchMap.insert ( std::pair<std::string, std::string>( inParameterID, linkedControl ) );
        }
    }
    [...]
}
```

3. The SET token goes into the system and comes back to the plugin via [AAECEffectParameters::UpdateParameterNormalizedValue\(\)](#).
 - If the parameter is linked then the other linked parameter should have its value set for its linked behaviour. The system knows this is a linked parameter so when the value comes back to the plug-in via [UpdateParameterNormalizedValue\(\)](#) it will know not to perform linked behaviors on that value change. To determine if a parameter should set a linked parameter you check it with the [AAECEffectParameters::IsParameterTouched\(\)](#) method.
4. The plug-in updates its internal state and sends an UPDATE tokens for both parameters.

```
// ****
// METHOD: UpdateParameterNormalizedValue
// ****
AAX_Result DemoGain_Parameters::UpdateParameterNormalizedValue (
```

```

    AAX_CParamID inParameterID, double inValue, AAX_EUpdateSource inSource )
{
    AAX_Result      result =
        AAX_CEffectParameters::UpdateParameterNormalizedValue
        ( inParameterID, inValue, inSource );
    bool          touched = this->IsParameterTouched ( inParameterID );

    [...]
    if ( touched && inSource == AAX_eUpdateSource_Unspecified )
    {
        if ( type == eType_Pan )
            this->SetParameterNormalizedValue( linkedControl, (1.0 - inValue) );
        else if ( type == eType_Gain )
            this->SetParameterNormalizedValue( linkedControl, inValue );
    }
}
}

```

5. Repeat steps 2-4 while changing the parameter.
6. The user lets go of the GUI or controls surface. A TOUCH token with the released state should be sent.

- The touched parameter status comes back to the plug-in. If the parameters were linked the other linked parameter should have a TOUCH token with the release status sent. This again is done by overriding the `AAX_CEffectParameters::UpdateParameterTouch()` method.

```

// ****
// METHOD: UpdateParameterTouch
// ****
AAX_Result DemoGain_Parameters::UpdateParameterTouch ( AAX_CParamID inParameterID,
    AAX_CBoolean inTouchState )
{
    if ( inTouchState )
    {
        [...]
    }
    else
    {
        [...]
        this->ReleaseParameter ( iter->second.c_str () );
        [...]
    }

    return AAX_SUCCESS;
}

```

12.32.3.2 Automation Playback

1. The SET token comes from the automation system and enters the plugin via `UpdateParameterNormalizedValue()`.
 - The plug-in will know this is not from the user editing therefore it will NOT set the other linked parameter. Remember ONLY LINK USER EDITING. That way there's no conflicts if the user edited the automation or if the order in which automation arrives at the plug-in changes.
2. The plug-in updates its internal state and sends an UPDATE token.
3. Repeat steps 1-2 while playing back automation.

12.32.3.3 Chunk Restoring

1. Plug-in loads the chuck.
2. The plug-in sets every parameters value.
3. The SET tokens comes back to the plugin via `UpdateParameterNormalizedValue()`.
 - The plug-in will know this is not from the user editing therefore it will NOT set the other linked parameter. Remember ONLY LINK USER EDITING. Hopefully the result of this is that the contents of the chunk will be restored to its exact state.
4. The plug-in updates its internal state and sends out UPDATE tokens.

12.32.4 Changing Tapers

One common use of linked parameters is to change the taper associated with a parameter. For changing tapers there are basically only a two rules you need to follow:

1. When you're loading a new chunk you need to set the taper values first. If a parameter is what updates the taper then set that value first. That way when the value of a parameter is set from a chunk it wont change because of a taper change.
2. Update the taper from the `UpdateParameterNormalizedValue()` method. If the new taper needs to change the value of the parameter you only do so if the user is editing the linked parameter. This still follows the ONLY LINK USER EDITING rule.

```
AAX_Result Simple_Parameters::UpdateParameterNormalizedValue (
    AAX_CParamID inParameterID, double inValue, AAX_EUpdateSource inSource )
{
    // GetLinkedControl() is a user defined method which determines the linked control ID.
    AAX_CParamID linkedControl = this->GetLinkedControl ( inParameterID );
    if ( linkedControl )
    {
        // IsParameterLinkReady() is a built in method of AAX_CEFFECTPARAMETERS which determines if the
        // parameter should perform linked behaviors based on the touch state of the parameter and the
        // source of the UpdateParameterNormalizedValue() call.
        if ( this->IsParameterLinkReady ( inParameterID, inSource ) )
            this->SetParameterNormalizedValue( linkedControl, inValue );
    }

    // Call the inherited method for the original parameter
    AAX_Result result =
        AAX_CEFFECTPARAMETERS::UpdateParameterNormalizedValue
        ( inParameterID, inValue, inSource );
    return result;
}
```

Collaboration diagram for Linked parameters:



12.33 Linked parameter update sequences

Sequence diagrams for some common linked parameter update scenarios.

12.33.1 On this page

- [User-generated update](#)
- [Update from automation playback](#)

Note

To enable logging for these events at run time set DTF_AUTOMATION=file@DTP_LOW in the [DigiTrace](#) configuration file.

12.33.1.1 Notes on threading for these sequences

- Calls from the host into [AAX_IEffectParameters](#) may occur on any thread. In general, the only synchronization that is guaranteed for data model calls in these diagrams is that the call will follow whatever event is indicated as its trigger.
- Calls from the host into [AAX_IEffectGUI](#) will occur on the main application thread unless indicated otherwise in the [AAX_IEffectGUI](#) documentation.
- Host-driven updates to the algorithm context are always synchronized with the real-time processing thread

See also

[Basic parameter update sequences](#)

12.33.2 User-generated update

This is the sequence of calls for a parameter update triggered by the user. For this sequence, we assume that the edit was triggered by a GUI event. Updates from control surfaces are handled in exactly the same way, except that the parameter touch, set value, and release tokens are generated by the control surface.

In this example the updated parameter is reciprocally linked to one other parameter. These are the "linker" and "linked" parameters, respectively.

This procedure is very similar to the non-linked case described [here](#). In the diagrams below, red arcs and pink section headings are used to indicate events that are specific to the linked parameter case.

Notes:

1. This sequence shows the linked parameter reciprocally issuing a touch on the linker parameter. The touch fails since the linker parameter is already touched at this time. If the roles were reversed (if an edit occurred on the linked parameter) then this touch would succeed.
2. The host flags all set value tokens that are triggered by a plug-in within the scope of [AAX_IEffectParameters::UpdateParameterNormalizedValue\(\)](#) "UpdateParameterNormalizedValue()". When those set value tokens are processed they result in additional calls to [AAX_IEffectParameters::UpdateParameterNormalizedValue\(\)](#) "UpdateParameterNormalizedValue()". The host sets `iSource` to [AAX_eUpdateSource_Parameter](#) for each of these subsequent calls to indicate that the update originated from within a parameter update event.
3. [IsParameterLinkReady\(\)](#) returns `true` during the linker parameter update because the update source is unknown and the parameter is touched. Both conditions must be true in order for the linking logic to proceed with setting linked parameters' values.
4. [IsParameterLinkReady\(\)](#) returns `false` during the linked parameter update because the source is [AAX_eUpdateSource_Parameter](#). This prevents update cycles for reciprocally linked parameters, as demonstrated [here](#).

12.33.2.1 High-level interface calls and events

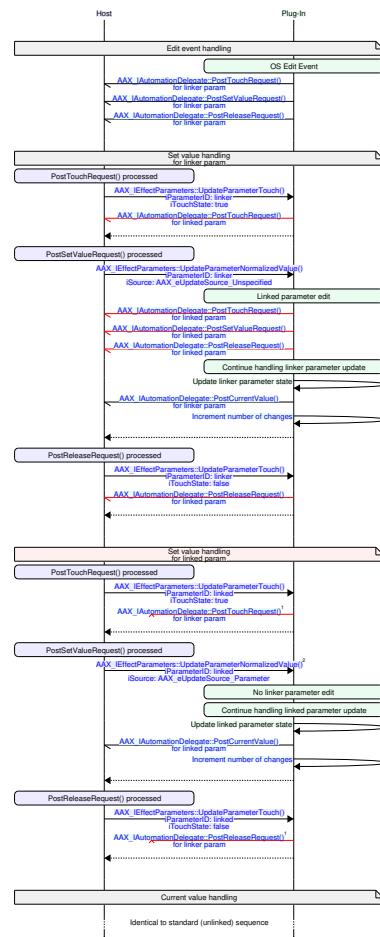


Figure 12.12: High-level sequence of interface calls and events for a reciprocally linked parameter update following a user-generated edit

12.33.2.2 Detailed interface calls and events

Note that this diagram assumes a GUI implementation that uses `SetParameterNormalizedValue()`. The implementation could also use other parameter set methods, either in `AAX_IEffectParameters` or directly on an `AAX_IParameter`. The overall sequence would remain the same.

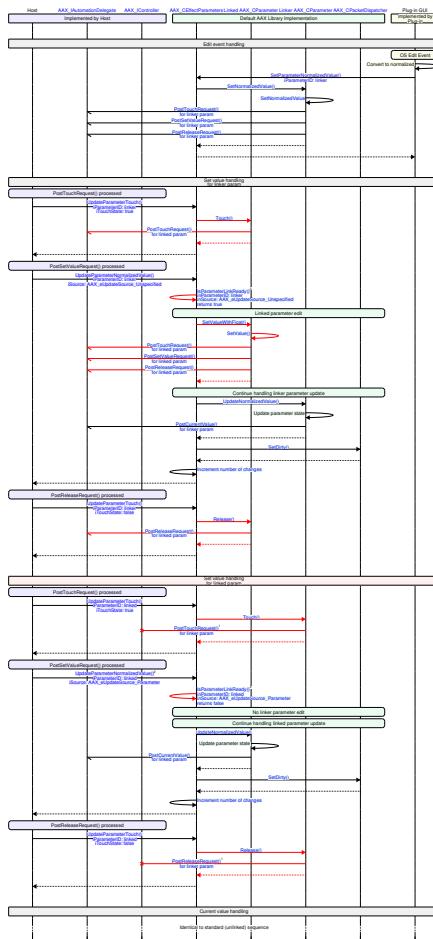


Figure 12.13: Detailed sequence of method calls and events for a reciprocally linked parameter update following a user-generated edit on the plug-in GUI

12.33.3 Update from automation playback

Since all parameter linking occurs while recording automation, automation playback is very simple. The automation lanes may contain any arbitrary values, so, in order to avoid fighting between incompatible values, the plug-in should respect all automation values during playback.

Notes:

1. `IsParameterLinkReady()` returns `false` during automation playback because the updated parameter is not touched. This ensures that automation playback will proceed with the written values and also guarantees that the user will always be able to override the automation using a control surface encoder or GUI editor.

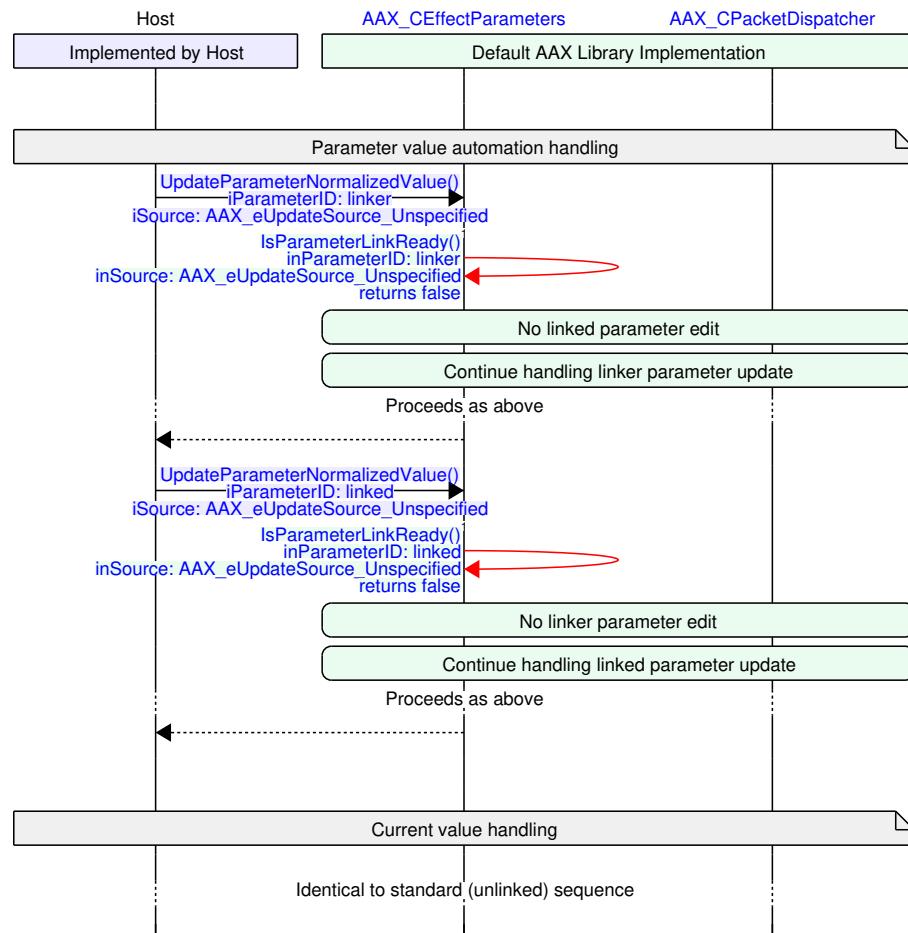
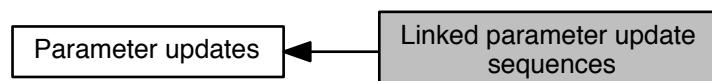


Figure 12.14: Sequence of method calls and events during automation playback with linked parameters

Collaboration diagram for Linked parameter update sequences:



12.34 Plug-in type conversion

Specification for valid conversions between plug-in types.

12.34.1 About this specification

The specification on this page defines the valid AAX plug-in type conversions. An AAX host may use this specification to perform automatic type conversions. For example:

- When a session that was saved with a DSP plug-in Type is opened on a Native system the saved DSP Type should be converted to its Native counterpart.
- When a session saved with the free version of a plug-in is opened on a system that has the full version installed then the saved free Type may be converted to the full version.

12.34.2 Terminology

In this specification the term "Type" refers to a specific configuration of an AAX plug-in.

Each Type is uniquely identified by a combination of five values:

- Manufacturer ID
- Product ID
- Plug-In ID
- Sample rate bit mask
- Architecture

Each Type is defined by a particular call to [AAX_IComponentDescriptor::AddProcessProc_Native](#) or [AAX_IComponentDescriptor::AddProcessProc_TI](#) in the plug-in's description method.

The Plug-In ID is defined as one of [AAX_eProperty_PluginID_Native](#) or [AAX_eProperty_PluginID_TI](#).

In this specification, the format for describing an ID is:

[ID triad + sample rate + architecture]

Where the ID triad may be expanded to [[Manufacturer ID] [Product ID] [Plug-In ID]]

- Explicit values are given in plain face
- Arbitrary constant values are given in *bold face*
- Wildcard values are given in *italics*

For example:

- [*ID triad* + *any sample rate* + Native]
 - Defines all Type identifiers with the same ID triad and the Native architecture, regardless of sample rate.
- [[*Manufacturer ID*] [*Product ID*] [*any ID*]] + *sample rate* + Native
 - Defines all Type identifiers with the same Manufacturer and Product IDs, the same sample rate, and the Native architecture, but with any plug-in ID.

12.34.3 Scope of this specification

For the purposes of this specification we are not concerned with AudioSuite Types. Currently these Types are never type-converted.

In theory, an AAX host may apply a "partial" type conversion by swapping between different ProcessProcs without destroying and re-building any of the plug-in's objects (data model, GUI). For the purposes of this specification we are not concerned about whether a given conversion is "partial" or "total"; all conversions are treated the same.

Plug-in conversions are also required between Types of different stem formats. In fact, the supported stem format is an integral part of a Type's unique identifier. This specification ignores the question of stem formats entirely; we assume that each Type supports all necessary stem formats for legal conversion with other Types.

In general, type swapping rules for deprecated types and related types are equivalent. See [Type deprecation](#) for more information about the differences between related and deprecated types.

12.34.4 Topological constraints

- All Types included in a single [AAX_ICollection](#) must have the same Manufacturer ID
- All Types included in a single [AAX_IEffectDescriptor](#) must have the same Manufacturer ID and Product ID
- The Sample rate bit masks for two Types that share all other identifiers must be non-overlapping. I.e. `0x0 == sr_mask1 & sr_mask2`
- Two Types may not be registered that differ only in their Architecture
- Type relationships may only be defined between mutually exclusive Types. Types are mutually exclusive if:
 - Their sample rate support is non-overlapping
 - The "before" Type's ID triad is not associated with any Type in the plug-in

12.34.5 Implicit conversions

The AAX host should automatically convert between Types within the following IDs:

- [*ID triad + any sample rate + Architecture*]
- [[*Manufacturer ID* [*Product ID*] [*any ID*]] + *sample rate + any architecture*]

These conversions occur only if both Types are included in the plug-in when the conversion is made. Consider the following scenario:

1. A session is saved including an plug-in instance with the following Type identifier: [My ID triad + 48 kHz + TI]
2. The plug-in is updated to a version that does not include this Type identifier, but that does include [My ID triad + 48 kHz + Native]
3. The session is opened on a Native system

In this scenario, if the plug-in had not been updated then an automatic conversion would occur. However, since the plug-in no longer includes the saved Type identifier, no automatic conversion occurs.

A plug-in can work around this situation by including the "old" Type identifier as a Related (or Deprecated) Type (see [Explicit conversions](#).)

When a plug-in instance is saved after making an implicit conversion, plug-ins may be saved with the session using their new Type identifier. This is not required. For example, Pro Tools will prefer to save plug-in instances that were converted from DSP types as DSP, even if they were converted to corresponding Native types when the session was loaded onto and saved from a native Pro Tools system.

12.34.6 Explicit conversions

AAX includes properties that allow a plug-in to define explicit relationships between different Types. These properties only operate on ID triads. Each property can be associated with an array of ID triads to define a one-to-one or a many-to-one association between the given ID triads and the specific Type to which the property is attached.

- [AAX_eProperty_Related_DSP_Plugin_List](#) / [AAX_eProperty_Deprecated_DSP_Plugin_List](#)
 - All of the given ID triads specify TI Types
- [AAX_eProperty_Related_Native_Plugin_List](#) / [AAX_eProperty_Deprecated_Native_Plugin_List](#)
 - All of the given ID triads specify Native Types

The AAX host should convert between related Types within the following constraints:

- [[[Manufacturer ID] [Related Product ID] [Related TI Plug-In ID]] + *sample rate* + TI]
– -> [[[Manufacturer ID] [New Product ID | Related Product ID] [New Plug-In ID]] + *sample rate* + *any architecture*]
- [[[Manufacturer ID] [Related Product ID] [Related Native Plug-In ID]] + *sample rate* + Native]
– -> [[[Manufacturer ID] [New Product ID | Related Product ID] [New Plug-In ID]] + *sample rate* + *any architecture*]

These conversions will occur regardless of whether the related ID triad is used for any of the Types in the plug-in.

When a plug-in instance is saved after making an explicit conversion, all plug-ins are saved with the session using their new Type identifier.

Host Compatibility Notes Pro Tools versions prior to Pro Tools 12.3 do not allow explicit type conversion between types with different product ID values. Beginning in Pro Tools 12.3 both the product ID and the plug-in ID may differ between explicitly related types.

12.34.7 Type deprecation

There are two varieties of explicit plug-in type association: related types and deprecated types.

Properties that create a related type association:

- [AAX_eProperty_Related_Native_Plugin_List](#)
- [AAX_eProperty_Related_DSP_Plugin_List](#)

Properties that create a deprecated type association:

- [AAX_eProperty_Deprecated_Native_Plugin_List](#)
- [AAX_eProperty_Deprecated_DSP_Plugin_List](#)
- [AAX_eProperty_PluginID_Deprecated](#)

With a few exceptions, these two types of explicit association are treated identically. These are the ways in which deprecated plug-in type association differs from related plug-in type association:

- If plug-in types A and B are both installed, and if type A deprecates type B, then type B will be excluded from all insert lists in Pro Tools and will be made invisible to the user.
- If plug-in types A and B are both installed, and if type A deprecates type B, then instances of type B will be swapped to type A when a session containing saved instances of type B is opened.

- Upon the next session save following a swap due to a deprecated type association, the plug-in instance will be saved into the session using the ID of the new plug-in type. Therefore deprecated type swaps do not round-trip when moving between multiple systems if some of the systems have the deprecated types, but not the deprecating types, installed.

Type deprecation should only be used when a new version of an effect completely replaces an old version of the effect. For all other situations, type relationship should be used.

Host Compatibility Notes

- Pro Tools versions before Pro Tools 12.3 treat deprecated and related type associations identically and do not support type deprecation features
- Media Composer does not support type deprecation features
- VENUE does not support type deprecation features

Collaboration diagram for Plug-in type conversion:



12.35 The Avid Component Framework (ACF)

12.35.1

How the AAX C++ interfaces work.

The objects and interfaces in AAX are based on the Avid Component Framework (ACF). The ACF is Avid's implementation of COM, and is the framework that AAX, as well as AVX (Avid Video Externsions) plug-ins are built on.

ACF can be considered an implementation detail of the AAX SDK; the SDK is written to protect plug-in developers from the intricacies of ACF, and it is not necessary to understand ACF or COM in order to use the SDK.

12.35.2 More details

As in COM, ACF draws a distinction between the concept of an object and the concept of an interface. An object is treated as a "black box" of code, whereas an interface is a class of pure virtual methods that allows one to access the functionality inside the object. An object in ACF is represented by the [IACFUnknown](#) interface, which is binary compatible with the COM class `IUnknown`. (Likewise, [IACFUnknown](#) follows the same reference counting rules as `IUnknown` objects.) This interface allows a client to get pointers to other interfaces on a given object using the [QueryInterface\(\)](#) method.

Reference counting is an important aspect of both COM and ACF. Simply put, reference counting is the practice of tracking all references to an object, so that a program can determine when the object can safely be deleted. The AAX SDK library handles this reference counting behind the scenes, so plug-ins that call into the SDK library to manage their component interfaces will not leak references.

Many additional resources can be found both online and print that cover COM and reference counting in greater detail.

12.35.3 ACF interfaces in AAX

The binary interface between an AAX plug-in and host is defined by a series of ACF interfaces. Each of these interfaces inherits from [IACFUnknown](#). The implementation of each ACF interface typically uses `CACFUnknown`, a utility class that provides basic reference counting and additional fundamental ACF details to satisfy [IACFUnknown](#).

These ACF interfaces may be implemented by either the AAX plug-in or the host. The host retains a reference to each interface that is implemented by the plug-in in order to call methods on the plug-in's implementation. Correspondingly, the plug-in retains references to various interfaces that are implemented by the host, and may call host methods via these interfaces.

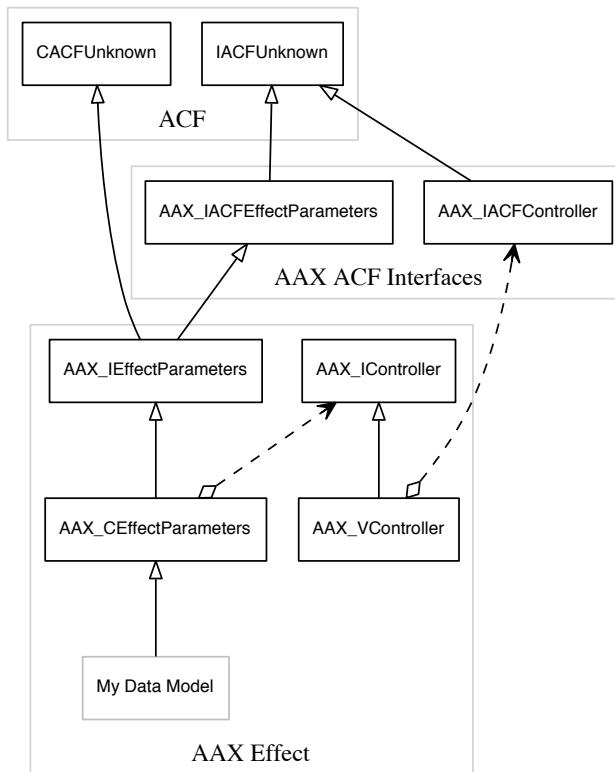


Figure 12.15: ACF interfaces: [AAX_IACFEffectorParameters](#) and [AAX_IACFController](#)

The figure above demonstrates this design: the plug-in implements [AAX_IACFEffectorParameters](#) directly, and retains a reference to an [AAX_IACFController](#) that is implemented by the host.

In order to implement [AAX_IACFEffectorParameters](#), [AAX_IEffectParameters](#) inherits from [CACFUnknown](#) and implements [QueryInterface\(\)](#) to ensure that the [IACFUnknown](#) interface is implemented. The rest of the implementation of [AAX_IACFEffectorParameters](#) is contained in [AAX_CEffectParameters](#) and the plug-in's custom data model class.

The reference to [AAX_IACFController](#) is managed by a versioned implementation class. For more information about this design, see below.

12.35.4 Using ACF interfaces

Depending on where an interface is implemented, there are two specific ways to acquire a reference to the underlying AAX object from an [IACFUnknown](#) pointer:

- Host-provided interfaces
- Plug-in interfaces

12.35.4.1 Host-provided interfaces

Interfaces that are managed by the host must be carefully version-controlled in order to maintain compatibility with many different host versions. The AAX SDK includes "AAX_V" classes to handle this versioning. AAX_V classes

are concrete classes that query the host for the correct version of the requested interface. These classes can also handle re-routing deprecated calls and other complicated versioning logic.

To create an AAX_V object, pass an [IACFUnknown](#) pointer to the underlying host-managed interface in to the AA \leftarrow X_V class' constructor. ACF reference counting is handled automatically by the object's construction and destruction routines, so no additional calls are necessary to acquire and release the reference.

```
#include "AAX_VController.h"

void SomeFunction (IACFUnknown * inController)
{
    // When object is created, a reference is acquired
    AAX_VController theController (inController);

    //
    // ...
    //

    // When object goes out of scope, the reference is released
}
```

12.35.4.2 Plug-in interfaces

Interfaces to objects that are owned by the plug-in always have a known version and therefore do not require AAX_V object management. Instead, these interfaces must be acquired and released directly using ACF.

```
#include "AAX_UIDs.h"
#include "AAX_ASSERT.h"
#include "AAX_IEffectParameters.h"
#include "acfunknown.h"

void SomeFunction (IACFUnknown * inController)
{
    // When interface is queried, a reference is acquired
    if (inController)
    {
        AAX_IEffectParameters* myEffectParameters = NULL;
        ACFRRESULT acfErr = ACF_OK;
        acfErr = inController->QueryInterface(
            IID_IAAXEffectParametersV1,
            (void**)&myEffectParameters);

        AAX_ASSERT(ACFSUCCEEDED(acfErr));
    }

    //
    // ...
    //

    // The reference must be explicitly released when finished
    if (myEffectParameters)
    {
        myEffectParameters->Release();
        myEffectParameters = NULL;
    }
}
```

12.35.5 Interface versioning in AAX

The ACF-based interface used by AAX is designed to allow additional features to be added to the architecture. This can be achieved via the addition of new kinds of interfaces (e.g. [AAX_IEffectDirectData](#)) or by extending the existing interfaces. In this section, we will describe an approach for interface extension.

First, here is a more complete picture of "version 1" of the [AAX_IACFEffectorParameters](#) and [AAX_IACFController](#) interfaces, including a possible host implementation of [AAX_IACFController](#):

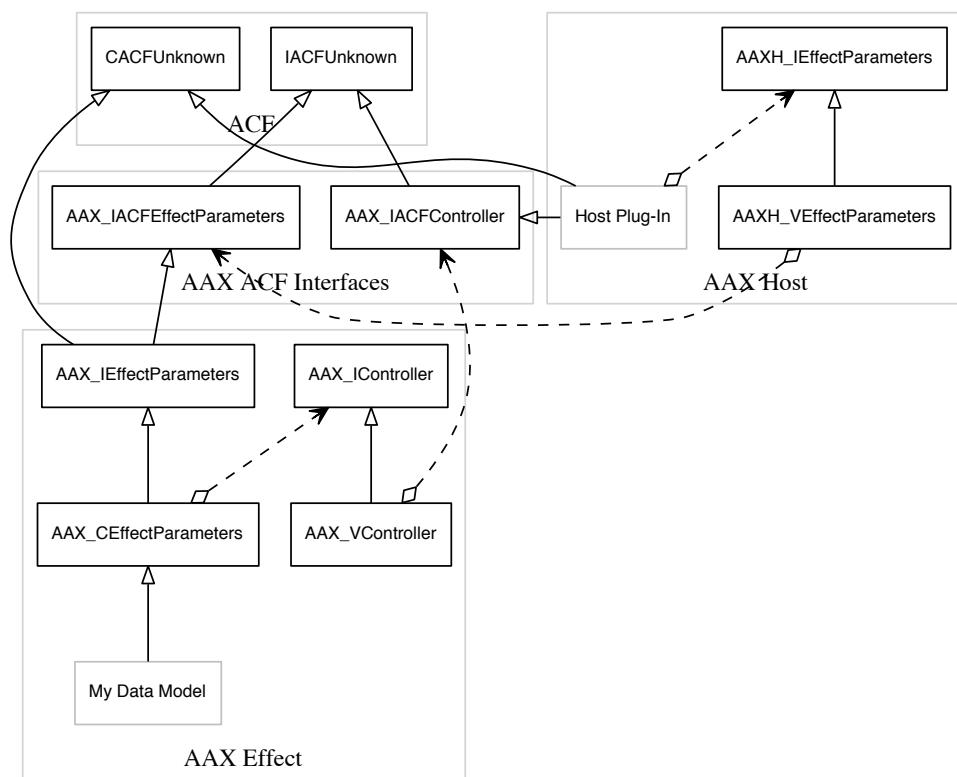


Figure 12.16: ACF interfaces: AAX_IACFEffectParameters and AAX_IACFController (with possible host design)

To extend these interfaces, new "version 2" interfaces are created that inherit from the original interface classes. Although any version 1 method could be called on the new version 2 class, references to each interface are retained by the client in order to clarify the specific version in which each method was introduced.

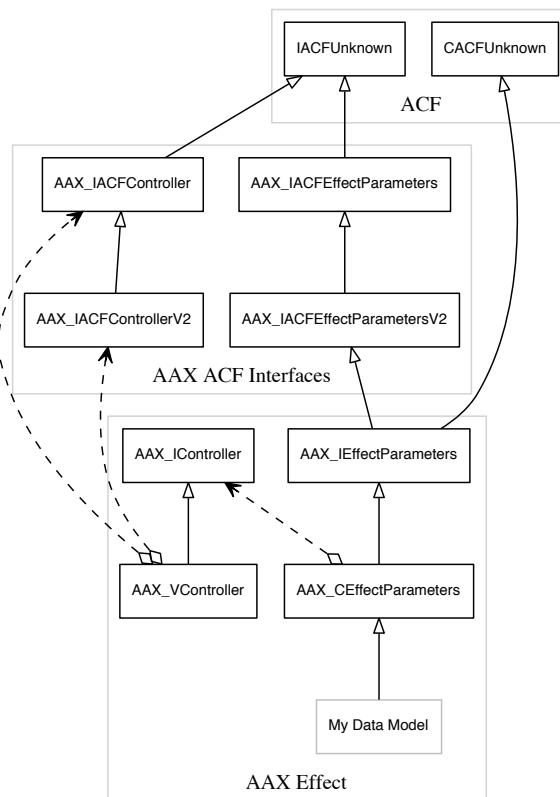


Figure 12.17: Adding a new version to **AAX_IACFEffectorParameters** and **AAX_IACFController**

In this example, if the plug-in is loaded by an older host, the reference to **AAX_IACFControllerV2** will return as NULL, and calls to the V2 methods in **AAX_IController** will return an "unimplemented" error code. Similarly, if a plug-in that only implements **AAX_IACFEffectorParameters** is loaded into a host that supports **AAX_IACFEffectorParametersV2**, that host will receive a NULL reference to the newer interface version and will only be able to call methods on the plug-in's implementation of the original interface.

As a final example, here is a possible design involving new versions of both **AAX_IACFEffectorParameters** and **AAX_IACFController**, with an example design for the host's implementation as well as the plug-in's:

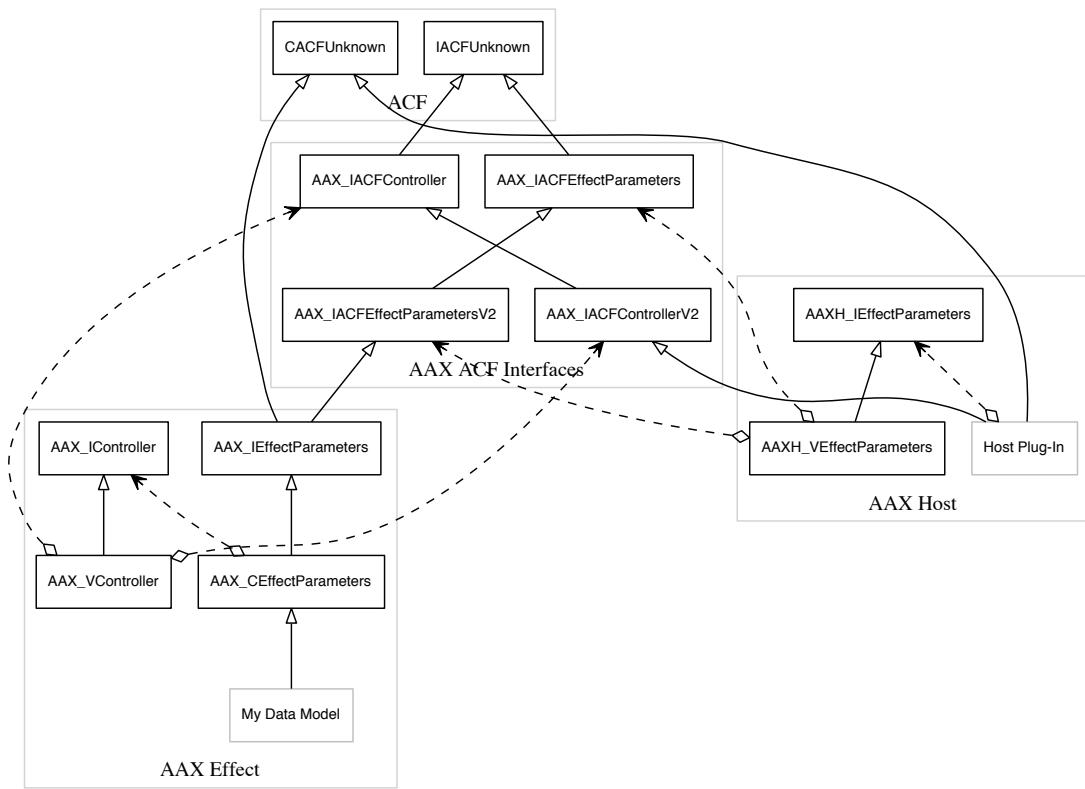


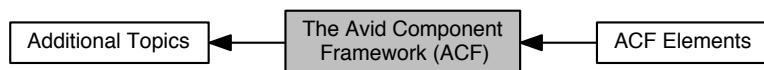
Figure 12.18: Complete design example with versioned ACF interfaces

Documents

- [ACF Elements](#)

ACF classes that are used by common AAX interfaces.

Collaboration diagram for The Avid Component Framework (ACF):



12.36 ACF Elements

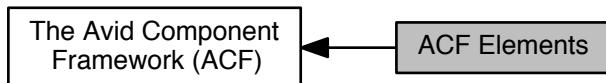
12.36.1

ACF classes that are used by common AAX interfaces.

Classes

- interface [IACFUnknown](#)
COM compatible IUnknown C++ interface.

Collaboration diagram for ACF Elements:



12.37 AAX Host Guides

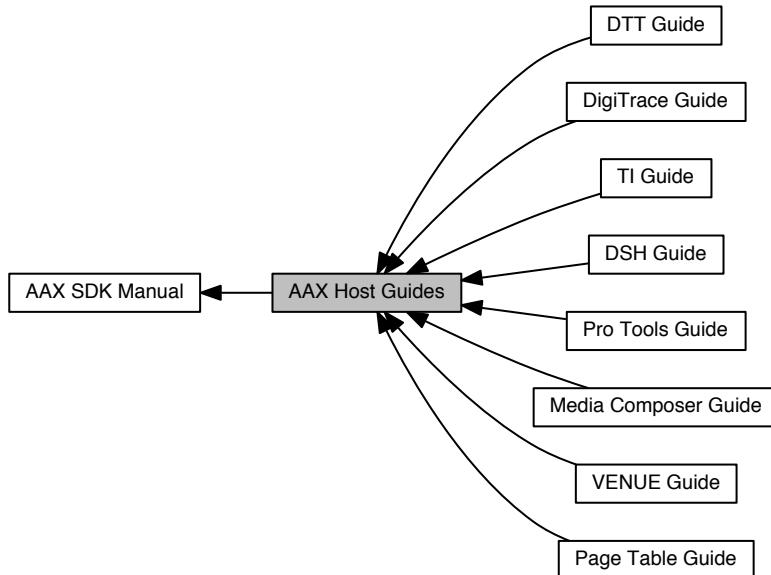
12.37.1

Documentation for specific AAX host environments.

Documents

- [Pro Tools Guide](#)
Details about using AAX plug-ins in Pro Tools.
- [Media Composer Guide](#)
Details about using AAX plug-ins in Media Composer.
- [TI Guide](#)
How to write AAX plug-ins for Avid's TI-based platforms.
- [Page Table Guide](#)
How to map a plug-in's parameters to control surfaces.
- [DigiTrace Guide](#)
How to add tracing to your plug-ins and view logging from the plug-in host.
- [DSH Guide](#)
How to test basic functionality of AAX plug-ins using DSH test tool.
- [DTT Guide](#)
How to automate different test scenarios for DSH.
- [VENUE Guide](#)
Details about using AAX plug-ins in VENUE live sound systems.

Collaboration diagram for AAX Host Guides:



12.38 Pro Tools Guide

Details about using AAX plug-ins in Pro Tools.

12.38.1 Contents

- [About this document](#)
- [Processing modes](#)
- [Requirements for AAX plug-in compatibility with Pro Tools](#)
- [Audio Engine Behavior and Features](#)
- [Basic plug-in operation](#)
- [Optional plug-in features](#)
- [Debugging AAX plug-ins](#)
- [Troubleshooting common AAX plug-in failures](#)
- [Using DigiOptions](#)
- [Compatibility Notes](#)

12.38.2 About this document

This guide discusses specific details related to using AAX plug-ins with Pro Tools, such as loading and initialization procedures, GUI hosting, and other application-specific features. This guide is not intended to provide complete documentation for the Pro Tools application. For more information about the features, functionality, and use of Pro Tools see the Pro Tools Reference Guide, available for download from the Avid web site.

This guide is a work in progress, and is extended as needed to describe different aspects of and caveats to the Pro Tools implementation of the AAX host spec.

12.38.3 Processing modes

Pro Tools supports three AAX processing modes: AudioSuite, AAX Native, and AAX DSP.

- AudioSuite plug-ins perform non-real-time, random-access, file-based processing entirely on the host CPU.
- AAX Native plug-ins perform real-time, linear, non-destructive processing entirely on the host CPU. Native plug-ins are also used by Pro Tools to perform offline rendering.
- DSP plug-ins perform real-time, linear, non-destructive processing on DSP-accelerated hardware, with non-real-time tasks performed on the host CPU.

Each of these processing modes offers specific advantages and trade-offs in functionality, power, and development effort, and plug-in developers may choose to develop only for specific processing modes if the features provided by those modes are required by the plug-in.

12.38.3.1 Real-time processing

Real-time processing allows users to operate plug-ins in live signal paths or in complicated audio routing schemes when the future input data is not known.

Plug-ins operating in real-time are clients of the Pro Tools automation system, meaning that control movements can be dynamically recorded and played back with the audio track, written by hand onto the Pro Tools timeline for future playback, and/or edited by and broadcast to attached control surfaces.

To instantiate a plug-in for real-time processing in Pro Tools, click on an insert slot in the desired track and select the plug-in from the menu that appears

12.38.3.1.1 Native real-time processing

When an AAX plug-in is run natively, all of its components and processing elements are loaded into the host environment. The host CPU handles the plug-in's real-time audio processing as well as its data model, GUI, and other tasks.

12.38.3.1.2 DSP real-time processing

When an AAX plug-in is run with Avid's DSP-enabled hardware, the plug-in's real-time processing code is loaded onto the external DSP device, while the remaining plug-in modules continue to be run by the host CPU. Each DSP in this system provides dedicated processing capacity that is not shared with an OS or other processes, and therefore this architecture allows users to achieve highly reliable and deterministic low-latency processing even when many DSP plug-ins are instantiated.

Figure 1: Real-time insert slots in the Pro Tools Edit and Mix windows.

12.38.3.1.3 CPU reporting

To guarantee absolute reliability, AAX DSP plug-ins are required to report their worst-case performance metrics to Pro Tools. Pro Tools uses this information to ensure that each DSP in the system will be loaded with only the number of plug-ins that it can support given a worst-case processing load.

12.38.3.2 Non-real-time processing (AudioSuite)

The non-real-time AudioSuite processing mode is file-based, meaning that the results of AudioSuite processing are applied destructively to audio files (generally to new, empty files provided by Pro Tools.) AudioSuite processing can only be performed on preexisting blocks of audio.

There are two primary ways to apply AudioSuite processing in Pro Tools. The first way is to selectively apply the processing algorithm to the audio tracks and clips that are selected on the Pro Tools timeline. This is known as "destructive" processing, because the original audio track is replaced by the new processed audio track. There are no limitations governing the amount of time required to process a track in this manner.

Audio Suite plug-ins also have a second optional mode in which they can run. This is referred to as Preview mode. The Preview feature allows you to monitor the audio processing applied to a track in semi-real-time. Because this is a real-time process, it is not applicable to all types of file based processing. You may elect not to support this mode in your plug-in if its algorithm does not lend itself to real-time, linear processing. Preview mode is implemented in a non-destructive manner, as Preview mode exists for auditioning only with no actual replacement of audio data on the Pro Tools timeline.

To instantiate an AudioSuite plug-in in Pro Tools, select the plug-in from the "AudioSuite" menu in the Pro Tools application menu bar

12.38.3.3 Multichannel and Multi-Mono

Pro Tools supports various surround stem formats throughout the entire signal chain, including multi-channel processing through AAX plug-ins.

Pro Tools also allows a plug-in to function in multi-mono mode if the plug-in does not explicitly support certain channel formats. In multi-mono mode, Pro Tools instantiates a separate instance of a plug-in for every channel in the track. In this mode, plug-in controls across all channel-instances in a multi-mono collection are linked by default, though channels can be unlinked by toggling the blue link button in the plug-in header and selecting the channel whose controls you wish to modify.

For more information about multi-mono mode, please refer to the Pro Tools Reference Guide.

12.38.4 Requirements for AAX plug-in compatibility with Pro Tools

In addition to implementing the client-side AAX API, all Pro Tools plug-ins must:

1. Be installed to the AAX plug-ins directory
2. Use a valid file name
3. Be signed with a valid digital signature

Note

Digital signatures for plug-ins are not required in Pro Tools developer builds

12.38.4.1 Install directories

AAX plug-ins must be installed in the system's AAX Plug-Ins directory. See [Building your plug-in installer](#) for more information about creating a plug-in installer.

Plug-ins that are uninstalled but still present on the system are placed into the "Plug-Ins (Unused)" directory, which is located next to the Plug-Ins directory.

Pro Tools will also search for a Plug-Ins directory next to the actual Pro Tools application, and this directory will be used if present. This debug feature can be useful for testing specific plug-ins.

12.38.4.2 Plug-in name and file structure

In order to be recognized by AAE, all AAX plug-in bundles must use the ".aaxplugin" file name suffix. On OS X, the plug-in bundle must use this suffix while the binary itself does not require a suffix. On Windows, the plug-in binary (DLL) must use this suffix.

The directory structure of an AAX plug-in bundle is also important. See [.aaxplugin Directory Structure](#) in the [AAX Format Specification](#) document for more information.

12.38.4.3 Digital signature

As an added security measure against digital piracy, all AAX plug-in binaries must be digitally signed in order to run in Pro Tools. This signature step does not interfere with your existing copy protection and licensing solutions - it is simply a build step that you incorporate into your plug-in before releasing the binary.

Digital signatures are generated based on the plug-in binary and act as a guarantee against binary modification. Therefore, any build steps that modify the binary, such as symbol stripping, must be performed prior to signature generation. Digital signatures apply to the full .aaxplugin bundle, so any operation that modifies the contents of the bundle will invalidate its digital signature even if the operation does not affect the plug-in binary itself. Therefore, the generation and application of a digital signature should be the last step in any release plug-in build process.

Note

The digital signature requirement applies to Beta and Release software. This requirement does not apply to Development builds of Pro Tools or to other developer tools which can load unsigned binaries.

Requesting the digital signing toolkit

The AAX digital signatures required by Pro Tools are generated using digital signing tools licensed from PACE Anti-Piracy, Inc., which acts as the certificate authority for all AAX digital signatures. To request access to these tools, send an e-mail to devauth@avid.com with "Pace Tools Request" in the subject. Include the following information in your request:

- Company name
- Company mailing address

- Contact information for your AAX development lead or team
- iLok username which you will use for your digital signing admin account

Once your request has been approved you will be contacted by PACE with further instructions for acquiring and using the digital signature toolkit.

Signature requirements in Pro Tools 11

Host Compatibility Notes Pro Tools 11 requires PACE Eden digital signatures for AAX plug-ins.

Pro Tools 11 and higher use the Eden toolset. This toolset integrates fully with platform-specific signatures, so you only need to do one post-build step using the Eden digital signing tools to sign your plug-in with both the Eden signature and the relevant Apple GateKeeper or Microsoft Authenticode signature. For more information, see the Eden digital signature toolkit documentation.

Pro Tools 11 and higher will only accept the Eden signature; AAX plug-ins signed by earlier generations of PACE digital signing tools will not load in Pro Tools 11.

Signature requirements in Pro Tools 10

Host Compatibility Notes Pro Tools 10 requires either PACE DSig or Eden digital signatures for AAX plug-ins.

For compatibility with Pro Tools 10, AAX plug-ins must be digitally signed using PACEDSigTool.

Pro Tools 10.3.5 and above also support the new Eden signatures from PACE, so that universal binaries can be shipped which target both Pro Tools 11 or later and Pro Tools 10.3.5 or later.

Optional Signature for Pro Tools AAX DSP binaries

Host Compatibility Notes As of Pro Tools 10.2, support has been added to allow binary-level encryption of AAX DSP algorithms. Please note that this is *NOT* a requirement of AAX DSP plug-ins, and serves only as an additional security measure to protect an algorithm's DLL.

For more information about signing AAX plug-ins for use with Pro Tools 10, please contact PACE.

Automatic signature application by PACE tools

If you already protect your plug-ins using one of the anti-piracy technologies available from PACE then you may not need to perform any additional action:

- you are wrapping using PACE InterLok MasterMaker, your binaries will be automatically signed.
- If you are using Fusion Hybrid without wrapping with MasterMaker, please carefully review the "Adding digital signature checks" section of the Fusion Hybrid manual.
- If you are using PACE APIs (like PACE Interface or CDRM) without InterLok wrapping, please see either the latest PACEDSigTool read me or the Fusion Hybrid manual for additional details regarding digital signing.

12.38.5 Audio Engine Behavior and Features

Pro Tools hosts AAX plug-ins using the *Avid Audio Engine* (AAE). AAE implements all host-side AAX interfaces such as [AAX_IController](#) and the [AAX_ICollection](#).

12.38.5.1 Plug-in loading and AAE initialization

When Pro Tools launches, it immediately begins loading AAE. AAE searches the system for valid AAX plug-ins, checks each plug-in's digital signature, and loads, initializes and catalogs any valid plug-in modules that it happens to find.

This initialization is performed via the plug-in's `Describe` implementation; once AAE loads a plug-in binary, it calls the plug-in's `Describe` method to retrieve (and cache) the basic configuration of the plug-in. AAE then hands this information back to Pro Tools so that Pro Tools knows what plug-ins are available and what their basic properties are. Once a complete list of plug-in descriptions has been generated, AAE can construct any plug-in's individual modules and manage its algorithm.

12.38.5.1.1 Plug-in configuration cacheing

AAE is pretty smart, and it knows during initialization if anything has changed within the AAE Plug-Ins folder since the last time it was run. If nothing has changed, AAE relies on plug-in descriptions that it cached during the previous launch to speed through the plug-in loading process. If, however, any plug-ins have been added, removed, or updated since the last launch, AAE loads and re-caches the description for every installed plug-in.

Note

We recommend that you always enable the `AlwaysRebuildCache` DigiOption during plug-in development. See [Using DigiOptions](#) for more information.

12.38.5.2 Plug-in initialization

When a new plug-in instance is created in Pro Tools, AAE performs the following steps:

1. The plug-in's data model component is loaded
2. The default state of the plug-in is set (see [Default plug-in settings](#))
3. The plug-in's GUI and other host modules are loaded
4. The plug-in's algorithm private data state is initialized
5. The plug-in's algorithm is loaded and initialized
6. The plug-in's algorithm processing is initiated

12.38.5.3 Run-time processing behavior

Pro Tools 11 The audio engine in Pro Tools 11 includes some advanced real-time processing features that are not present in earlier versions of Pro Tools:

- When certain tracks with plug-ins have been silent for a period of time or Pro Tools is not in playback, those plug-in instances are automatically deactivated to reduce processing load on the host processor
- In certain situations such as playback or offline bounce where low latency is not required, Pro Tools may call AAX Native plug-ins with a larger buffer than normal.

This latter behavior is possible due to the fact that AAE uses two latency domains for plug-ins: a high-latency domain that operates over large block sizes and a low-latency domain that operates over small block sizes. Since processing at higher block sizes is generally more efficient, plug-in instances that are running in the high-latency domain generally consume less CPU cycles for their processing than instances that are running in the low-latency domain.

Pro Tools 11 may swap plug-in instances back and forth between these two domains at run-time and uses a set of rules designed to optimize the system's CPU resources while at the same time providing the best and most responsive user experience in every situation. These rules are different depending on whether the system is using an HDX card as its playback engine.

Here are some of the specific rules that are followed by the current versions of Pro Tools 11 at the time of this writing. These rules are subject to change from release to release:

- (*HDX and Native*) If there is any live audio or MIDI feeding into the plug-in's track and if the track is sending audio to an active output then all plug-in instances on the track will be run at low latency.

- (*HDX only*) If any AAX DSP plug-in instances are present in the signal path that feeds an AAX Native plug-in instance then the AAX Native plug-in will be run at low latency.
- (*HDX only*) Any AAX Native plug-in instance on an AUX track will be run at low latency.

Pro Tools 10

In Pro Tools 10, the audio engine is constantly running. Real-time plug-in instances are active from the moment they are created until they are removed from the session, regardless of audio playback, routing, or other run-time state changes. This design carries some implications for plug-in behavior in Pro Tools 10:

- When Pro Tools is not in playback, plug-ins are sent a constant stream of "silence" via zeroed audio buffers
- Plug-ins must process these buffers and may generate output data into them at any time
- Processing "tails" (e.g. reverb or delay) should be applied at all times, even after playback has stopped

Note that, during looped playback, processing from the end of the loop carries seamlessly to the beginning of the loop - the plug-in's state is not reset at the loop point.

12.38.5.4 New to Pro Tools 11

In addition to the real-time processing changes mentioned above, several other relevant features were added to the audio engine in Pro Tools 11.

For a full list of compatibility and feature differences between different AAX plug-in hosts, see [Host Support](#).

12.38.5.4.1 Deterministic Plug-in Automation

Starting in Pro Tools 11, Native and DSP plug-ins will receive automation changes in a deterministic manner. Each time the transport is played, automation events will be delivered to the plug-in for processing at the same moment on the timeline. Note that this does not mean automation is sample-accurate with respect to where the automation breakpoints are placed in the timeline, but rather that the timing will be the same between transport runs.

12.38.5.4.2 Deprecated and Related Plug-in Lists

To help ease the transition for users to 64-bit, Pro Tools 11 includes support for deprecated and related plug-in types. This allows you to associate any legacy plug-ins with new AAX plug-in types so that Pro Tools can automatically convert sessions with older plug-ins to the new types.

12.38.5.4.3 Offline Bounce

Pro Tools now supports faster-than-real-time offline bounce for all sessions. All plug-ins with AAX Native types are supported. For AAX DSP plugins, the offline bounce process will temporarily convert those to their corresponding AAX-Native types to complete the bounce. Because offline bounce is faster-than-real-time, audio processing callbacks will occur as fast as the algorithm will allow for. For this and other reasons, your algorithms should never depend on wall-clock time for features such as LFO, delay time, etc. Instead, all algorithms should always base time calculations on sample time so that the output will still be correct even if the algorithm is being called from an offline bounce.

12.38.5.4.4 AAX Hybrid Plug-ins

Pro Tools 11 also supports [AAX Hybrid](#) plug-ins, which can have both a Native and a DSP algorithm processing component. Audio-rate data can also be shared between the two processing components, which allows you to split your algorithm up into low-latency and high-latency contexts for better efficiency and to enable plug-ins such as convolution reverbs, spectrum analyzers, and other similar architectures.

12.38.6 Basic plug-in operation

12.38.6.1 Configuration management

Each Effect in an AAX plug-in may contain multiple configurations. Pro Tools automatically determines the appropriate plug-in configuration for each Effect insert at run time based on the insert's required sample rate, channel width, and processing mode (Native or DSP.) Under some circumstances, the configuration requirements for an Effect insert may change at run-time. Here are some examples:

- When a width-changing plug-in (e.g. mono-to-stereo) is instantiated on a track then all of the following inserts must be converted to the new stem format
- When a user imports session data between sessions at different sample rates then all of the imported plug-ins must be converted from the old sample rate to the new sample rate
- When a user opens a session that contains deprecated effects, they must be replaced by the corresponding installed effects

Whenever a new configuration is required, Pro Tools automatically determines whether one is available that meets the new requirements and, if it is, swaps in a new plug-in instance using a copy of the previous configuration's settings.

In order for Pro Tools to deterministically select the appropriate Effect configuration to load in any given scenario, each of the configurations that are registered in the Effect must be described with distinct and mutually exclusive compatibility requirements.

12.38.6.2 Plug-in activation and deactivation

In Pro Tools, real-time plug-in inserts can be either active or inactive. Inactive plug-ins are not instantiated and are entirely removed from the processing chain, though they are still saved with the session and maintain a placeholder in their track's insert list for easy activation at a later point.

Active plug-ins may be de-activated manually by the user or automatically by Pro Tools. Plug-ins may be loaded as inactive when a plug-in that has been saved in a session has been uninstalled and is no longer available, when a required plug-in configuration is not available, or at any other time when a particular plug-in instance cannot be loaded.

12.38.6.3 Plug-in bypass

AAX plug-ins must implement a Master Bypass parameter, which is controlled via the "Bypass" button in the Pro Tools plug-in window header. While bypassed, the plug-in must not apply any processing to the audio that is passed to it (except delay, see below.) The plug-in may choose to smoothly transition into and out of bypass however it chooses.

Any algorithmic delay that a plug-in incurs during normal operation must be maintained by the plug-in during bypass. For more information about this requirement, see [Automatic Delay Compensation](#).

12.38.6.4 Presets and settings management

Pro Tools includes a plug-in preset management system that can be accessed from the plug-in window header. With this system, users can save plug-in settings to disk and load the settings later to restore the plug-in's configuration.

Preset files can be bundled with an AAX plug-in to demonstrate a variety of uses for the plug-in or as recommended settings for different situations, and, as a plug-in developer, you are encouraged to provide a large selection of pre-configured presets along with your AAX plug-ins. See [Check and finalize page tables](#) for more information about bundling presets with your plug-in.

Aside from user preset management, there are many cases when the state of a plug-in must be captured or restored by AAE. For example, AAE must restore plug-in settings when a session is loaded and when converting a plug-in between different configurations.

12.38.6.4.1 The plug-in preset menu

Plug-in presets are available to the user via the Plug-In Settings menu in the Pro Tools plug-in window header. This menu supports nesting presets into sub-folders, which provides a convenient way to categorize and organize large sets of presets. In addition, users may save custom presets and add these custom presets to the menu.

Figure 2: The Plug-In Settings menu in the Pro Tools plug-in window header

The preset menu in the Pro Tools plug-in header is built from the following two directories:

- *Session file*/../Plug-In Settings
- *User Library root*/Plug-In Settings

The default setting for the User Library root directory is ~/Documents/Pro Tools on OS X, but the user can change this setting in the Pro Tools preferences.

12.38.6.4.2 Factory presets

Starting in Pro Tools 10.3.6, Pro Tools supports automatic installation of plug-in presets. AAX plug-ins should include a set of presets in the following directory within the .aaxplugin:

- *MyPlugIn.aaxplugin*/Contents/Factory Presets/*MyPlugInPackage*/

Where *MyPlugInPackage* is the plug-in's longest [Package Name](#) with 16 characters or fewer.

On Pro Tools launch, all installed AAX plug-in settings are copied from the .aaxplugin bundles' "Factory Presets" folders into the User Library directory (see [above](#).)

Note

Since the User Library root directory is a customizable setting, you should never install presets directly onto a user's system. If you require a central repository of settings on the system that is under control of your installer then you should handle these settings as external resources. You can use custom settings chunks in the plug-in's "Factory Presets" .tfx files to redirect your plug-in to read the appropriate installed resources.

12.38.6.4.3 Default plug-in settings

When the first instance of an effect is made active in a session, Pro Tools queries the instance's state and stores this data as the effect's "factory default" preset. This preset is cached by Pro Tools and will be set on each subsequent instance of the plug-in with the same configuration.

The plug-in's factory default settings are stored on disk in a temporary file location that is specific to the user. Pro Tools looks for the factory default settings file for a plug-in each time an instance of the plug-in goes from an inactive state to an active state, including when the instance is first created. If there is no factory default settings file on disk then Pro Tools will create it using the plug-in's current settings.

All factory default settings files are deleted during the Pro Tools shutdown procedure. Therefore, under normal operation, these files will be refreshed with each launch of Pro Tools.

Note

If the Pro Tools shutdown procedure is not completing, for example if you regularly terminate Pro Tools from a debugger, then the plug-in factory default settings files will not be deleted automatically.

When a session is loaded Pro Tools will perform the following steps on each plug-in instance:

1. Instantiate the plug-in and create a [AAX_IEffectParameters](#) object
2. Look for the cached factory default settings file in the file system
3. If the factory default settings file is not found, query the plug-in for its current settings and create the factory default settings file using these settings

4. Set the instance's default settings based on the settings stored in the cached factory default file
5. Send the instance a [AAX_eNotificationEvent_SessionBeingOpened](#) notification
6. Set the saved settings from the session

12.38.6.4.4 The Compare Light

The plug-in window header in Pro Tools includes a "Compare" button, the Compare Light control. This control allows the user to compare the current state of the plug-in with the last preset that was loaded, or the plug-in's default settings if no other preset has yet been loaded.

Pro Tools polls each displayed plug-in periodically to determine whether or not its state matches the currently loaded preset. While the state matches, the Compare Light is inactive and unlit. As soon as the plug-in's state differs from the preset, the Compare Light becomes active and is highlighted.

When the Compare Light is active, the user may click on it to cache the current plug-in settings and temporarily swap in the last preset that was loaded. Clicking on the Compare Light a second time will restore the cached plug-in settings.

The specific operation of the Compare Light is determined by the plug-in's implementation of [AAX_IEffectParameters](#). To determine the correct state for a plug-in's compare light, Pro Tools makes regular calls to [AAX_IEffectParameters::GetNumberOfChanges\(\)](#) from a callback timer. If this method's aValueP parameter has changed since the last time the plug-in was queried then Pro Tools proceeds to call [CompareActiveChunk\(\)](#). If [CompareActiveChunk\(\)](#) returns with isEqual==false then the Compare Light will be lit, otherwise the light will be dimmed.

12.38.6.4.5 Basic chunk handling

All of these situations use the same basic settings management infrastructure in Pro Tools, which uses the "chunk" API of [AAX_IEffectParameters](#) to retrieve arbitrary blocks of data from the plug-in (to retrieve a preset) and send the same block back to the plug-in (to set a preset.)

When retrieving a preset from a plug-in, Pro Tools first asks for the size of the plug-in's settings chunk(s). Pro Tools then provides the plug-in with a pre-allocated buffer of memory into which the plug-in may store its settings information using any format that it chooses.

When Pro Tools needs to restore the plug-in to this preset state, it sends a copy of this data back to the plug-in. The plug-in must interpret this data and set its internal state to match the preset.

12.38.6.5 Modifier key behavior

In order for users to have a consistent experience, all AAX plug-ins should provide standard modifier key behaviors as described in this section. These operations are demonstrated by all Avid plug-ins in Pro Tools, and you can experiment with Avid's AAX plug-ins to demonstrate the correct plug-in modifier key behavior.

The following modifier key combinations must be handled explicitly by the plug-in:

OS X Keys	Windows Keys	Expected Behavior
Command-click	Control-click	Adjust the parameter's value with fine control, for continuous controller widgets
Option-click	Alt-click	Return the parameter's value to default*
Shift-click	Shift-click	Link parameters across all channels, if applicable

*Set-to-default may also be handled directly by the host, depending on the host version (see below).

In addition to these events, there are also specific behaviors which Pro Tools and other AAX hosts provide for certain key combinations in plug-in GUIs. For example, Pro Tools provides the following modifier key behavior overrides:

OS X Keys	Windows Keys	Expected Behavior
Command-Control-click	Control-Start-click	Show parameter automation lane in the Pro Tools Edit Window, if automation is enabled for the parameter
Command-Right click	Control-Right click	
Command-Option-Control-click	Control-Alt-Start-click	Activate pop-up menu for automation
Command-Option-Right click	Control-Alt-Right click	

Other AAX plug-in hosts implement different host-managed behavior for modifier key combinations, and additional host-managed key combinations may be added to any AAX host in the future. For example, Pro Tools adds host-managed support for setting plug-in parameters to their default values beginning in Pro Tools 12.0.1.

In order to allow the AAX host to handle these operations, a plug-in must always call the handler methods in [AAX_IViewContainer](#) before handling any mouse events in its own GUI. It is important to call these methods for *all* mouse events, in case additional handlers are added to future versions of the host or the plug-in is run in a new AAX host with a different set of handled modifier key combinations.

See the [AAX_IViewContainer](#) class documentation for more information about passing mouse events to the AAX host.

12.38.7 Optional plug-in features

Pro Tools plug-ins offer users a rich set of integrated features. To make sure your plug-ins integrate into users' expected Pro Tools workflows, where applicable you should implement all of the features presented in this chapter.

For more information about any of these features in Pro Tools, see the latest Pro Tools Reference Guide.

12.38.7.1 Audio management features

12.38.7.1.1 Sidechain input

If applicable, plug-ins may choose to enable [sidechain inputs](#). If a sidechain is enabled, a menu is added to the plug-in's header that allows the user to choose an interface or bus as the sidechain, or "key input". For AudioSuite, the user can only use an existing audio track as the sidechain input. Once enabled, the plug-in will be able to access sidechain input just like any other input signal.

12.38.7.1.2 Auxiliary Output Stems

Pro Tools has the capability to show and route multiple "auxiliary" outputs from a plug-in to other tracks. These are known as [Auxiliary Output Stems](#) (AOS), a stem referring to one set of outputs. A stereo stem contains two outputs, left and right, and a mono stem contains one output. The outputs will appear in the input assignment pop-up menu of each track under the category "plug-in".

Some notes regarding this feature:

- Only mono and stereo stems are available as auxiliary outputs.
- The aux outputs cannot be added and removed from the system dynamically though they can be made inactive by the user. The total number of aux outputs, stem types, names, paths, and ordering are defined only once by the plug-in.
- Plug-in aux outputs are not available from the sidechain input popup menu in other plug-ins. Users will not see the "plug-in" submenu when clicking on a plug-in sidechain popup.
- There cannot be any multi-mono multi-output plug-ins. If a mono plug-in instance offers multiple outputs it cannot support multi-mono.

If a plug-in is going to utilize the AOS feature, it will be responsible for a few details that are summarized below:

- Aux Output Paths

The plug-in is responsible for the definition of valid aux output paths. This definition includes the total number of outputs, the desired order of stereo and mono paths. Pro Tools will query each plug-in for available valid paths and populate its track input selector popup menus accordingly.

- Aux Output Path Order

The plug-in is responsible for specifying the type and name of each of its aux output paths. A plug-in decides whether the aux outputs are all stereo, all mono, "X" stereo outputs followed by "Y" mono outputs, or some other combination. Pro Tools lists each output in the order given by the plug-in. If mono and stereo paths are interleaved the input popup menu of the mono tracks keeps that order and breaks the stereo paths into their respective left and right sides using ".L" and ".R" suffixes.

- Aux Output Names

A plug-in is responsible for giving meaningful names to aux outputs. Names are only defined once, so they will stick. At the very least, individual outputs should be labeled "Output xx", where "xx" is the aux output number as it is defined in the plug-in. The output name should also include the words "mono" and "stereo" to support when users are looking for an output with a specific stem format.

- Aux Output Numbering

The plug-in is responsible for defining the lowest available aux output number. Plug-ins should base this number on the width of the plug-in's main outputs. For example, when using a stereo instance of a sampler the first aux output should be #3, when using a 5.1 instance of the sampler the first aux output should be #7, etc. This is to keep the numbering scheme inside of the plug-in and in Pro Tools consistent. From the perspective of Pro Tools, plug-ins typically enumerate all available outputs and do not differentiate between main and aux outputs. The first "N" outputs are used for the main outputs, and all the remaining outputs are available for aux output paths.

- Separate Multi-Output Plug-in Process Type

Plug-in developers are encouraged to offer both "regular" and "multi-output" versions/types of any multi-output capable plug-in. We strongly suggest this to conserve resources and to keep the user's workspace as uncluttered as possible. Users can choose to use the regular version/type for plug-ins they don't need aux outputs for. Multi-output versions can be created as separate process types so that there need not be separate binaries. Such additional process types will be listed in the plug-in menu next to their regular version siblings. They should be nominally distinguished by appending phrases like "multi-output" to the plug-in name, for example.

Note

When moving sessions between different PT systems, multi-output process types will NOT be automatically converted to regular process types if multi-output types are not available.

- No Multi-Mono Implementations

A plug-in is responsible for not having multi-mono enabled if it utilizes auxiliary outputs stems. Auxiliary output stems will not work in multi-mono enabled plug-ins. Multi-mono is automatically disabled for AOS in the Effect Layer.

12.38.7.1.3 External metering and internal clip

Pro Tools may use the meter values reported by a plug-in for display on attached control surfaces and other external plug-in views. In general, the behavior of a plug-in meter on these devices will depend on the meter's properties as registered in Describe. The meter behavior may also depend on the plug-in's registered category. See [Plug-in meters](#) for more information about how to register your plug-in's meters.

- Gain reduction metering

Pro Tools versions that support gain reduction metering will display an inverted gain reduction meter next to each plug-in insert and also next to the track's main meter in the Pro Tools Mix and Edit windows.

All registered plug-in gain reduction meters are used by the Pro Tools gain-reduction metering UI. The plug-in gain reduction meters in the Pro Tools Mix and Edit windows will combine metering data for all gain-reduction meters of the same type ([Compressor/Limiter](#) or [Expander/Gate](#)) in the plug-in:

- For plug-ins with multiple gain-reduction meters of the same type, the minimum (most gain-reduced) meter value for the current buffer will be used
- For multi-mono plug-ins, the minimum meter value across all of the per-channel mono instances will be used

Pro Tools can be set up to display [Compressor/Limiter](#), [Expander/Gate](#), or both types of metering data in these displays via Preferences > Metering > Display > Gain Reduction Meter Type. If both types are used, the displayed meter level is simply the sum of the selected values for each type.

The track gain-reduction meter displays the sum of all the track's inserts' gain reductions, using the same rules as above.

- **Plug-in internal clipping**

Pro Tools uses a plug-in's reported meter values to determine whether the plug-in may have clipped internally. Any meter that reports a value greater than 1.0 is considered to be clipped, and Pro Tools will report that the plug-in has clipped internally if any of its registered meters have clipped. This clip state is held for a length of time based on a user preference in Pro Tools.

The plug-in header has a clip light that indicates that the plug-in has clipped somewhere internally. Additionally, plug-ins that have clipped internally will appear in red on the insert, even if the plug-in window is not open. This allows users to see at a glance where clipping has occurred in their mix. Currently, plug-in clip indicators are available in Pro Tools HD software only.

Host Compatibility Notes In Pro Tools 11 and above, the Avid Audio Engine (AAE) no longer automatically generates clipping indication for plug-ins. This is because the new engine can operate properly well beyond a unity sample value of 1.0. Thus, it is up to the plug-in itself to set and clear its clip indicators as needed.

12.38.7.1.4 Automatic Delay Compensation

Automatic Delay Compensation maintains time-alignment between tracks that have plug-ins with differing algorithmic delays, tracks with different mixing paths, tracks that are split off and recombined within the mixer, and tracks with hardware inserts. To maintain time alignment, Pro Tools adds the exact amount of delay to each track necessary to make that particular track's delay equal to the delay of the track that has the longest delay.

In order to be compensated correctly, AAX plug-ins must report any algorithmic delay that they incur. This delay may be reported in the plug-in's description, and may also be changed at run-time.

Automatic Delay Compensation Notes

- Currently, Pro Tools will not update its delay compensation settings while Pro Tools is in playback, so a plug-in that dynamically changes its delay settings at run-time should either prevent any algorithmic delay updates during playback or give a visual indication to the user when the delay that it applies and the delay that Pro Tools is compensating for different delay settings.
- Pro Tools does not update delay compensation settings when plug-ins go into and out of bypass, and does not automatically maintain bypass audio buffers for delayed plug-ins. It is therefore required that all plug-ins incur the same amount of delay when bypassed as during normal operation.
- Automatic Delay Compensation is not applied during offline (AudioSuite) processing for plug-ins which use the [Host Processor](#) interface. If your Host Processor plug-in incurs algorithmic delay then you must incorporate audio lookahead via the Host Processor interface's random timeline access API.
- Given the many routing possibilities in Pro Tools, the Automatic Delay Compensation feature involves some subtleties that may not be immediately apparent or intuitive. For more information about this feature, we strongly recommend that you review the relevant chapters in the Pro Tools Reference Guide.

12.38.7.2 Plug-in categories

The plug-in menus in Pro Tools are hierarchical and by default are organized by category. These general categories represent common plug-in functions like EQ, dynamics, reverb, etc. Plug-ins may report one or more of these

categories in order to be placed into the proper menu. For a complete list of plug-in categories available, refer to the [AAX_EPlugInCategory](#) enum.

Some features, such as control surface center-section mappings, are only available to plug-ins that report a particular category, so it is important for plug-ins to report the correct set of categories.

12.38.7.3 Advanced non-real-time processing

AudioSuite processing allows AAX plug-in to operate on audio in a non-real-time manner. AudioSuite plug-ins will appear in the AudioSuite menu in Pro Tools. By default, any AAX-Native plug-in will appear in the menu as long as an [AAX_eProperty_PlugInID_AudioSuite](#) property is defined alongside the corresponding [AAX_eProperty_PlugInID_Native](#) ID. However, to make use of extended AudioSuite features such as non-real-time sample access, the Analysis pass, a separate entry method subclassed from the [AAX_CHostProcessor](#) implementation in the SDK should be used.

12.38.7.3.1 AudioSuite processing modes

Pro Tools includes a number of different AudioSuite processing modes, each of which changes the precise behavior of an AudioSuite processing event.

Output modes

- *Overwrite files* Output audio destructively overwrites the selected audio files on disk
- *Create individual files* Individual new files are created for each processed clip
- *Create continuous file* A single new file is created with data from the full processing pass

Input modes

- *Clip by clip*
- *Entire selection*

The plug-in may optionally disable the "clip by clip" processing mode if continuous input data is required, by using the property [AAX_eProperty_ContinuousOnly](#).

Channel modes

- *Mono mode* Each selected channel is processed as an individual mono audio stream
- *Multi-input mode* Selected channels are sent to the plug-in in multi-channel streams

Multi-input mode is only valid with the "entire selection" input mode, since the "clip by clip" input mode requires that each clip be processed individually as a standalone audio channel.

The plug-in may optionally disable "mono mode" processing if its algorithm is only valid for multi-channel input, by defining the [AAX_eProperty_MultiInputModeOnly](#) property.

12.38.7.3.2 Analysis

AudioSuite plug-ins support an optional Analysis pass, which allows a plug-in to access the incoming audio before the actual Render pass starts. When Analysis is defined with either [AAX_eProperty_OptionalAnalysis](#) or [AAX_eProperty_RequiresAnalysis](#), an Analyze button will appear in the plug-in footer in the GUI.

An analysis pass is useful for collecting pitch, spectrum, loudness, noise threshold, or other data that will help the user set up parameters based on the audio content being processed.

12.38.7.3.3 Reverse

A "reverse" feature is available for [Reverb](#) and [Delay](#) AudioSuite plug-ins. This effect will reverse the source audio, apply the AudioSuite plug-in processing, and re-reverse the source audio back to its original orientation, thereby applying the AudioSuite effect in reverse.

Reverse replaces the Analysis pass in the Pro Tools UI, so AudioSuite plug-ins in these categories do not receive a user-triggered analysis pass.

12.38.7.3.4 Random-access and non-linear processing

The generation of output samples by an AudioSuite Process must occur linearly and incrementally; however, the source of input samples may optionally be randomly accessed from the entire selected track. This enables many advanced processing capabilities such as whole-file analysis, audio reverse effects, and timeline-level modifications such as expanding, contracting, or shifting the processed region.

In order to prevent invalid data from being randomly accessed, the "overwrite files" processing mode is disabled for plug-ins that use random-access processing.

12.38.7.3.5 AudioSuite Handles

By default, when processing audio segments with an AudioSuite plug-in, Pro Tools will also process an extra region before and after the selected audio. These extra regions will be trimmed out of the selected.

The reason for this feature is so that the user has some room to expand the resulting audio clip (for fades or other reasons). However, certain AudioSuite plug-ins will operate more intuitively if these handles are not processed (such as delay, reverb, loudness normalization, and other plug-in types). To disable extended handle processing, set the [AAX_eProperty_DisableHandles](#) property to true.

12.38.7.3.6 Extended features

AAX-AudioSuite plug-ins also have several other optional features including custom progress dialogs, reverse mode, and side-chains. For a complete reference of supported AudioSuite-related properties, refer to the properties between [AAX_eProperty_AudiosuitePropsBase](#) and [AAX_eProperty_MaxASProp](#) found in [Interfaces\AAX_Properties.h](#).

12.38.8 Debugging AAX plug-ins

12.38.8.1 Debugging within Pro Tools

Shipping versions of Pro Tools do not support attaching a debugger. This is to prevent malicious users from compromising Pro Tools security as well as the security of third-party plug-ins.

As an AAX plug-in developer, you are granted access to debuggable "developer build" versions of Pro Tools to help your development efforts. Some Pro Tools developer builds are feature-limited; for example, developer builds of Pro Tools 10 do not allow saving or exporting sessions.

The easiest way to debug plug-ins within Pro Tools is to start up Pro Tools, open a session, attach your debugger, and then instantiate your plug-in. This order seems to work the best for most users. If you need to debug the initial host query of your plug-in at Pro Tools start, it is possible to launch Pro Tools from within your debugger. However, this method is sometimes known to cause problems with certain debuggers.

AAX plug-in developers are also able to download pre-release and beta versions of upcoming Pro Tools releases. For now, these pre-release versions are not debuggable, but that is expected to change in the future as we work to make a unified debuggable pre-release installer available for upcoming Pro Tools versions.

Both debuggable and pre-release versions are available for download as part of the AAX SDK Toolkit on the [My Toolkits and Downloads](#) page at [avid.com](#).

12.38.8.2 DigiShell

DigiShell is a software tool that provides a general framework for running tests on Avid audio hardware. As a command-line application, DigiShell may be driven as part of a standard, automated test suite for maximum test coverage. The latest DigiShell tools may be downloaded as part of the AAX SDK Toolkit on the [My Toolkits and Downloads](#) page at [avid.com](#).

More information about DSH in general and about loading and testing plug-ins in DSH can be found in [DSH Guide](#).

12.38.8.3 DigiTrace

All Avid AAX hosts provide tracing functionality based on Avid's DigiTrace tool. DigiTrace is a library that provides high-performance logging and tracing capabilities to Pro Tools and its components, including plug-ins. More information about DigiTrace can be found on Avid's audio developer website.

To enable trace logging for AAX plug-ins, use the `AAX_TRACE` macro defined in [AAX_ASSERT.h](#). A separate macro, `AAX_ASSERT`, is also available to signal run-time errors. These macros are both cross-platform and will function whether the algorithm is running on the TI or on the host.

For more information about DigiTrace, see [DigiTrace Guide](#).

12.38.8.3.1 Tracing requirements

The `AAX_ASSERT` and `AAX_TRACE` macros are debug-only and will not provide tracing output from release builds of your plug-in. `AAX_TRACE_RELEASE` may be used for tracing in both debug and release configurations. These macros require that the `DTF_AAXPLUGINS` facility to your DigiTrace configuration file. You can toggle this facility to enable or disable AAX algorithm-level tracing. For details on setting up tracing on AAX TI plug-ins, please refer to the [TI Guide](#).

12.38.9 Troubleshooting common AAX plug-in failures

Pro Tools presents a "Move Unauthorized Plug-ins" dialog after attempting to launch with my plug-ins installed, and the plug-ins do not appear in the Pro Tools insert menus

- This error indicates that Pro Tools was not able to load the plug-in binary for some reason. The error indicates a copy protection failure since that is by far the most common reason for users to encounter this kind of error in released plug-ins, but any other error that prevents the plug-in DLL from loading in Pro Tools may also cause this error message.

This error does *not* indicate a failure when checking the plug-in's digital signature. A digital signature failure would generate a different error message that would specifically mention the plug-in's signature.

The `DTF_AAXHOST` [DigiTrace](#) facility provides additional information about AAX plug-in load errors.

One common cause of DLL loading failures is a failure to dynamically link to other required libraries. In this case, the `DTF_AAXHOST` tracing will indicate something like the following:

— `AAXH_CEffectFactory::ParseDLL - exception thrown(The specified module could not be found. (126) while`

This exception indicates that some DLL upon which the plug-in depends is not present in the system. This is most commonly due to dynamic linking to CRT libs, but it could also be caused by any other DLL dependency.

12.38.10 Using DigiOptions

DigiOptions provide a way to override the standard Pro Tools configuration. These options are designed to assist with testing and development of Pro Tools, and they are often useful during plug-in development as well.

To configure DigiOptions, place a plain-text file named `DigiOptionsFile.txt` next to the Pro Tools application. On OS X, place this file next to the `Pro Tools.app` bundle and on Windows place it next to the `Pro Tools` executable.

In recent versions of Pro Tools, a red notice will appear in the Pro Tools splash screen and in the application About Box indicating when DigiOptions are enabled in the build.

Figure 3: DigiOption activation notice in the Pro Tools splash screen.

Note

If suffixes are hidden on your OS then be careful that you do not accidentally give the file an incorrect name such as `DigiOptionsFile.txt.txt`.

12.38.10.1 Useful DigiOptions

Note

Support for these options may vary from release to release

- `AlwaysRebuildCache 1`

This option forces Pro Tools to re-load all installed plug-ins each time the application is launched. This can be very useful during development, since some plug-in updates during development will not result in an updated plug-in binary or an updated modification date for the top-level .aaxplugin bundle.

Note

If you have changed your plug-in's ID values during development and if you encounter AAE error -20038 when your plug-in is loaded then Pro Tools may be using a cache of the outdated plug-in ID. Use the AlwaysRebuildCache DigiOption or launch Pro Tools once without your plug-in installed to clear this state.

- `NeverUnloadPlugInBundles <int>`

Enable plug-in bundle unloading. The default behavior in Pro Tools 11 and Pro Tools 12 is for this option to be set to 1. In a future release of Pro Tools 12, the default for this option will be changed to 0. Therefore it is very important to test your plug-ins and make sure that they operate correctly with the option set to 0.

Host Compatibility Notes

Supported in Pro Tools 12 and higher.

- `LogInterruptDataEveryNSeconds <int>`
`LogInterruptDataEveryNSeconds_HL <int>`

These options enable regular DigiTrace performance logging from the low-latency and high-latency real-time audio render threads in the Avid Audio Engine. For example, `LogInterruptDataEveryNSeconds_HL 2` would trigger a performance log for the high-latency render thread every two seconds. For more information about performance logging in AAE see [Real-time AAE performance logging with DigiTrace](#).

- `PauseDuringLaunchToAttachDebugger 1` (*Starting in Pro Tools 11*)
`AssertOnLoadPlugins 1` (*Pro Tools 10 and earlier*)

This option will trigger a dialog and pause execution just before Pro Tools begins loading plug-ins. Due to Pro Tools copy protection it may be impossible to launch Pro Tools Developer Builds directly from a debugger, and, when this is the case, this dialog provides a good point at which to attach a debugger to Pro Tools.

- `DisplayHostPlugInLatency 1`

Display information about the plug-in's processing domain and dynamic processing status in the Pro Tools plug-in window header.

Figure 4: DisplayHostPlugInLatency info in plug-in header.

- `TestGetCurveData <0, 1, 2>`

Display the plug-in's curve data as a plot in the Pro Tools plug-in window header. Possible values are:

1. Off
2. [EQ curve](#)
3. [Gain reduction curve](#)

Warning

The implementations of this test curve and the actual curve data shown on Avid control surfaces are different. In particular, the display in Pro Tools is updated regularly at idle time, whereas the curve on a control surface is only updated when certain parameter changes occur (see [PTSW-195316 / P-T-218485](#)). The graph point interpolation, range, and sample points are also not exactly equivalent to the values used on an actual control surface, so you should not expect the curve shown in the debug view in Pro Tools to exactly match what would appear to users. After performing early prototyping of your curve data using the TestGetCurveData DigiOption you are strongly encouraged to use either the S6 Surfulator software or the [Pro Tools | Control](#) app to test and refine the plug-in's curve data in a real-world environment.

For more information about graph curves see [EQ and Dynamics Curve Displays](#).

Host Compatibility Notes Supported in Pro Tools 12 and higher.

Figure 5: TestGetCurveData EQ graph in plug-in header.

- RenderMissingFilesAsBlank 1

Beginning in Pro Tools 10, this option may be used to automatically render test tones into audio clips with missing source media. The test tones are rendered with different frequencies and magnitudes. This option can be useful when troubleshooting using a session file provided by an end user, since session-specific issues are rarely dependent on the source media, but may depend on there being some signal present in the session. This option requires that the Avid Signal Generator plug-in is installed.

- 44100_Rate <int>
48000_Rate <int>

Set the new base sample rate for each set of sample rate multiples. Can be useful for simulating sample rate pull-up by up to +5% (e.g. 44100_Rate 45000 in DigiOptions.txt.) The 44100_Rate option affects 44100, 88200, and 176400 Hz rates, while the 48000_Rate option affects 48000, 96000, and 192000 Hz rates.

- EnablePlugInHotSwap 1

Note

This option is currently non-functional for AAX plug-ins Pro Tools. See [PTSW-188653 / PT-218451](#)

This option will allow Pro Tools to recognize changes to your plug-in during run-time. This allows you to re-compile and load your updated plug-in without re-launching Pro Tools. The following conditions must be true in order to enable hot-swapping between versions of your plug-in:

- There cannot be any instances of the plug-in currently in Pro Tools.
- Both EnablePlugInHotSwap 1 and AlwaysRebuildCache 1 must be set.
- WinDLErrorMode <int>

Set the Windows error mode during DLL loading and unloading. The value of this option will be set as the uMode for a call to the SetErrorMode Windows API during DLL loading and unloading.

This option can be useful when debugging plug-in load errors on Windows, for example errors that cause a "The following Plug-Ins failed to load because no valid authorization could be found" dialog to appear during Pro Tools launch.

Host Compatibility Notes Supported in Pro Tools 12 and higher.

- DisableCMNAssert 1

Disable assert dialogs in Pro Tools. Use this option if your Pro Tools debugging sessions are being interrupted or terminated due to assert failures in the app.

Note that Pro Tools asserts may be triggered by your plug-ins; you should always investigate any assert that you see to determine whether it is being caused by a plug-in. If you need information about any Pro Tools asserts, post a question here on the [AAX](#) developer forums or write to us at devservices@avid.com and we will be happy to help.

- TestPlugInDescriptions 0

Use this DigiOption to toggle the plug-in description validation check in Pro Tools developer builds. Developer builds beginning in Pro Tools 12.8.2 will check plug-in description information when the plug-in is loaded and will present a warning dialog if the check fails. See [Describe Validation](#) section in the [Description callback](#) page for more information.

Host Compatibility Notes Supported in Pro Tools 12.6 and higher.

- `RealTimeDenormalsAreZero <int>`

Use this DigiOption to toggle the default denormal handling policy of the AAE real-time processing threads. A value of 1 means that DAZ+FZ will be enabled for all AAX real-time processing threads unless explicitly changed using thread-specific primitives, while a value of 0 means that DAZ+FZ will be disabled unless explicitly changed.

The default state of the DAZ+FZ flags for AAE real-time processing threads varies depending on the AAE version and on the OS: On Windows, DAZ+FZ is turned on (i.e. denormals are set to zero) in the AAE builds used in Pro Tools 11 and higher, and in the corresponding AAE builds used in Media Composer and VENUE. On Mac, DAZ+FZ is turned off by default in the AAE builds used in all releases prior to Pro Tools 12.7. Starting in Pro Tools 12.7, DAZ+FZ is turned on by default for both Mac and Windows.

Host Compatibility Notes Supported in Pro Tools 12.8.2 and higher.

12.38.11 Compatibility Notes

See [Host Support](#) and [Known Issues in Pro Tools](#) for additional details regarding Pro Tools compatibility

12.38.11.1 Changing parameter names

Name changes for automatable plug-in parameters are only supported starting in Pro Tools 11.1. This applies both to changing a parameter Name at runtime with [AAX_IParameter::SetName\(\)](#) and to changing a parameter Name between different versions of the plug-in.

Versions of Pro Tools prior to Pro Tools 11.1 will not be able to load automation data for a parameter whose Name has been changed. If any of your plug-in's automatable parameters' Names change, you must document that the plug-in is not supported in any Pro Tools versions prior to Pro Tools 11.1.

Specifically, automation data for a parameter will be lost if the following conditions occur:

- The session was saved using the parameter Name A.
- The default Name for the parameter is not A when the session is opened.
- Either of the following:
 - The session is being opened in a version of Pro Tools prior to Pro Tools 11.1
 - The session was last saved in a version of Pro Tools prior to Pro Tools 11.1 and the parameter's ID is not A

In addition, any parameter whose Name has changed will not appear as automatable (its automation lanes will disappear and it will not appear in the plug-in's automation list) if a session last saved in Pro Tools 11.1 is opened in an earlier version of Pro Tools. In this situation the automation data for the parameter may still be available, and may be restored if the session is later re-opened in Pro Tools 11.1.

Collaboration diagram for Pro Tools Guide:



12.39 Media Composer Guide

Details about using AAX plug-ins in Media Composer.

12.39.1 Contents

- [About this document](#)
- [Processing modes](#)
- [Compatibility requirements](#)
- [AAX feature support in Media Composer](#)
- [Additional Information](#)

12.39.2 About this document

This guide discusses specific details related to using AAX plug-ins with Media Composer, such as loading and initialization procedures, GUI hosting, and other application-specific features.

For more information about the features, functionality, and use of Media Composer see the Media Composer user documentation.

12.39.3 Processing modes

Media Composer supports AAX plug-ins in two processing modes: AudioSuite and AAX Native

- AudioSuite plug-ins perform non-real-time, random-access, file-based processing entirely on the host CPU.
- AAX Native plug-ins perform real-time, linear, non-destructive processing entirely on the host CPU.

AAX plug-in processing in Media Composer is managed by specific Tools. Each of these Tools can be accessed using the "Tools" menu in the Media Composer application.

12.39.3.1 Non-real-time processing (AudioSuite)

Use the AudioSuite Tool to perform AudioSuite processing in Media Composer. The AudioSuite Tool applies an effect to a clip in the timeline of the record monitor.

Specific AudioSuite plug-ins appear in the Plug-In Selection menu in the AudioSuite window.

Note

Unlike Pro Tools, the effect to clip relationship is remembered along with the effect parameters used. Parameters to the effects can be changed at a later time, and at any time the effect can be re-rendered with the saved effect parameters. Therefore it is very important for AudioSuite plug-ins to maintain compatibility between instances, versions, and systems in order to function properly in Media Composer workflows. See [Preset management](#) for more information.

Media Composer supports two AudioSuite processing modes:

- Apply a plug-in to a clip in the Timeline. This method creates a rendered effect.
- Use the controls in the AudioSuite window to create a new master clip. This method lets you process more than one channel at a time and to create new media with a duration longer or shorter than the source media.

By default, the AudioSuite window displays the controls for applying a plug-in to a clip in the Timeline. When you drag a master clip into the window, the window expands to display additional parameters for working with master clips.

12.39.3.1.1 Applying an AudioSuite Plug-in to a Clip in the Timeline

The following illustration shows the default layout of the AudioSuite window:

Figure 1: The AudioSuite window

To apply an AudioSuite plug-in to a clip in the Timeline:

1. Open the AudioSuite window by doing one of the following:
 - Select Tools > AudioSuite
 - If an audio tool is already open, click the Effect Mode Selector menu and select AudioSuite
2. Use the Track Selection Menu button to select the tracks that you want to modify.
 - When you select an item from this menu, the system selects or deselects the corresponding track in the Timeline
 - To select multiple tracks, press the Shift key while you select additional tracks from the Track Selection menu. Plus signs (+) mark the additional tracks and indicate that the effect is applied to more than one track.
3. Click the Plug-In Selection menu, and select a plug-in
4. Click the Activate Current Plug-In button. This opens a dialog box associated with the plug-in.

From the AudioSuite dialog box, you may make any necessary adjustments to the plug-in and Preview the effect in real-time.

- To save the effect, click OK
- To close the dialog box without saving the effect, click Cancel
- To save the effect as a template, drag the effect icon to a bin

12.39.3.1.2 AudioSuite Master Clip Mode

Drag a Master Clip into the AudioSuite Tool to engage AudioSuite Master Clip Mode. This mode supports all AudioSuite effects, including those that change the width or length of the effected clip. A new Master Clip is generated for each AudioSuite processing pass applied in this mode.

In Master Clip Mode, the AudioSuite window will be expanded to display additional controls. You can also click the Display/Hide Master Clip Controls button to display or hide the additional parameters.

The following operations are available in Master Clip Mode:

- Apply AudioSuite plug-ins to more than one track at the same time. For example, a plug-in might let you process two separate tracks as a stereo pair. This enables you to use plug-ins that perform linked compression, reverb, and other effects that allow multichannel input.
- Create new media with a longer or shorter duration than the source media. This lets you use effects that perform time compression and expansion. For example, you can use a Time Compression Expansion plug-in to change the length of the audio file, or you can lengthen the file in order to add a reverb trail.
- Apply one mono AudioSuite effect to multiple inputs of a master clip in a multiple-mono fashion.

For more information about processing in Master Clip Mode, see the Media Composer user documentation.

12.39.3.1.3 Restrictions on AudioSuite processing

- Media Composer does not support width-changing AudioSuite effects except in [Master Clip Mode](#). See [Processing configurations](#) for more information about supported stem formats in Media Composer.
- AudioSuite effects that change the clip length should only be used in [Master Clip Mode](#), because consolidated sequences will not consolidate the correct media length.

12.39.3.2 Real-time processing

Use the Audio Track Effect Tool to perform real-time processing in Media Composer. Audio Track Effects appear in the Audio tab of the Effect Palette, as well as in the menus of the Audio Track inserts in the Audio Mixer Window and the Timeline Track Control Panel.

Real-time AAX processing in Media Composer is analogous to the track inserts feature in Pro Tools. For more information about track inserts in Pro Tools, see the [Real-time processing](#) section in the [Pro Tools Guide](#).

12.39.3.2.1 Creating and accessing real-time plug-in instances

To insert a plug-in effect on a track in Media Composer, select the track where you want to apply the effect, which insert location you want to use on the track, and the specific effect you want to add to your sequence.

You can also insert a plug-in track effect by dragging an Audio Track Effect template from a bin to your sequence.

To insert an Audio Track Effect plug-in from the Timeline Right-click the Record Track button or the Track Control panel for the track where you want to apply the insert and select **AAX Effects [track number] > Insert [a-e] > [insert]**.

To insert an Audio Track Effect plug-in using the insert button

1. Click an Audio Effect insert button in the Track Control panel for the track where you want to apply the insert. This opens the Audio Track Effect tool.
2. Click the Select Effect button, and select an Audio Track Effect plug-in effect. Figure 1: Select an insert in the Audio Track Effect Tool

To insert an plug-in using the Effect Palette

1. In the Project window, click the Effects tab. This opens the Effect Palette. Figure 2: The Effect Palette
2. Click the Audio tab.
3. Click an effect category, select the effect you want, and drag it to the segment or to the Audio Track Effect insert button where you want to apply the insert. This opens the Select Insert dialog box. Figure 3: The Select Insert dialog box

Note

You can only insert mono effects on a mono track, stereo effects on a stereo track, and surround sound effects on a surround sound track.

4. Do one of the following:
 - If you want to add a new insert, click an [Empty] insert button.
 - If you want to replace an existing insert, click the appropriate insert button.

The plug-in effect is inserted in the track to which you dragged the effect icon.

To edit an existing Audio Track Effect Plug-In After you insert an Audio Track Effect plug-in on an audio track, you can access the plug-in controls by using the Track Control panel or the Audio Track Effect tool.

Figure 4: Audio Track Effect plug-in inserts in the Track Control panel Figure 5: Audio Track Effect tool: Select Track, Select Insert, and Select Effect buttons (left), Bypass button (center), and Save Effect button (right)

When you select an insert button in the Track Control panel or an effect in the Audio Track Effect tool, the controls for the plug-in appear in the Audio Track Effect tool window.

Figure 6: The Compressor/Limiter Dyn 3 plug-in window displayed in the Audio Track Effect tool dialog box

You can also open the tool by selecting Tools > Audio Track Effect Tool or right-clicking the Record Track button for the track where you want to edit an insert and selecting Audio Track Effect tool. You can use the buttons in the tool to select a specific insert to edit.

To save changes to a plug-in's settings, do one of the following:

- Click the Save Effect icon in the Audio Track Effect tool
- Close the Audio Track Effect tool

12.39.3.2.2 Using Audio Track Effect Templates

If you apply an Audio Track effect and make a set of adjustments to it, you can quickly recreate the same sound on other tracks in your sequence or project. You can save an Audio Track effect with its parameter settings to a bin as an effect template. You can then apply the template to other audio tracks at any time.

You can apply an Audio Track effect template with all its parameters directly to an Audio Track Effect insert button in the Track Selection panel or to clips in the Timeline.

To save an Audio Track Effect as a template Do one of the following:

- Click the Save Effect button in the Audio Track Effect tool and drag it to a bin
- Click an Audio Track Effect button and drag it to a bin

A new track effect template appears in the bin, containing the parameter setting information for the effect. The new effect template is identified in the bin by an effect icon. By default, your Avid editing application names the template by the plug-in name.

To apply an Audio Track Effect template to an audio track Do one of the following:

- Drag the Audio Track Effect template from the bin to an insert button in the Track Selection panel
- Drag the Audio Track Effect template from the bin to a segment on the track where you want to apply the effect. The Select Insert dialog box opens so you can select the insert where you want to apply the effect.

This applies the effect to the track.

12.39.4 Compatibility requirements

Media Composer supports 64-bit AAX Native plug-ins beginning in Media Composer 8.1. There are no Media Composer versions that support 32-bit AAX plug-ins, and Media Composer does not currently support AAX DSP plug-ins.

In addition to implementing the client-side AAX API for a supported platform, Media Composer AAX plug-ins must:

1. Be installed to the AAX plug-ins directory
2. Use a valid file name

12.39.4.1 Install directories

AAX plug-ins must be installed in the system's AAX Plug-Ins directory. See [Building your plug-in installer](#) for more information about creating a plug-in installer.

Host Compatibility Notes Some early versions of Media Composer 8 do not search the system plug-ins directory recursively. If your plug-ins are installed into a sub-directory beneath this main directory then they will not be loaded by the affected versions of Media Composer.

Plug-ins that are uninstalled but still present on the system are placed into the "Plug-Ins (Unused)" directory, which is located next to the Plug-Ins directory.

Media Composer will also search for a Plug-Ins directory next to the actual Media Composer application, and this directory will be used if present. This debug feature can be useful for testing specific plug-ins.

12.39.4.2 Plug-in name and file structure

In order to be recognized by AAE, all AAX plug-in bundles must use the ".aaxplugin" file name suffix. On OS X, the plug-in bundle must use this suffix while the binary itself does not require a suffix. On Windows, the plug-in binary (DLL) must use this suffix.

The directory structure of an AAX plug-in bundle is also important. See [.aaxplugin Directory Structure](#) in the [AAX Format Specification](#) document for more information.

12.39.5 AAX feature support in Media Composer

Media Composer supports many of the same AAX features as Pro Tools. However, some features are not available in Media Composer, and other features are managed differently between the two applications. This section describes how Media Composer handles various optional AAX features.

12.39.5.1 Processing configurations

Sample rates Media Composer operates at sample rates of 32000, 44100, 48000, 88200, 96000 Hz, as well as each rate's film pulldown version scaled by a ratio of 1000/1001: approximately 31968, 40959, 47952, 88111, 95904 Hz.

Note

The AAX API does not currently provide a selector for 32 kHz sample rate support

Track formats Media Composer supports only four track formats:

- Mono
- Stereo (interleaved L/R)
- 5.1 surround in Pro Tools order (L, C, R, Ls, Rs, Lfe)
- 7.1 surround in Pro Tools order (L, C, R, Lss, Rss, Lsr, Lsr, Lfe)

Effects will only see these track formats on input.

Note

Plug-ins that support width-changing configurations between supported and unsupported track formats are not compatible with Media Composer

Channel ordering for plug-ins in Media Composer is identical to the channel ordering in Pro Tools. The channel ordering presentation to users may vary from the channel ordering that is used when sending audio buffers to Pro Tools; Media Composer re-orders channels to Pro Tools order prior to presenting the audio to the effect.

12.39.5.2 Preset management

Media Composer stores plug-in presets in several locations within the app. Presets may be stored and accessed through the following workflows:

- Presets can be stored in Media Composer bins by dragging the pink effect icon from the top of the effect editor window into a bin window.
- Track effect presets are stored with their tracks in the sequence
- AudioSuite presets are stored with their audio clips in the sequence

The storage of AudioSuite presets with clips in Media Composer is very different from Pro Tools. To ensure compatibility with Media Composer, it is very important that any AudioSuite effect can be re-rendered from the source media at any time.

12.39.5.2.1 Plug-in preset compatibility and persistance

It is always important to design AAX plug-in preset data in a way that will be compatible across different systems and at different points in time. This is particularly true when designing an AAX plug-in to be compatible with Media Composer.

Media Composer sequences carrying presets can be exported as AAF, and these sequences may be moved freely between Media Composer systems on different operating systems and platforms. Therefore, it is important that plug-in preset data is not platform specific. A plug-in loaded in any given Media Composer system must be able to successfully read, parse, and apply preset information that was created on a different system.

Presets also persist for a long time in sequences, so preset information should be formatted in a way that newer plug-ins can read older version's data, and older versions can read newer version's data.

In addition, Media Composer 8.4 and higher can access factory presets and user-created presets interoperably with Pro Tools. A user can save a preset in one application, and access it in the other.

These preset compatibility considerations also apply to plug-ins carried over from legacy plug-in formats such as T↔DM/RTAS. Media Composer 8.1 and higher (with 64-bit AAX support) will match plug-in IDs when loading sequence data saved with Media Composer 7 and below, which use older plug-in formats. The same system is used for matching plug-in IDs when moving presets between different versions of Pro Tools, and between Pro Tools and Media Composer: in all cases, a preset saved for a particular plug-in ID must be compatible with all other plug-ins that use that ID, regardless of the plug-in format.

12.39.5.2.2 Plug-in preset data comparison

Media Composer's rendered AudioSuite effect feature relies on a comparison of plug-in settings chunk data. Unlike in Pro Tools, this operation uses direct data comparison rather than [AAX_IEffectParameters::CompareActive↔Chunk\(\)](#). Therefore, Media Composer compatibility and proper operation of AudioSuite rendering in Media Composer depends on the plug-in having fully consistent AAX preset contents from one run to the next.

Two specific areas where problems can occur are:

- Uninitialized memory in the preset chunk data
- Floating point values in the preset chunk data

Both of these can result in differences between settings chunks representing the same plug-in state, which causes Media Composer to perpetually re-render the plug-in.

The problem of uninitialized memory is obvious. Given a particular plug-in state, Media Composer expects that any retrieved settings chunk will contain matching data regardless of when the chunk is retrieved. When the chunk contains uninitialized data this data does not match between different retrieved chunks. The fix, of course, is to make sure the entire chunk is initialized, for example by setting the entire chunk to zeroes before filling in the data.

The problem of floating point values is more subtle. Depending on the plug-in's parameter implementation, floating point values may be slightly different in the lowest-order bits when set onto the plug-in as part of an incoming chunk and when subsequently read out. When this occurs, Media Composer sees a mismatch in the chunk data, which causes the AudioSuite plug-in to unexpectedly be seen as requiring a new render.

AAX plug-in developers will need to avoid both of these conditions in order to maintain compatibility with Media Composer's AudioSuite effect rendering model.

12.39.5.3 Unsupported features

The following AAX features are not supported by Media Composer. Plug-ins that require these features will not be compatible with Media Composer. If your plug-ins use these features for advanced functionality but not for basic operation then you should document this restriction for Media Composer users.

- Advanced audio routing Media Composer has a simplified audio topology with only tracks and a single master fader. There are no side chains, no busses, and no submasters. As a result, Media Composer does not support extended routing options such as [Sidechain Inputs](#) or [Auxiliary Output Stems](#)

- Transport interface Media Composer does not fully support the [AAX_ITransport](#) interface. In addition, early versions of Media Composer 8 that do not support this interface at all may incorrectly return [AAX_SUCCESS](#) to method calls on this interface. More recent versions of Media Composer will either provide valid information or return [AAX_ERROR_UNIMPLEMENTED](#).
- MIDI Media Composer does not support MIDI routing to and from plug-in instances, and no [AAX MIDI features](#) are supported by Media Composer.
- External control surfaces Although Media Composer does support external control surfaces for some editing functions, it is not currently possible to control plug-in parameters using external control surface hardware in Media Composer.

12.39.5.4 Additional feature support notes

- [AAX](#) plug-in notification support varies between Media Composer and Pro Tools. Media Composer does not support the following notifications, and may not support additional notifications as well:
 - [AAX_eNotificationEvent_ASProcessingState](#)
 - [AAX_eNotificationEvent_ASPreviewState](#).
 - [AAX_eNotificationEvent_SessionBeingOpened](#)
- Media Composer does not support AudioSuite rendering to a separate track (see [AAX_eProperty_<--DestinationTrack](#))

12.39.6 Additional Information

12.39.6.1 Audio Engine features and behavior

Media Composer shares the same audio engine as Pro Tools (AAE) and both applications share the same advanced audio processing features. However, some aspects of plug-in operation are different between the two apps.

Here are some important notes regarding how Media Composer handles plug-in instances within the audio engine:

- Media Composer only runs plug-ins when Media Composer is playing. Unlike Pro Tools, Plug-ins stop processing when Media Composer stops playing.
- Media Composer buffer sizes are always 1024 samples, and execution is not strictly linked to real-time. Processing is generally between four and eight frames ahead of when the audio is heard.
- Media Composer will render, mix down, and export real-time effects as fast as the processor will allow, typically much faster than real-time, so be careful of introducing real-time dependencies.
- Media Composer has a background render capability, so you cannot expect the GUI to be available, or even be possible on the system performing the render.
- Plug-ins are frequently disposed and re-created on their preset data. This happens with every edit that changes the number, length, or position of playable clips in the timeline.

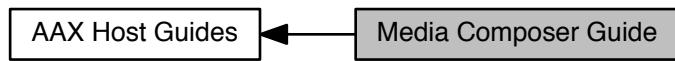
For more detailed information about how AAE handles plug-in loading and processing, see [Audio Engine Behavior and Features](#) in the [Pro Tools Guide](#).

12.39.6.2 Debugging AAX plug-ins in Media Composer

Media Composer does not support attaching a debugger in order to debug plug-ins while they are loaded within the app. In addition, Avid does not currently provide debuggable "developer build" versions of Media Composer. You must therefore rely on logging information for debugging your plug-ins in Media Composer, or debug your plug-ins using other AAX hosts such as a Pro Tools development build or the DigiShell command-line environment.

For more information about debugging in Pro Tools and DigiShell, see [Debugging AAX plug-ins in the Pro Tools Guide](#).

Collaboration diagram for Media Composer Guide:



12.40 TI Guide

How to write AAX plug-ins for Avid's TI-based platforms.

12.40.1 Contents

- [Overview of TI Algorithms in AAX](#)
- [The HDX Platform](#)
- [Requirements for TI Plug-Ins](#)
- [TI Development Tools](#)
- [Common Issues with TI Development](#)
- [TI Optimization Guide](#)
- [Error Codes](#)

12.40.2 Overview of TI Algorithms in AAX

Avid's hardware-accelerated audio systems allow AAX plug-ins to offload their real-time processing tasks to a dedicated processor, guaranteeing reliable performance at ultra-low latency. Avid's TI-based products utilize Texas Instruments DSP chips to host plug-ins in a managed shell environment.

The AAX host handles all system-level communications and resources on the DSP and provides a consistent API to manage communication between the plug-in's real-time algorithm and its other components. This design allows AAX plug-ins to use the same communication methods whether they are running natively, on a TI-based accelerated system, or in some other distributed environment.

Each AAX plug-in contains a real-time algorithm callback. For TI DSP-based platforms, this callback is compiled into a relocatable ELF DLL. This library is loaded onto the appropriate DSP by the host, and may share the DSP with other plug-ins if the host determines that the required system resources are available.

12.40.3 The HDX Platform

HDX is Avid's PCI-based core mixer and plug-in accelerator platform. Each HDX card includes 18 TI C6727 DSPs, each clocked at 350 MHz. These DSPs utilize a 32-bit floating-point architecture, with the option to perform 64-bit double-precision operations at some performance cost.

12.40.3.1 DSP characteristics: instruction processing

The C6727 DSP utilizes a VLIW architecture and contains dual data paths. Each data path includes four independent functional units, so the DSP can accommodate up to 8 parallel instructions per cycle. To take advantage of this architecture, the TI compiler relies heavily on instruction pipelining for optimization.

In order to realize the maximum possible performance benefit from this architecture, HDX uses a four-sample processing quantum. Plug-ins that require additional processing time per callback, e.g. to mitigate the overhead cost of the chip's DMA facilities, may optionally request a 16, 32, or 64-sample quantum. But please note that at higher block sizes, the number of potential I/O channels available to plug-ins on a chip will be reduced. By guaranteeing that each algorithm will be called with a consistent buffer size, the TI compiler is able to properly account for any possible iterative instruction pipelining, resulting in large performance gains.

Host Compatibility Notes 32 and 64-sample quantum is available in Pro Tools 10.2 and higher

12.40.3.2 DSP characteristics: memory

Each DSP on the HDX platform includes 16 MB of external RAM and 256 kB of internal RAM. The DSP has the ability to execute code from either internal or external RAM, though the real-time performance cost of external RAM accesses is significant. The chip's internal RAM is addressable at the core clock rate.

Each DSP also has a program cache of 32 kB. Plug-in code is loaded into this cache from internal memory, so for best performance your plug-in should not use more than 32 kB for its program code. You can look at the CCS-generated .map file to find your plug-in's program code size.

12.40.3.2.1 SDRAM performance

Asynchronous access to data in the C6727's SDRAM is very slow, requiring 50 cycles/word to read and 15 cycles/word to write. This is primarily due to clock domain bridging, lack of data caching, and the fact that data from the core is given a low priority in order to avoid stalling real-time DMA transfers.

12.40.3.2.2 Executing program code from external memory

The TI C6727 supports executing program code from external memory. When executing from uncached external memory, expect cycle counts to increase by a factor of 4x to 5x compared with the equivalent internal-memory code. Assuming that no cache thrashing occurs, subsequent calls will be cached and thus the program's location in either external or internal memory will produce similar cycle counts.

Note

The CCSv4 Profiler contains a bug that produces incorrect cycle counts for cached external-memory program code. Therefore, when gathering cycle count data for a plug-in that stores its program data in external memory, an RTI-based timing method should be used.

12.40.3.3 System characteristics: DSP/host data transfers

Plug-ins loaded onto the HDX platform may transfer arbitrarily large data blocks between the DSP and the host, within the limits of available DSP memory and system bandwidth.

12.40.3.3.1 DSP/host bandwidth

The recommended upper limit for DSP/host data transfer requests in an individual plug-in is 10 MB/s, divided by the maximum number of plug-in instances that will run on a single chip. On the HDX card, DSPs are wired to the FPGA crossbar in groups of three, with a data bandwidth of approximately 67 MB/s for each group. The overall system bandwidth for each DSP is therefore approximately 20 MB/s. This bandwidth is shared by all data reads and writes, including custom data transfer requests.

HDX does not include any explicit plug-in bandwidth limiting constraints. If a plug-in's data transfer requests bump up against the physical bandwidth limit for the system then this will delay the blocking data transfer request on the host, as the transfer will be held off for higher-priority operations on the DSP, and may also delay automation data from reaching other plug-ins on the three affected DSPs.

12.40.3.3.2 DSP/host data transfer characteristics

The minimum data transfer size for all host-to-DSP communications in HDX is 128 bytes. This limit applies to all host-to-DSP data transfers, including data sent to buffered ports, unbuffered ports, and private data blocks (via the AAX Direct Data interface.)

Since each transfer has a minimum size of 128 bytes, the use of many small packets does not increase transfer efficiency or save system bandwidth. Quite the opposite: updating a single 64-byte packet would require less bandwidth than updating two 4-byte packets in an HDX system, since the former would require only one 128-byte transfer while the latter will require two.

12.40.3.4 TI Shell characteristics: Memory allocation

12.40.3.4.1 Memory resource availability

The TI Shell code that is loaded onto each DSP uses approximately 56 kB of internal memory, leaving 200 kB of internal memory per DSP. This memory is shared between the plug-ins on the chip and holds the plug-ins' code and data, per-instance blocks declared in `Describe()`, and instance overhead.

As a general guideline, plug-in instances should not use more than 200 / n kB of internal memory, where n is the number of instances of your plug-in that will run on a single chip based on its cycle count requirements.

12.40.3.4.2 Shared and per-instance memory allocation

When a plug-in instance is created on a DSP, its program code is loaded onto that DSP. This copy of the program code is then re-used for all subsequent instances of the effect that are loaded onto the DSP. Static and global data are also shared between all instances of an effect on the DSP. Other allocations, such as coefficient and private data blocks, are per-instance.

Host Compatibility Notes Beginning in Pro Tools 11, AAX DSP algorithms also support optional temporary data spaces that can be described in the `Describe` module and are shared among all instances on a DSP. This is an alternative to declaring large data blocks on the stack for better memory management and to prevent stack overflows. Please refer to [AAX_IComponentDescriptor::AddTemporaryData\(\)](#) for usage instructions.

12.40.3.4.3 Placing data into external memory

An AAX plug-in may optionally request that its private data or program code be placed into external memory. Because standard access calls to the DSP's SDRAM are very slow, it is strongly recommended that all of a plug-in's real-time data be placed in internal RAM, and the TI Shell will load a plug-in's program code and all private plug-in data blocks into internal memory by default.

Requesting more than 256 kB of data in internal memory for plug-in data plus the memory required by the TI Shell will lead to undefined behavior, so it is important to explicitly request external memory for plug-in data when appropriate.

For private data blocks that should be loaded into external memory, use the [AAX_ePrivateDataOptions_External](#) flag when calling [AAX_IComponentDescriptor::AddPrivateData\(\)](#). This flag will be ignored by the host, so Native AAX plug-ins will have the same functionality with or without this property.

To load program code, static data, or global variables into external memory, use the `TI SECTION` pragmas. For example, `#pragma CODE_SECTION_(".extmem")` can be used before function definitions that are either initialization code, or infrequently used background code. For static variables, use `#pragma DATA_SECTION_(".extmemdata")` before each variable definition.

12.40.3.4.4 DMA support

Because of the slower access time of external RAM, you should consider using a [DMA transfer](#) for recurring transfers, and possibly even for larger one-time transfers. This is of particular relevance for data reads, which must traverse the various clock domains and priority switches twice (address send, and then data return.)

The TI Shell supports three DMA modes: Scatter (for transfers from internal to external memory), Gather (for transfers from external to internal memory), and Burst (contiguous block copies). The Scatter mode can accomplish transfer speeds of up to 2.1 DSP cycles/byte transferred, while the Gather mode can accomplish 2.7 cycles/byte transferred.

The Scatter and Gather DMA facilities use a linear buffer for internal memory and a FIFO for external memory. It is possible to transfer to or from multiple offsets within the external memory FIFO using an offset table, which can contain up to 65,536 (216) entries. The offset (burst) length may be 4, 8, 16, 32, or 64 bytes long.

The TI Shell also supports a Burst DMA mode which implements linear data reads or writes.

For more information on DMA support and for example code, see `\ExamplePlugIns\DemoGain_DMA` in the SDK.

12.40.3.5 TI Shell characteristics: Data packet services

In addition to supporting direct transfers of arbitrary data via DMA, the TI Shell also supports a packetized data delivery mechanism for host-to-DSP data transfers. Packet delivery ports may be either unbuffered or buffered, and are described using the [AAX_EDataInPortType](#) parameter in [AAX_VComponentDescriptor::AddDataInPort\(\)](#).

12.40.3.5.1 Unbuffered ports

Unbuffered ports use a straightforward implementation that delivers posted packets to the algorithm as soon as possible. In an unbuffered port, newer packets will always override older packets. Therefore, an algorithm may not receive every packet that was posted to an unbuffered port, but it will always receive the most up-to-date information possible.

Unbuffered ports deliver their data without blocking or synchronizing with the algorithm's execution. Although bus arbitration guarantees that a read from the algorithm callback will not occur in the middle of a write from the host, it is important to note that the data in an unbuffered port may change during algorithm execution.

12.40.3.5.2 Buffered ports

Buffered data ports store incoming packets in a host-managed queue. This queue acts as a buffer and provides the host with more flexibility in how it delivers packets. A key feature of buffered data ports is that new data will never be delivered to these ports during algorithm execution.

The behavior of buffered data ports varies depending on the host platform. In HDX plug-ins, Buffered data ports use a FIFO to queue data packets as they are posted. New packets are dequeued and delivered to the algorithm individually, with the next packet arriving before each algorithm render callback.

12.40.3.5.3 Data port overhead and restrictions

Each HDX DSP supports a maximum of 164 buffered data ports, which matches the maximum I/O limit for each DSP. System overhead costs associated with using the on-chip packet services are as follows:

Memory Overhead

- The memory overhead for an unbuffered data port is simply the size of the data packet.
- This DSP memory overhead for a buffered data port is two times the size of the data packet. A large (>100-element) packet queue is also allocated on the host.

CPU overhead

Unbuffered ports do not incur any additional CPU overhead.

Individual buffered ports do incur non-trivial CPU overhead. For example, in Pro Tools 10.2 each buffered port requires 5 cycles of overhead per render callback. This overhead can quickly add up in "small" plug-ins that contain many buffered data ports. Therefore, we strongly recommend that plug-ins use consolidated coefficient packets when possible in order to minimize this overhead. This optimization can result in large performance gains for callbacks that require 1000 or fewer cycles to operate.

The trade-off of this optimization is that more work ends up being done on the host and more data must be transmitted to the algorithm, since the entire coefficient packet must be re-calculated and re-sent every time any of its input parameters change. This is usually beneficial trade-off to make, especially given the 128-byte per-transfer minimum discussed above. However, care must be taken in extreme cases such as when packet delivery threatens to bump up against the maximum recommended bandwidth for host/DSP data transfers.

12.40.3.6 TI Shell characteristics: Instance allocation

12.40.3.6.1 Multi-shell packing

With a few exceptions, AAX DSP plug-ins will share DSPs with other plug-ins. This occurs transparently to the plug-in due to the fact that all system resource management is handled by the TI Shell.

When a new plug-in instance is created, the TI Shell and AAX host will attempt to intelligently allocate it to a DSP based on both memory and CPU resource requirements. If one plug-in on the chip requires a large amount of memory and very few processing cycles, it may be packed with another plug-in that does not require much memory but that is very CPU intensive.

The exceptions to this model are plug-ins that use DMA, register for a background processing callback, register a maximum number of instances per chip or use a processor affinity constraint when reporting CPU requirements. With the exception of a processor affinity, these plug-ins will receive dedicated DSPs to which only additional instances of the same plug-in type will be added.

Host Compatibility Notes Beginning with Pro Tools 10.2, the TI shell supports a "processor affinity" property, which indicates that a DSP ProcessProc should be preferentially loaded onto the same DSP as other instances from the same DLL binary. This is a requirement for some designs that must share global data between different processing configurations.

Note that this property should only be used when absolutely required, as it will constrain the DSP manager and reduce overall DSP plug-in instance counts on the system.

12.40.3.6.2 DSP Shuffles

A DSP shuffle will occur in Pro Tools when the engine must re-allocate DSP resources in order to make more processing power available. A shuffle will force the re-instantiation of the plug-in's DSP algorithm component, potentially on a new chip, while leaving the plug-in's host objects intact. During a shuffle, the engine will perform the following steps:

1. Disconnect audio from an effect
2. Call instance initialization with the removing instance flag on the old location
3. Repeat for all instances of all DSP Effects in the system
4. Load the effect in the new location
5. Re-send the last packets to all data-in ports
6. Call private data init for any private data
7. Call instance init with the 'adding instance' flag, in the new location
8. Begin audio processing
9. Reconnect audio
10. Repeat the instantiation and connection process for all instances of all DSP Effects in the system

Note that the system may perform some audio processing with each new instance before all of the Effect instances in the system have been re-instantiated.

12.40.3.7 Additional TI Shell services

12.40.3.7.1 Background processing

AAX plug-ins may request idle time from the main TIShell thread. This results in a true idle context callback which can be used for non-critical [background processing](#) tasks on the DSP. This facility restricts the DSP to only allocate plug-in instances of the same type.

A plug-in's background processing callback is not provided with a reference to the plug-in's data structures and must therefore access plug-in data via global variables. The background process will be interrupted by system events and the audio render callback. For more information and an example on how to create a plug-in that relies on background processing, see `\ExamplePlugins\DemoGain_Background` in the SDK.

12.40.4 Requirements for TI Plug-Ins

12.40.4.1 Plug-in description

To support AAX TI DSP platforms, a plug-in must add a TI ProcessProc (real-time processing entrypoint) for each of its algorithms. This is done via a call to [AAX_IComponentDescriptor::AddProcessProc_TI\(\)](#), which is parametrized with the names of both the algorithm's TI DLL and of its exported entrypoint.

At minimum, the TI ProcessProc requires the following AAX Properties:

- A TI plug-in ID: [AAX_eProperty_PluginID_TI](#)
- The audio buffer size that will be used by the ProcessProc: [AAX_eProperty_DSP_AudioBufferLength](#), set with a value from [AAX_EAudioBufferLengthDSP](#)

12.40.4.2 Performance measurement and reporting

In order to determine each algorithm's resource requirements, the host collects cycle count information from the plug-in via the plug-in's Describe callback. Each plug-in Effect is responsible for correctly reporting its algorithms' cycle counts for each accelerated platform that it supports. For plug-ins that use DMA or background threads, a maximum per-chip instance count is also required.

Note

All reported values must represent the algorithm's worst case performance.

Each of these values are reported as properties of a given algorithm ProcessProc and are provided by the plug-in via [AAX_IComponentDescriptor::AddProcessProc_TI\(\)](#). If an effect does not report its cycle count usage then it will be limited to a single instance per TI chip. This can be useful during development, but is not a supported mode for general use; all shipped plug-ins must correctly report their cycle requirements.

Development Builds of Pro Tools include DigiShell, a utility that can be used to accurately measure plug-in cycle count requirements. For more information about DigiShell, see [DSH Guide](#).

12.40.4.2.1 Shared vs. per-instance cycles

Because a single call into a plug-in is used to process multiple instances of that effect on that chip, two cycle count properties must be reported for each TI algorithm:

1. [AAX_eProperty_TI_SharedCycleCount](#) This property describes the algorithm's one-time processing overhead that doesn't change as instances are added to a chip.
2. [AAX_eProperty_TI_InstanceCycleCount](#) This property describes the additional cycle counts that each instance adds to the base shared overhead.

Many plug-ins exhibit different performance characteristics for both of these metrics depending on the plug-in's state. When reporting a plug-in's shared and per-instance cycle count requirements it is important to ensure that the reported values are the *maximum possible requirements* of the algorithm.

Often a plug-in will experience its worst-case per-instance processing load in one configuration and its worst-case shared processing load in another configuration. In this situation, the plug-in's reported cycle count requirements should reflect the state in which the *sum* of the two metrics is highest.

It's a common practice to not describe [AAX_eProperty_TI_InstanceCycleCount](#) and [AAX_eProperty_TI_SharedCycleCount](#) for the plug-ins during development and debugging process of the DSP plug-ins. This is acceptable, although in this case the one instance of such a plug-in will require the whole chip. In AAX SDK example plug-ins this is implemented using [AAX_TI_BINARY_IN_DEVELOPMENT](#) macros. If defined, it turns off the cycle count properties for the plug-in.

12.40.4.2.2 Measuring shared cycles

Measuring shared cycle counts requires instantiating multiple instances of an effect and observing how the processing time changes as instances are added. The shared and instance cycle counts are then calculated by performing a linear regression on the number of uncached cycle counts as the number of plug-in instances on the chip increases.

Note that these values will differ between debug and release builds of an algorithm, so a plug-in's describe function should report the correct cycle count values based on the relevant build configuration.

DigiShell includes the ability to measure shared cycle counts using the `DAE.cyclessshared` command. For more information about performance profiling using DigiShell, see [Cycle count performance test](#).

Note

HDX requires reporting of an algorithm's *worst-case* cycle counts.

12.40.4.2.3 DMA and background thread performance reporting

For algorithms that use [DMA](#) or [background thread](#) facilities, the maximum number of algorithm instances that will fit on a chip is difficult to predict from cycle counts alone. Due to the asynchronous behavior and limited capacity of the DMA system, the DMA system may begin to miss its deadlines before the CPU is fully loaded. In addition, due to differences in background processing requirements between algorithms, an effect's background process may begin to miss its deadlines and be starved before the interrupt-time audio processing is at capacity. Plug-ins that use these facilities must therefore report the maximum number of instances that will run reliably at a given sample rate, in addition to reporting their shared and per-instance cycle counts as above.

Maximum reliable instance counts are reported using an additional property, `AAX_eProperty_TI_MaxInstancesPerChip`. A plug-in should register separate components for the following three sample rate ranges in order to register distinct values for this property:

1. Sample rates from 42kHz to 50kHz
2. Sample rates from 84kHz to 100kHz
3. Sample rates from 168kHz to 200kHz

Notes regarding DMA and background thread performance reporting:

- Because the number of instances will decrease as sample rate increases, the plug-in must be tested at the highest available pulled-up sample rate (i.e. 50kHz instead of 48kHz) in each of these three ranges.
- On the HDX platform, effects that use DMA or background threads will not be mixed with effects of other types on a given chip.
- The maximum number of instances per DSP cannot be measured via DSH in these cases, so careful listening tests must be manually performed in order to determine whether a certain number of instances of a DMA or background-enabled plug-in actually operate correctly on a DSP.

12.40.4.2.4 Dynamic resource usage

All resources used by an AAX DSP plug-in algorithm are considered static. Plug-ins may not dynamically change the amount of memory or DSP cycles that are allocated to them after these metrics are provided in `Describe`.

The ability to dynamically change DSP cycle count requirements at run time is provided in the AAX SDK but is not currently supported by any host.

12.40.4.3 Plug-in compilation and packaging

12.40.4.3.1 Exported symbols

Each TI algorithm (ELF DLL) may contain multiple entrypoints. A single DLL may be used for all of your plug-in's entrypoints and program code, or you may divide your plug-in's entrypoints and program code between multiple DLLs.

Your plug-in must export one "C"-style callback for each algorithm ProcessProc that your plug-in registers. This entrypoint must conform to the standard AAX real-time algorithm callback prototype:

```
# include "elf_linkage.h" // Includes required TI_EXPORT definition
extern "C"
TI_EXPORT
void
MyEffect_AlgorithmProcessFunction(
    SMyEffect_Alg_Context * const inInstancesBegin [],
    const void *           inInstancesEnd)
```

Listing 1.1: The standard AAX real-time algorithm callback prototype

For Code Composer Studio projects from Code Composer Studio version 5 and higher (running Code Generation Tools 7.4.x or higher), you should include the following header instead of elf_linkage.h:

```
# include "elf_linkage_aax_ccsv5.h"
```

Listing 1.2: Header which should be included into all CCSv5 plug-in projects.

It is located in AAX_SDK/TI/CCSv5 folder, so you will also need to add this path to the include path of your projects.

Note

- There is a compiler option in Code Composer Studio that will add an underscore to the exported entrypoint's name. We recommend keeping this option disabled in order to avoid ambiguity between the exported symbol name and the function name as it appears in your source code.
- If you encounter undefined symbol errors when linking to a DSP library that uses a C-style interface then add the extern "C" keyword before the lib function prototypes. This should resolve the majority of such linker errors.

12.40.4.3.2 Packaging

The ELF DLLs for an AAX DSP plug-in must be placed in the ./Content/Resources directory within the plug-in bundle.

12.40.5 TI Development Tools

Development for TI algorithms is primarily performed in TI's Code Composer Studio. Code Composer Studio (C↔CS) is a full-featured, Eclipse-based IDE providing JTAG hardware debugger support, a hardware simulator, and a suite of profiling tools. Most importantly, CCS includes an excellent C compiler that is capable of providing highly optimized DSP instructions without too much tuning.

Note

As of this writing, Code Composer Studio for Mac does not support the C6000 series processor. CCS for Windows is required for AAX DSP plug-in development. See [MacOS Host Support CCSv7](#) on the Texas Instruments wiki for current compatibility information.

12.40.5.1 Code Composer Studio

The AAX SDK supports Code Composer Studio versions 4 ("CCSv4") and higher ("CCSv5", etc.), with hardware debugging support beginning in version 4.2. As of the writing of this documentation, CCS versions 4, 5, and 7 have been tested by Avid.

Note

This documentation was originally written for CCSv4 and was later updated with instructions for updating from CCSv4 to CCSv5. Versions 5 and higher use a different project file format from version 4; when this documentation describes changes required for version 5 then these changes will also be required by other later versions which use this new project format.

12.40.5.1.1 Installation

1. Download and install the latest Code Composer Studio from TI's website.

Note

Windows 10 requires Code Composer Studio version 6.1.3 or higher

As of Code Composer Studio version 7 TI does not charge for licenses. You can simply download the tool and start using it. Along with this the end user license agreement has changed to a simple TSPA compatible license. For more information see the TI web site.

2. The default installation will work fine, but a custom install will be smaller. You only need support for the C6000 chipset and the Spectrum Digital JTAG drivers, so you can deselect all the other chipsets and JTAG drivers.
3. Go to [TI's Code Generation Tools](#) page. You will need to log in.
4. Download and install the C6000 Code Generation Tools v7.0.x or later, using the typical installation settings. For [AAX](#) DSP development you will only need support for the C6000 chipset and, if you will be using a hardware debugger, for the Spectrum Digital JTAG drivers, so you may deselect all the other chipsets and JTAG drivers.
 - (a) Launch CCS and go to Help > Install New Software...
 - (b) In the opened dialog select "Code Generation Tools Updates" in the "Work with:" drop-down list.
 - (c) Select "TI Compiler Updates" > "C6000 Compiler Tools [version]".
 - (d) Press Next and continue installation using the "typical" installation settings.

As of the publishing of this version of the AAX SDK Avid is internally using v7.4.6. Avid has tested 7.4.4 and 7.4.6, but we assume that later revisions will work as well. The latest CGTools version available as of this writing is v7.4.21.

For more information about configuring your CCS workspace with CGTools v7.4.x, see [Workspace setup](#)

12.40.5.1.2 Workspace setup

The idea of a CCS workspace is similar to a Visual Studio solution file. Note that workspaces tend to store absolute paths and developer-specific info, so you may wish to avoid checking them in to your source control server.

Setting up workspace-global macros

To set up workspace global macros:

1. When you open CCS for the first time, select a directory for your "workspace". As mentioned above, we recommend that this be outside of your source tree.

Note

Pay attention that you can not reuse your Code Composer Studio workspace after updating to a later versions. In particular, we have found the CCSv4 workspaces are incompatible with CCSv5. After updating your system to a later Code Composer Studio version you must create a new workspace and import your existing projects into this new workspace.

2. Go to File > Import... and select Code Composer Studio > Build Variables (CCS > Managed Build Macros in CCSv4.) Click Next.
3. Browse to TI/Common/macros.ini in your AAX SDK directory and click Finish.
4. This will define an "SDK_SOURCE_ROOT" Linked Resource path variable and Managed Build macro, which associates the CCS workspace with a single AAX SDK installation.

Note

A side effect of this is that you cannot use projects from multiple distinct AAX SDK installations in the same CCS workspace.

5. To verify that the correct path has been set, go to Window > Preferences... and look in General > Workspace > Linked Resources, and C/C++ > Build > Build Variables (C/C++ > Managed Build > Macros for CCSv4.)

Importing projects into your workspace

To import projects into your workspace:

1. In the IDE, go to Project > Import Existing CCS/CCE Eclipse Project
2. In Select search-directory, select the root of your AAX SDK installation
3. The projects in the resulting Projects list will automatically be selected
4. Click Finish, and then wait while the projects are imported.

In order to import CCSv4 projects into later versions of Code Composer Studio it is necessary to add a .cdtproject file to the project. If you don't have this file in your project, then you can copy it from any other existing project which was created using CCSv5 or later. Otherwise you will most likely see something similar to this error:

"Error: Import failed for project 'xxxx' because its meta-data cannot be interpreted."

If you try to build this newly imported CCSv4 project in a later version of Code Composer Studio then you will get the warning:

"This project was created using a version of compiler that is not currently installed: 7.0.5 [C6000]. Another version of the compiler will be used during build: 7.4.6. Please install the compiler of the required version, or migrate the project to one of the available compiler versions by adjusting project properties."

This warning may be cleared by changing Properties > General > Compiler Version from TI v7.0.x to the current version (e.g. TI v7.4.x). After that the "Output format" field, which is next one to the "Compiler version" field and is typically grayed out, will become active. You should choose "eabi (ELF)" there. Otherwise Code Composer the build will fail with errors:

- *--dynamic=lib not supported when producing TI-COFF output files"*
- *--export=_auto_init_elf not supported when producing TI-COFF output"*

Note

After successful conversion of the project and successful build, the remeasurement of cycle count should be done, because it may change. Most likely it will decrease, as compared to the version which was built with CCSv4, but that is not guaranteed. Also the size of the DLL may increase, which may require reducing code size in order to properly instantiate the plug-in.

12.40.5.1.3 Creating new projects

New project setup

Use the following settings in the "New Project..." wizard. Defaults are in *italics*.

- Project Type: C6000
- Output type: Executable
- Device Variant: Generic C67x+ Device
- Device Endianness: little
- Code Generation Tools: 7.4.6 or later (7.0.5 for CCSv4)
- Output format: eabi (ELF) (in CCSv4 this field will be grayed out.)
- Linker Command File: CommonPlugIn_Linkercmd.cmd (see note below)
- Runtime Support Library: <automatic>

Note

You can edit the Linker Command File setting to use the `SDK_SOURCE_ROOT` macro by manually editing the project's .project XML file or by adding the file to your project using a relative path. See the SDK sample plug-in projects for an example.

12.40.5.1.4 Recommended settings for AAX plug-in projects

Tool Settings C6000 Compiler Include Options `-include_path "${SDK_SOURCE_ROOT}/Interfaces"`
`-include_path "${SDK_SOURCE_ROOT}/[Plug-in directory]"`

The `SDK_SOURCE_ROOT` macro is defined via the macros.ini file, located in the SDK's /TI/CCSv4 directory. If you encounter errors using this macro, import the file using **File > Import... > CCS > Managed Build Macros**.

Tool Settings C6000 Compiler Command Files `-cmd_file "${SDK_SOURCE_ROOT}\TI\CCSv4\←CommonPlugIn_CompilerCmd.cmd"`

This file contains additional compiler commands that should be common to all AAX plug-in projects

Tool Settings C6000 Linker Basic Options `-o "${ConfigDir}/${PackageName}/Contents/←Resources/${ProjName}.dll"`

This path will ensure that your compiled TI DLL is placed in the appropriate location inside your AAX plug-in bundle.

Tool Settings C6000 Linker Runtime Environment (No "Initialization model" options set)

Build Settings Artifact name `${ConfigDir}/${PackageName}/Contents/Resources/${Proj←Name}`

This path will ensure that your compiled TI DLL is placed in the appropriate location inside your AAX plug-in bundle.

Build Settings Artifact extension `dll`

AAX TI libraries should use the `.dll` extension

Binary Parser Elf Parser

AAX TI libraries should use the `Elf` binary parser only

Macros Project User Macros `ConfigDir = ${OutDir}/${ConfigName}` `IntDir = ${ConfigDir}/int/${PackageName}/TI/${ProjName}` `OutDir = ${ProjDirPath}/.../.../WinBuild` `PackageName = [Plug-in name]`

These macros are used by the other settings here to ensure proper path set-up and artifact naming. Don't worry that `ConfigName` shows up as undefined - it will be defined as Debug/Release at compilation.

12.40.5.1.5 Recommended Release configuration settings

Tool Settings C6000 Compiler Basic Options `-symdebug:none -O3`

Tool Settings C6000 Compiler Predefined Symbols `-define=NDEBUG`

Tool Settings C6000 Compiler Optimizations `-os -on2 -op3`

Tool Settings C6000 Compiler Assembler Options `-keep_asm`

12.40.5.1.6 Other useful project settings

Tool Settings C6000 Compiler Predefined Symbols `-define _DEBUG`

This option is useful for differentiating cycle count reporting for Debug vs. Release builds.

Tool Settings C6000 Compiler Directory Specifier `-ft "${IntDir}" -fr "${IntDir}" -fs "${Int←Dir}"`

Useful for collecting intermediate files

Tool Settings C6000 Linker Basic Options `-m "${IntDir}/${ProjName}.map"`

Useful for placing the map file alongside all other intermediates

Tool Settings C6000 Linker File Search Path `-l (nothing)`

You can exclude libc.a, which is included by default, from this option unless you require C library features.

12.40.5.1.7 Adding files and folders

In CCS, dragging files into the project, using "Add Files to Project...", or using "Link Files to Project..." will either copy the file into the project directory or create an absolute path to the file. This is usually not the desired behavior. Use the following steps to add a file using a relative path:

1. Right click on the project you'd like to add files to, and select New > File (NOT "Source File" or "Header File").
2. Click "Advanced >>".
3. Check the box that says "Link to the file in the system". Click "Variables..."
4. Select the appropriate variable (usually either `SDK_SOURCE_ROOT` or `SOURCE_ROOT`) and click "Extend..."
5. Find the file you want to add. Click OK. Click Finish.

Note that, when adding folders, *everything* in the folder will be built by default. You can exclude files to work around this behavior.

12.40.5.2 The TMS320C6000 C++ compiler

One of the primary goals of AAX is to provide a platform-agnostic development architecture in which products can easily be developed and re-used across a wide variety of platforms. However, it is still occasionally necessary to write platform-specific code. This section will document methods for producing code that is specific to the TI C6727 platform using the TMS320C6000 C++ compiler.

12.40.5.2.1 C++ standard support

The TMS320C6000 compiler supports C++ as defined in the ISO/IEC 14882:1998 standard. The exceptions to the standard are as follows:

- Complete C++ standard library support is not included. C subset and basic language support is included.
- These C++ headers for C library facilities are not included:
 - `<clocale>`
 - `<csignal>`
 - `<cwchar>`
 - `<cwctype>`
 - `<ciso646>`
- These C++ headers are the only C++ standard library header files included:
 - `<new>`
 - `<typeinfo>`
- No support for `bad_cast` or `bad_type_id` is included in the `typeinfo` header.
- Run-time type information (RTTI) is disabled by default. RTTI can be enabled with the `-rtti` compiler option.
- The `reinterpret_cast` type does not allow casting a pointer to member of one class to a pointer to member of another class if the classes are unrelated.
- Two-phase name binding in templates, as described in `tesp.res` and `temp.dep` of the standard, is not implemented.
- The `export` keyword for templates is not implemented.
- A `typedef` of a function type cannot include member function cv-qualifiers.
- A partial specialization of a class member template cannot be added outside of the class definition.

12.40.5.2.2 Predefined environment symbols

The following symbols are predefined by the compiler on the TI architecture, and should be used in code concerned with cross-platform support:

- `_TMS320C6X` Identifies that the chip is a C6000 variant. This is the symbol that we commonly use to distinguish whether code is being compiled for AAX-Native (Mac/Windows) or AAX-TI.
- `_TMS320C6700_PLUS` Identifies that the chip is a C6700-plus variant

Although you should not require them for AAX development, equivalent assembly predefines are as follows:

- `.TMS320C6X` Identifies that the chip is a C6000 variant
- `.TMS320C6700_PLUS` Identifies that the chip is a C6700-plus variant

12.40.5.2.3 Loop controls

The TI compiler supports several pragmas that can be used to give the compiler additional information about loops.

- `#pragma MUST_ITERATE(min, max, multiple)` This pragma helps the compiler optimize loops. `min` is the minimum number of times the loop will execute, `max` is the maximum number of times the loop will execute, and `modulo` is used if the loop will only execute a certain multiple of some number.
- `#pragma PROB_ITERATE(min , max)` If extreme cases prevent the use of `MUST_ITERATE`, `PROB_ITERATE` allows you to specify the usual number of times a loop executes. For example, `PROB_ITERATE` could be applied to a loop that executes for eight iterations in the majority of cases but that sometimes may execute more or less than eight iterations.
- `pragma UNROLL(n)` Helps the compiler use SIMD instructions, where `n` is the unrolling factor. By specifying `UNROLL(1)` you can prevent the compiler from automatically unrolling a loop. In general, we recommend using `MUST_ITERATE` instead unless you have specifically identified a situation where manually unrolling a loop improves performance.

12.40.5.3 DigiShell test tool (DSH)

DigiShell is a software tool that provides a general framework for running tests on Avid audio hardware. As a command-line application, DigiShell may be driven as part of a standard, automated test suite for maximum test coverage. DSH supports loading all types of AAX plug-ins including Native and DSP, and is especially useful when running performance and cancellation tests of AAX-TI types. DigiShell is included in Pro Tools Development Builds as `dsh.exe` (Windows) or as `dsh` in the `CommandLineTools` directory (Mac).

More information on DSH test tool can be found in [DSH Guide](#).

12.40.5.4 Hardware Debugging

12.40.5.4.1 Requirements

Relocatable ELF DLLs (TI algorithms) can be debugged with some help from the DIDL loader, the TI Shell Manager, and a script called `DLLView_Elf_Avid.js`.

These are the minimum requirements for hardware debugging for TI plug-ins:

- Code Composer Studio version 4.2 or later
- XDS510 hardware debugger

- JTAG-enabled HDX card

We recommend using Spectrum Digital's XDS510 USB Plus JTAG Emulator, as it is the only one our internal developers have used and tested in-house. Both Spectrum Digital and TI have useful technical reference/installation guides, both of which can be found on the AAX Developer Forum under the 'Development Tools' discussion.

12.40.5.4.2 How it works

The ridl ELF loader inside DIDL stores a module and segment list containing the paths of all loaded modules and where their segments are loaded. The TI Shell Manager gets a serialized version of this table and loads it to a block of external memory on the chip at a known location. The `DLLView_Elf_Avid.js` script queries this memory via the debugger and extracts the paths of the modules and the ELF segment load locations, which it then passes on to the `GEL_SymbolAddELFRel` scripting console command (new to CCSv4.2). You can also use that command directly at the console.

12.40.5.4.3 Connecting a JTAG Emulator

A JTAG-enabled HDX development card includes a "riser" PCB section extending about a centimeter above the production card PCB. This riser includes two JTAG connectors. The two connectors correspond to the two banks of 9 DSPs on the HDX card. Assuming that you are instantiating your plug-in for debugging on the first available DSP, you will want to connect your JTAG emulator to the connector that is closest to the card's user-visible ports. This connector corresponds to the first 9 DSPs on the card.

12.40.5.4.4 Linking to TIShell.out

Hardware debugging, as well as several other debugging facilities, requires that the DSP plug-in project is linked to `TIShell.out` in Code Composer Studio.

To link a plug-in project to `TIShell.out`, follow these steps:

1. Open the plug-in project's properties window and navigate to the `C/C++ Build > Tool Settings > C6000 Linker > File Search Path` properties pane.
2. Add "`TIShell.out`" to the "Include library file" (`-l`) property list.
3. Under "Add <dir> to library search path" (`-L`), add the file path of the Pro Tools build you will be using to test the plug-in. This directory should already include the build's `TIShell.out` file.
4. Repeat this process for each Configuration of the plug-in project that you will be testing.
5. Add "[path to AAX SDK root]\TI" to the project's list of source file include directories

12.40.5.4.5 Adding the HDX Target Descriptor File

To add the HDX Target Descriptor File:

1. In the IDE, go to Window > Preferences, CCS > Debug. Point the "Shared target configuration directory" to `/TI/Common` in your AAX SDK source tree
2. In the IDE, go to Window > Show View > Target Configurations.
3. Click refresh if you don't see the configuration file
4. Right click `Raven_C672x_XDS510_USB.ccxml`, and click "Set as Default".

12.40.5.4.6 Setting up the DLLView script

Once you have successfully installed the XDS510, you will have to do a little bit of setup with CCS. Before starting this process, verify that you are running CCSv4.2 or later and the C6000 code generation tools v7.4 or later (or 7.0.5 for CCSv4). CCS should recognize the installed emulator and prompt you to download the necessary drivers. Once completed, you will then want to setup your DLLView script.

To set up the DLLView script:

1. In the IDE, open the Scripting Console under View > Scripting Console
2. At the Scripting console, type one of the following to load the DLLView script (insert your own source tree path, and make sure to load the version that corresponds to your installed CCS version): Code Composer Studio 4: `loadJSFile "[PATH TO AAX SDK]/TI/CCSv4/dllView_Elf_Avid.js" true` Code Composer Studio 5 and later: `loadJSFile "[PATH TO AAX SDK]/TI/CCSv5/dllView_Elf↔_Avid.js" true`

You should now see a new menu item under the Scripts menu: "DLLView -Load Pro Tools Plug-In Symbols" This should load every time CCS starts.

12.40.5.4.7 Loading Symbols for Debugging

You will need to get your code loaded and running on the TI before you load symbols. You can do this directly through Pro Tools, or by using our DigiShell test tool. If using the DigiShell test tool, load the DAE dish and then a plug-in via the following commands: `load_dish DAE` Loads the DAE dish `run` Lists available plug-ins with their index and spec `run<index>` Instantiates the <index> plug-in

Use the DLLView script to load symbols for ELF DLLs. After setting up the DLLView script and connecting to the desired chip in the Debug pane, run the "DLLView -Load Pro Tools Plug-In Symbols" script from the Scripts menu in Code Composer Studio.

Note

The chip will need to be Suspended in the debugger in order to load symbols.

To load symbols for debugging:

1. In CCS, Launch the TI Debugger (Target > Launch TI Debugger)
2. Connect the debug target to the appropriate chip
3. Suspend the chip
4. Run Scripts > DLLView -Load Pro Tools Plug-In Symbols.

Note

This script can take a moment to load; look at the Scripting Console to view its progress if you like. This script may print a warning about TIShell.out not existing. This warning is benign for plug-in debugging since the TIShell symbols are not required in this case.

This will load symbols for all symbol-rich modules running on the chip(s) connected to the debugger. If you load or unload plug-ins after this, you can simply repeat the "DLLView -Load Pro Tools Plug-In Symbols" command, which will synchronize the debugger with the current configuration.

Note

When running a plug-in in Pro Tools, the first DSP chip is reserved for the HDX mixer. Therefore the first available DSP chip for plug-in instantiation is C672x_1. Under DSH, the first available DSP chip is C672x↔_0.

12.40.5.4.8 Breaking on first entry into algorithm

To break on the first entry into the plug-in's processing routine, use the manual single-buffer processing mode in DSH: `piproctrigger manual run<index>` Attach debugger, suspend the chip, load symbols, set breakpoint, resume `piproctrigger auto`

12.40.5.4.9 Breaking in the on-chip algorithm initialization callback

It is not currently possible to hit a breakpoint in the optional on-chip algorithm initialization callback for a plug-in. If you need to troubleshoot this callback then you should use tracing to print debug information to a log file.

12.40.5.5 Tracing

Avid's AAX DSP platforms provide tracing functionality based on Avid's [DigiTrace](#) tool.

To enable trace logging for TI plug-ins, use the `AAX_TRACE` or `AAX_TRACE_RELEASE` macros defined in `AAXAssert.h`. A separate macro, `AAX_ASSERT`, is also available for conditional tracing. These macros are cross-platform and will function whether the algorithm is running on the TI or on the host.

12.40.5.5.1 Tracing requirements

- The `AAX_ASSERT` and `AAX_TRACE` macros are debug-only and will not provide tracing output from release builds of your plug-in. `AAX_TRACE_RELEASE` may be used for tracing in both debug and release configurations.
- These macros require that the `DTF_AAXPLUGINS` facility is enabled in the DigiTrace configuration file. You can toggle this facility to enable or disable AAX algorithm-level tracing.
- In order for tracing to be successful on TI platforms, your plug-in's ELF DLL must dynamically link against `TIShell.out`, a component that is installed alongside the Pro Tools application. This file includes the 'glue' that is required in order for the linker to resolve the DigiTrace entrypoint symbol in the DLL.

To link your plug-in project to `TIShell.out` in Code Composer Studio, follow the steps listed in [Linking to TIShell.out](#).

12.40.5.5.2 Tracing example

```
int32_t
AAX_CALLBACK
MyExamplePlugIn_AlgorithmInit ( SExample_Alg_Context const *
    inInstance , AAX_EComponentInstanceInitAction inAction )
{
    AAX_TRACE_RELEASE (
        kAAX_Trace_Priority_Normal ,
        "MyExamplePlugIn_AlgorithmInit called for action : %d",
        inAction );
    return 0;
}
```

Listing 2: Adding trace code on TI

12.40.5.5.3 Usage notes

- When running on the DSP, the actual handling of each tracing call occurs in a separate thread. This can lead to incorrect data reporting if volatile data, such as a pointer to an audio sample, is passed in to the tracing statement as a parameter.
- DSP tracing is most reliable when using debug TI builds and when all TI compiler optimizations have been disabled
- Known and resolved issues with DSP tracing are logged on the [Known Issues](#) page

12.40.5.6 Testing in Pro Tools

12.40.5.6.1 The System Usage window

The System Usage window in Pro Tools includes some features specifically targeted at testing DSP plug-ins, and particularly for testing shuffle events. Starting in Pro Tools 10, the System Usage window includes the following test features:

- Shift + Drag DSP Meter - This shuffles everything on the chosen chip to another chip, which allows you to quickly test shuffle for a given chip.
- Hover mouse over DSP - Presents a tooltip to show the running plug-ins on a chip
- Cmd+Option+Shift Hover - Detailed debugging tooltip info

- Cmd+Option+Shift Click - Forces a full shuffle of all chips / cards
- Click on empty chip - Reserves a DSP to prevent allocation on that chip

12.40.5.6.2 DSP information tooltip

Pro Tools can display additional information for DSP plug-ins using some debug tooltips that are hidden in the plug-in window header and the System Usage window.

The tooltip in the plug-in window header displays information about the particular plug-in instance that is currently shown in the window. To display this tooltip, hold Command-Option-Shift (Mac) or Control-Alt-Shift (Windows) and hover the mouse cursor over the DSP > Native button in the plug-in header.

The tooltip in the System Usage window displays usage information for each DSP chip in the system. You can reveal this tooltip for a particular chip by mousing over the chip's usage meter while holding Command-Option-Shift (Mac) or Control-Alt-Shift (Windows). This tooltip shows the chip's total allocated cycles, internal, and external memory.

The information in these tooltips is generally targeted at systems-level debugging, but can prove useful for some plug-in troubleshooting as well.

Figure 1: DSP tooltip in the Pro Tools plug-in window header.

Figure 2: DSP tooltip in the Pro Tools System Usage window.

12.40.6 Common Issues with TI Development

12.40.6.1 Data structure compatibility

AAX DSP plug-ins use a set of custom data structures to exchange information with host. In order to preserve a consistent binary interface between the plug-in's host and algorithm, the layout of these structures must be identical on both platforms. Each structure must have the same size when compiled by both the host platform compiler and the TI DSP compiler, and any members that are referenced by both the host code and the DSP code must reside at the same offset within the struct on both platforms.

In order to satisfy this requirement, it is essential that an AAX plug-in's algorithm context structure and any other data structures that are passed between the host and the DSP use appropriate alignment. Data structures are usually aligned to 32-bit boundaries, and both Intel and TI compilers use identical struct alignment and packing for most cases. However, this behavior is not explicitly defined in the C standard.

Furthermore, different compilers may use different sizes for some built-in data types. It is therefore very important to use explicitly-sized types such as `int32_t` and `float` rather than ambiguous types such as `bool` or `int`. One particularly tricky data type is pointers, which may be compiled as 64-bit values on a 64-bit Intel system but as 32-bit values on the TI DSP.

Here are some specific scenarios when an unexpected difference in alignment or data type size may occur and cause an ABI incompatibility between a plug-in's host and DSP components:

- [Nested structures](#)
- [Usage of pragma pack](#)
- [Dynamic allocation of memory in structures and algorithm](#)
- [Incorrect use of pointer data](#)
- [Pointer data size incompatibility](#)

12.40.6.1.1 Nested structures

It can be particularly difficult to debug alignment issues in nested data structures. One reason is that nested structs do not necessarily have the same alignment as the parent struct. A nested structure will have the alignment that is set preceding its declaration, not the alignment of the structure in which it is contained.

Aside from avoiding nested structs entirely, one way to avoid potential issues is to make sure that nested structs always contain a double. This will guarantee that the structure is double-word aligned. We have also found that placing nested structs near the beginning of the parent struct results in more consistent alignment between Intel and TI compilers, even in cases where the actual alignment of each member is strictly ambiguous according to the standard.

Another important rule of thumb with nested structs is to define them inline in the enclosing structure. We have found that including one data structure as a member in another data structure will only be reliably aligned between Visual Studio and the TI compiler tools if the member structure's type is defined in-line. This does not appear to be an issue between clang and the TI compiler - the data structure alignment for the nested structure is consistent between those two compilers regardless of the location of the internal structure's definition.

```
#include AAX_ALIGN_FILE_ALG
struct SomeStruct
{
    float a;
    float b;
};

#include AAX_ALIGN_FILE_RESET

// Somewhere else...
#include AAX_ALIGN_FILE_ALG
class SomeClass
{
public:
    SomeStruct s; // Don't do this! Inconsistent between Visual Studio and TI
    // other stuff...
};

#include AAX_ALIGN_FILE_RESET
```

Listing 3: Problematic code: nested struct not defined in-line

```
#include AAX_ALIGN_FILE_ALG
class SomeClass
{
public:
    struct SomeStruct
    {
        float a;
        float b;
    } s; // This is fine - consistent between Visual Studio, clang, and TI
    // other stuff...
};

#include AAX_ALIGN_FILE_RESET
```

Listing 4: Fixed code: nested struct defined in-line

12.40.6.1.2 Usage of pragma pack

If you use pragmas to align your structs, then you should know that in most cases it will only decrease the natural struct alignment of a compiler. That means that if you have

```
#pragma pack(8)
struct x
{
    char a;
    float b;
};
```

Listing 5: Example of usage of #pragma pack where it has no effect

then struct x most likely won't be aligned to the 8 byte boundary. Therefore the pack pragma is not really useful for addressing alignment issues. Instead of using pack, one way to guarantee that a structure is double-word aligned, is to include at least one double member.

```
#pragma pack(8)
struct x
{
    float a;
    double b;
};
```

Listing 6: Example of usage of `#pragma pack` where it actually affects the alignment of the structure
In this case data will be double-word aligned.

12.40.6.1.3 Dynamic allocation of memory in structures and algorithm

The problem with dynamic allocation is that it's difficult to enforce specific alignment of the resulting block beyond the natural alignment of the structure. Newly allocated blocks are not double-word aligned by default. This prevents double-word memory access optimizations (see [Additional data type optimizations](#)) from working.

```
// blocks are not aligned to 8-byte boundaries by default. This prevents double-word
// memory access optimizations from working
float* floatBlock = new float[100];
delete[] floatBlock;

// Though AAX_Alignment.h does include some aligned memory allocators to counteract the alignment
// problem, their use is still strongly discouraged.
float* floatBlock2 = alignMalloc<float>(100, 8);
alignFree(floatBlock2);
```

Listing 7: Problems which may arise when using dynamic allocation of memory in algorithm

12.40.6.1.4 Incorrect use of pointer data

In general, you should avoid storing pointers to anything in any data structures that are passed between the host and the DSP. There are many possible problems and bugs that can be caused by this, for example:

- Often the memory map of packets can change out from under the plug-in
- It is easy to accidentally reference data in the wrong memory space when setting pointer values
- Pointer data types are not explicitly sized (see [below](#).)

One alternative to using raw data pointers is to store data offsets into a coefficient array rather than using direct pointers to other structure elements. A solution such as this that does not involve pointer data types will almost always end up being easier to implement, easier to troubleshoot, and easier to maintain than a solution that uses pointer data.

That said, if you must use pointer data types in any data structures that are passed between the AAX host and DSP components then you should be very careful to avoid the problems listed above.

12.40.6.1.5 Pointer data size incompatibility

Problems due to pointer data size incompatibility can be particularly difficult to debug. Pointer data types are not explicitly sized in C, and, starting with the 64-bit Pro Tools 11 release, pointers will have different lengths for host and TI binaries. This can cause subtle portability problems in certain circumstances, if proper care is not taken.

Consider the following state block:

```
struct SMyPlugInStateBlock
{
    float mInGain_Smoothed;
    some_t* mPointerP;
    float mOutGain_Smoothed;
};
```

Notice the pointer `mPointerP` (the type that it points to is irrelevant for this discussion). Perhaps it is a pointer that can reference different sets of coefficients, or perhaps it points to some sort of global variable. In any case, this pointer is 64-bits long on the host, and 32-bits long on TI.

In most cases, this won't cause a problem because the host simply allocates a bit more space for the state block than the TI needs and fills the allocated memory with 0s. But consider the case where we overload [ResetFieldData\(\)](#) to set `mOutGain_Smoothed` to something other than 0:

```
AAX_Result MyPlugIn_Parameters::ResetFieldData (AAX_CFieldIndex inFieldIndex, void
* inData, uint32_t inDataSize) const
```

```

{
    AAX_Result result;
    switch (inFieldIndex)
    {
        case (eMyAlgFieldIndex_State):
        {
            memset(inData, 0, inDataSize);
            SMyPlugInStateBlock* stateP = static_cast<SMyPlugInStateBlock*>(inData);
            stateP->mOutGain_Smoothed = mOutGain_Target;
            result = AAX_SUCCESS;
            break;
        }
        default:
        {
            result = AAX_CEffectParameters::ResetFieldData(
                inFieldIndex, inData, inDataSize);
            break;
        }
    }
    return result;
}

```

We might be doing this if `mOutGain_Smoothed` was a smoothing parameter and we want to start it at the target gain value (rather than having it smooth from 0.0 at instantiation). But if the Host and TI can't agree on where in the state block `mOutGain_Smooth` is located, then the result will be unexpected behavior that is difficult to debug.

The most direct way to avoid this problem is to use an explicitly-sized 32-bit type for any pointers in your state block:

```

struct SMyPlugInStateBlock
{
    float mInGain_Smoothed;
    uint32_t mPointerP;
    float mOutGain_Smoothed;
};

```

It will be necessary to use `reinterpret_cast<float*>(stateP->mPointerP)` to recast the pointer to a pointer data type on the TI, but that should not result in any extra processing cycles.

12.40.6.1.6 Alignment Reference

These are the data type sizes and default alignments for some common compilers when compiling for 64-bit binary formats:

char	TI		MS Visual C++		C++ Builder		GCC	
	1 byte	1-byte aligned	1 byte	1-byte aligned	1 byte	1-byte aligned	1 byte	1-byte aligned
short	2 bytes	2-byte aligned	2 bytes	2-byte aligned	2 bytes	2-byte aligned	2 bytes	2-byte aligned
int	4 bytes	4-byte aligned	4 bytes	4-byte aligned	4 bytes	4-byte aligned	4 bytes	4-byte aligned
long	4 bytes	4-byte aligned	8 bytes	8-byte aligned	8 bytes	8-byte aligned	8 bytes	8-byte aligned
long long	8 bytes	8-byte aligned	8 bytes	8-byte aligned	8 bytes	8-byte aligned	8 bytes	8-byte aligned
bool	1 byte	1-byte aligned	1 byte	1-byte aligned	1 byte	1-byte aligned	1 byte	1-byte aligned
float	4 bytes	4-byte aligned	4 bytes	4-byte aligned	4 bytes	4-byte aligned	4 bytes	4-byte aligned
double	8 bytes	8-byte aligned	8 bytes	8-byte aligned	8 bytes	8-byte aligned	8 bytes	8-byte aligned
long double	8 bytes	8-byte aligned	8 bytes	8-byte aligned	8 bytes	8-byte aligned	16 bytes	16-byte aligned
pointer	4 bytes	4-byte aligned	8 bytes	8-byte aligned	8 bytes	8-byte aligned	8 bytes	8-byte aligned

Also here are some useful links to web resources on the topic:

- A good resource for the TI DSP is <http://www.ti.com/lit/an/sprab89/sprab89.pdf> (Section 2 especially). This document includes some graphs of simple alignment examples.
- Another good reference regarding general struct alignment issues is available from publib.boulder.ibm.com: <http://publib.boulder.ibm.com/infocenter/macxhelp/v6v81/index.jsp?topic=/com.ibm.vacpp6m.doc/compiler/ref/rnpgpack.htm>

12.40.7 TI Optimization Guide

Optimizing AAX real-time algorithms for Avid's TI-based platforms is very similar to optimizing real-time algorithms for any architecture. When developers think about optimization, they often think "I want to make my code run faster". In reality, however, optimization is about making the processor do less. After all, the processor's clock rate is fixed and can only perform a limited number of instructions in a set amount of time. Therefore, our focus in this section will be on helping the compiler produce code with shorter execution paths and make full use of the TI chip's architecture.

Modern compilers have become extremely powerful at being able to optimize code, which is fortunate given the complicated architectures of today's DSP products. In this section we will not focus on instruction-level "optimizations" like the one below, which will automatically be done by the compiler. Instead of making our code faster, which it won't, little "tricks" like this really just make code harder to read:

```
int y = x;
y = y >> 1; // y = y / 2;
```

Listing 8: The kind of optimization that you won't be seeing in this section

Rather, we will focus on refactoring audio processing algorithms to be more efficient and on giving the TI compiler better information about the code, pointers, and data it is working with so it can perform more effective compile-time optimizations.

Finally, our optimization efforts will focus on the worst-case code path. For example, developers often try to optimize algorithms by conditionally bypassing portions of code that may be disabled by particular parameter states. This is counter-productive, because the system has to assume a plug-in's worst-case execution performance regardless of how much time the plug-in is actually using. Therefore, in the context of real-time algorithms running on AAX DSP platforms, it is best to only worry about worst-case execution time.

For more information about using TI's toolset to profile your code's performance, see [Cycle count performance test](#).

Note

The optimizations described in this section assume that you are using version 7 or higher of TI's C6000 Code Generation Tools (CGTools). We strongly recommend using v7.0.5 as earlier versions throw linking errors.

12.40.7.1 Optimization quick start

Here is a quick outline of the general optimization steps for an AAX DSP algorithm:

1. Before beginning your DSP optimizations, make sure that your Native algorithm has basic optimizations in place. In our experience, beginning the TI optimization process with a slow or needlessly precise Native algorithm will result in a long porting process. Here are some suggestions for common Native optimizations:
 - Identify unnecessary double precision
 - Identify tables that have too high of granularity
2. Make sure your compiler Release settings enable the compiler to optimize fully and give full optimization comments: -k -s -pm -op3 -os -o3 -mo -mw -consultant -verbose -mv67p
3. Use the load/update/store design pattern to reduce memory accesses in inner loops

4. Move any processing that does not directly depend on the audio signal out of the real-time algorithm
5. Declare non-changing variables and pointers (both local and in parameter lists) as `const`
6. Declare non-aliased pointers (both local variables and function parameters) as `AAX_RESTRICT`
7. Change any `long` variables to `int`, and change `double` variables to `float` if the reduced precision does not affect signal integrity (usually defined as cancellation with the plug-in's Native algorithm.)
8. Restructure inner processing loops so that they do not contain large conditional statements or other branches
9. Declare any functions that are called within the innermost processing loop as `inline` in order to allow the inner loops to pipeline
10. Add loop count information when known, using `#pragma MUST_ITERATE(min,max,quant)`

12.40.7.2 Compiler and linker options

As with any complex environment, many performance gains on the TI rely on the appropriate compiler and linker options. The options documented here will allow CGTools to apply its optimization logic to your algorithm.

When tweaking compiler options on the TI, keep in mind that, like on any CPU, it is useless to optimize Debug code or to profile its performance. This is especially true on TI processors because of the fact that generated Debug and Release assembly is almost completely different, assuming that heavy optimization options were chosen for the Release configuration.

In general, all recommended compiler options should be set correctly in the AAX SDK's example plug-in projects, and these settings may be used as a guide for your own plug-in projects. See the SDK files `CommonPlugIn_CompilerCmd.cmd` and `CommonPlugIn_LinkCmd.cmd` for the latest recommended settings.

12.40.7.2.1 Overview of optimization-related compiler options

- `-g` Full symbolic debug. This setting should be used in debug configurations to make stepping through code easier. It should not be defined in release configurations, as it will prevent the compiler from being able to fully optimize code.
- `-k` Keep generated .asm files. This should be turned on in release configurations so that you can use the ASM output as feedback when making optimization decisions and performance improvements.
- `-d "__DEBUG"` Defines the `_DEBUG` preprocessor macro that alters how certain code is generated (asserts, stdlib, etc). This should be turned on in debug configurations only. Note that TI does not require `NDEBUG` to be defined in release configurations.

Note

This will eventually be deprecated in favor of the pre-defined `"_TMS320C6X"` macro.

- `-mv67p` Specifies that the compiler should build code for the C67x+ chip variant we are using, which has some improvements beyond the original C67x. This option should be enabled in all build configurations that target the HDX platform.
- `-s` Specifies Opt-C/ASM interlisting. This interweaves modified C-code and ASM in the .ASM file produced by the `-k` option. You should use `-s` in release configurations so that the ASM file can be read more easily.

Note

Do NOT use the `-ss` option in release configurations. This option will negatively affect optimization

- `-pm` Program mode compilation. Instructs the C compiler to compile all files in the same compilation unit, so that it can optimize code further using information from all files being compiled. See [Program Mode optimization \(-pm\)](#) for more information.

- `-op3` A modifier for the `-pm` option, this specifies that there are no external variable references in the project. This option is appropriate for TI algorithms, which do have an external function reference (the process entry point) but do not have external variable references. This option allows the compiler to further optimize global variables without worrying whether they will be accessed outside of the compilation unit. See [Program Mode optimization \(-pm\)](#) for more information
- `-o3` File-level optimization. This flag gives the compiler full ability to optimize C-code by reordering instructions, inlining functions, and performing other optimizations. Note that the resulting ASM code will be very difficult to parse back into the original C and will make debugging very difficult, so this flag should only be used for Release code. See [Optimization flags \(-o\)](#) for more information.
- `-mo` Use Function Subsections. This instructs the compiler to place all functions into their own separate subsection in the linker map. This allows the linker to remove unused functions in order to reduce memory usage.
- `-mw` Generate a single iteration view of SP loops. This flag adds important information to the ASM output file that is useful when optimizing your code for pipelined loops.
- `-verbose` Output verbose status messages when compiling files. Though not very useful for humans, verbose output will produce some key information that text parsers can use, such as compiler versions and other details.

12.40.7.2.2 Overview of optimization-related linker options

- `-relocatable` Generate a relocatable non-executable.
- `-m"file.map"` Generate a map file. This file contains useful information about the memory footprint of your plug-in, which is useful for fixing large plug-ins that may not have fit into available program memory.
- `-w` Warn about output sections. This flag generates very useful information that tells you if there might be a problem with memory output sections you are trying to generate.
- `-x` Exhaustively read libraries. This is a useful flag if you do not want to worry about the order in which you specify required libraries.

12.40.7.2.3 Optimization flags (-o)

- Register (`-o0`) This option allows for some performance gains over non-optimized code by allocating variables to registers, inlining functions declared inline, etc.
- Local (`-o1`) This option enables local optimizations, with very similar results to the register-level optimizations of `-o0`.
- Function (`-o2`) This is the standard optimization level, and provides large gains over unoptimized code. This optimization level allows function-level optimizations such as software pipelining, loop optimization/unrolling, etc.
- File (`-o3`) This option can provide some speedup beyond function-level optimizations, but also mutilates assembly code beyond recognition. At this optimization level the compiler will remove unused functions, simplify code in the case of unused return values, auto-inline small functions, etc.

Like the corresponding Visual Studio options, `-o0` and `-o1` allow you to step through code line-by-line for debugging, at the cost of reduced performance. `-o2` and `-o3` sacrifice the ability to step through code and watch memory in favor of optimized code.

12.40.7.2.4 Program Mode optimization (-pm)

Program mode optimization gives the compiler further optimization information by compiling all files at once rather than individually. Thus global constants, function implementations, etc. can be made known to the entire program at compilation. This allows the compiler to inline functions more effectively and to determine loop unrolling based on constant loop iterators.

There are a few `-pm` options:

- `-pm -op0` Contains functions and variables that are called or modified from outside the source code provided to the compiler.

- `-pm -op1` Contains variables modified from outside the source code provided to the compiler but does not use functions called from outside the source code.

This option is not appropriate for AAX plug-in algorithms, because the algorithm component will be exported and called from outside the compiled source code.

- `-pm -op2` Contains no functions or variables that are called or modified from outside the source code provided to the compiler.

This option is not appropriate for AAX plug-in algorithms, because the algorithm component will be exported and called from outside the compiled source code.

- `-pm -op3` Contains functions that are called from outside the source code provided to the compiler but does not use variables modified from outside the source code.

This is the recommended Program Mode optimization level for TI plug-ins. This optimization level requires that no global variables are used outside of the algorithm callback. In general, any such variables should be passed in to a TI algorithm via the algorithm's context structure.

12.40.7.2.5 Compiler options to avoid

The following information was taken from the TMS320C6000 Programmer's Guide:

- `-g/-s/-ss` These options limit the amount of optimization across C statements, leading to larger code size and slower program execution.
- `-mu` This option disables software pipelining for debugging. If a reduction in code size is necessary, use the `-ms2/-ms3` options. These options will disable software pipelining among their other code size optimizations.
- `-mz` This option is obsolete. When using 3.00+ compilers, this option will decrease performance and increase code size.

12.40.7.3 The load-update-store pattern

The load-update-store pattern is one of the cornerstones of a fast iterative algorithm. This pattern specifies that locally accessed data should be loaded into memory at the start of processing, accessed during processing, and stored or saved after processing has completed. By using this pattern you will move memory reads and writes outside of your plug-in's innermost processing loop, which reduces data dependencies and shortens the critical inner loop.

As an example, consider the following unoptimized filter code:

```
inline void
ProcessDirectFormII(float* input, float* output, float* state, float*
coefs, int nsamp)
{
    // eB0 .. eA2 and eA0, eA1 are just integer enums to partition
    // the filter coefficients into A and B
    for(int i = 0; i < nsamp; ++i)
    {
        output[i] = input[i]*coefs[eB0] + state[0];
        state[0] = input[i]*coefs[eB1] + state[1] - output[i]*coefs[eA0];
        state[1] = input[i]*coefs[eB2] - output[i]*coefs[eA1];
    }
}
```

Listing 9: Unoptimized filter algorithm

Notice that in this code there are at least 15 memory accesses per loop iteration! This algorithm will be very inefficient as the value of `nsamp` increases.

The compiler should be able to optimize this algorithm to some extent by pulling certain memory accesses outside of the loop. However, the compiler cannot completely optimize the loop because it must assume that the

input/output/state/coefs pointers are aliased in memory. We will discuss the `const` and `restrict` keywords later, which are ways to give the compiler additional information it can use to optimize this loop. However, for now let's focus back on the basic design of this code.

Using load-update-store, we can refactor this loop to pull the memory accesses outside of the loop:

```
void
ProcessDirectFormII (float* input, float* output, float* state, float *
coefs, int nsamp)
{
    // eB0 .. eB2 and eA0, eA1 are just integer enums to partition
    // the filter coefficients into A and B

    // ---- LOAD ----
    float coefA0 = coefs [eA0];
    float coefA1 = coefs [eA1];
    float coefB0 = coefs [eB0];
    float coefB1 = coefs [eB1];
    float coefB2 = coefs [eB2];

    float state0 = state [0];
    float state1 = state [1];

    float output;

    // ---- UPDATE ----
    for (int i = 0; i < nsamp; ++i)
    {
        output = input [i]* coefB0 + state0;
        state0 = input [i]* coefB1 + state1 - output * coefA0;
        state1 = input [i]* coefB2 - output * coefA1;
        output [i] = output;
    }

    // ---- STORE ----
    state [0] = state0;
    state [1] = state1;
}
```

Listing 10: Refactored filter algorithm with load-update-store pattern applied. Not fully optimized.

Though the code initially appears longer, you will notice that we have reduced the loop to only 4 memory accesses! Though we have an additional 9 memory accesses outside the loop, they will only occur once per function call, resulting in significant savings at higher values of `nsamp`.

Note

we are not finished with this loop yet, because we can make some very significant gains by using the `restrict` and `const` keywords, as discussed in the section on [C keywords](#).

Before moving on from load-update-store, let's consider how this pattern should be applied to different categories of data that may be provided in an AAX DSP processing context:

- Coefficients and parameters Coefficients and parameters are read-only by definition. As such, they should be loaded into a local variable at the beginning of the algorithm callback and should not be modified further.
- Private state State parameters are writable and may be changed by the algorithm. Therefore, private state data should be loaded into a local variable copy, then stored back into memory after the local copy is updated.
- Output Output is write-only, so all calculations may be performed on a local variable and then stored into memory once per loop.

12.40.7.4 Case study: IIR filter implementation on TI 672x DSPs

In this section we will examine various IIR filter implementations as a specific example of the considerations that must be made when optimizing DSP code for the 672x.

The TI 67xx family of DSPs is notably different from some other typical DSP processors, such as the 56k and the Intel FPU, in that the TI DSP does not have an implicit higher-precision multiply-accumulate. It is of course capable of double precision accumulation, but this must be coded explicitly. In some ways, this is similar to the

Intel SSE processing unit, which jetisonned the 80-bit floating point stack used in the Intel FPU. The lack of higher precision accumulation in TI (and SSE) can sometimes result in unacceptable quantization noise performance for single precision filter implementations. Luckily, with the right choice of filter structure or coding for explicit double precision accumulation, excellent results can be achieved.

On fixed-point DSPs such as 56k, Direct Form I (DF1) implementation is the standard due to moderately good fixed point scaling properties, decent noise performance, and simple implementation. However, on a 672x DSP a single precision DF1 filter can have terrible noise performance (depending on the filter coefficients and the audio material being processed.) A degenerate case is a DF1 highpass filter processing low frequency material; in DF1, the feedforward coefficients subtract the previous sample from the current sample, and for low frequency material this produces very small numbers with low precision. Single precision DF2 structures also produce similarly poor results in this respect.

One option to improve upon these results is to use double precision throughout the 672x filter implementation. However, this results in a heavy cycle performance penalty due to the high cost of double operations on the TI DSP. Another, often better, option is to use single precision coefficients and state, with double precision accumulation:

```
float in, b0, b1, a1, state1;
double accum ;
accum = double (b0) * double (in) +
       double (b1) * double (state1) +
       double (a1) * double (accum);
state1 = in;
```

Listing 11: Mixed-precision DF1 filter implementation

The TI compiler will implement this using the mpypsp2dp instruction, since it knows that the operands started out as single precision and end up as double precision. This is considerably faster than going to a full double precision implementation, but it is still relatively slow compared to straight single precision. Making the state double precision will improve noise performance further, with some increase in cycle usage.

Another option that generally gets good results is the single precision DF2 Transpose (DF2T) filter. On TI the DF2T implementation is fast and generally has good noise performance. If you are looking for a simple recommendation that should work well enough for most applications, DF2T is a good choice.

The optimized C filter library available from TI uses the DF2 structure in its implementation. Even though DF2 has some limitations, this is a good starting point for seeing how to optimize filter code on TI; peak performance on TI is 2.25 cycles per biquad, so it's pretty amazing what can be done (to achieve that level of performance multiple series or parallel biquads need be put in a tight loop.) We have adapted some of this filter code to DF2T, and still achieved fairly similar cycle performance.

If the single precision DF2T noise performance is not good enough for your application, then either double precision or one of the myriad other filter structures, such as State Space, Gold-Rader, Lattice or Zolzer, should do the job. In fact, there is one relatively new filter structure which we think stands out, called the Direct Wave Form (DWF) filter. Details about this filter structure can be found in *Direct Wave Form Digital Filter Structure: an Easy Alternative for the Direct Form* by Jean H.F. Ritzerfel. According to the author the noise performance is 3dB within optimal, it's relatively efficient (5 multiplies per biquad), free of limit cycles, has simple coefficient generation and low coefficient quantization sensitivity. It might just be the perfect filter structure, but we'll let you be the judge of that; keep in mind that all filter structures have some tradeoffs, and the recommendations made here might not be the best for your particular application.

12.40.7.5 Understanding CGTools-generated ASM files

The ability to read the ASM files that are generated by CGTools is essential when optimizing a TI algorithm. Specifically, the information in these files will allow you to determine if anything is preventing software pipelining from occurring, which is the single most effective form of optimization on the C6727.

To view your project's ASM file, turn on the `-k` compiler option ("Keep Generated .asm Files", found under Build Options > Compiler > Assembly in the Code Composer Studio IDE.) By default, ASM files will be placed in the same directory as the corresponding source file.

Note

You should only examine ASM listings of Release code that has been optimized by the compiler. Debug code should not be optimized.

Each ASM file for a TI algorithm callback should contain text that marks the start of the assembly listing for the processing loop. For example:

```
;*****  
;* FUNCTION NAME: // [Your algorithm's ProcessProc symbol] _____*  
;*  
;* Regs Modified: A0,A1,A2,A3,A4,A5,A6,A7,A8,A9,A10,A11,A12,A13,A14, _*  
;* _____ A15,B0,B1,B2,B3,B4,B5,B6,B7,B8,B9,B10,B11,B12, _____*  
;* _____ B13,SP,A16,A17,A18,A19,A20,A21,A22,A23,A24,A25, _____*  
;* _____ A26,A27,A28,A29,A30,A31,B16,B17,B18,B19,B20,B21, _____*  
;* _____ B22,B23,B24,B25,B26,B27,B28,B29,B30, B31 _____*  
;* Regs Used: A0,A1,A2,A3,A4,A5,A6,A7,A8,A9,A10,A11,A12,A13,A14, _*  
;* _____ A15,B0,B1,B2,B3,B4,B5,B6,B7,B8,B9,B10,B11,B12, _____*  
;* _____ B13,DP,SP,A16,A17,A18,A19,A20,A21,A22,A23,A24, _____*  
;* _____ A25,A26,A27,A28,A29,A30,A31,B16,B17,B18,B19,B20, _____*  
;* _____ B21,B22,B23,B24,B25,B26,B27,B28,B29,B30,B31 _____*  
;* Local Frame Size: 0 Args + 148 Auto + 44 Save = 192 byte _____*  
;*****
```

Listing 12: CGTools-generated header for a processing loop assembly listing

Within this listing, you are looking for several things:

1. Function calls
2. Branches or control code
3. Software pipelining notes

12.40.7.5.1 Function calls

```
[ !B0] CALL .S1 __divd ; |213|  
|| [ !B0] MVKH .S2 0x40080000 ,B5 ; |213|  
|| [ B0] MV .L1X B10 ,A4 ; |213|  
$C$RL9 : ; CALL OCCURS {__divd} ; |213|
```

Listing 13: Function call in a CGTools-generated assembly listing

Function calls, such as the call in the listing above, cannot be effectively pipelined. If you find a function call figure out what C instruction it is caused by. Sometimes a function call will be made implicitly, such as when casting from float to int or when doing division. All function calls should be removed from the processing loop or inlined in order for the compiler to optimize effectively.

12.40.7.5.2 Branches

```
NOP 1  
B .S1 $C$L5 ; |213|  
NOP 4  
MPYDP .M1X A5:A4 ,B5:B4 ,A11:A10 ; |213|  
|| LDW .D2T2 *+ SP (124) ,B5 ; |218|  
; BRANCH OCCURS { $C$L5 } ; |213|
```

Listing 14: Branch in a CGTools-generated assembly listing

Branches can also prevent loop pipelining. If you find a branch in your algorithm's assembly, determine whether it is preventing the compiler from pipelining a loop. If it is preventing pipelining, you must figure out how to rewrite the conditional in your C code so that it will not be compiled into a branch.

12.40.7.5.3 Software pipelining notes

For each loop the compiler finds and is able to pipeline, the .ASM file should contain a section similar to the one below:

```

;-----*
;* SOFTWARE PIPELINE INFORMATION
;*
;* Loop source line : 68
;* Loop opening brace source line : 69
;* Loop closing brace source line : 124
;* Loop Unroll Multiple : 2x
;* Known Minimum Trip Count : 1
;* Known Max Trip Count Factor : 1
;* Loop Carried Dependency Bound (^) : 15
;* Unpartitioned Resource Bound : 20
;* Partitioned Resource Bound (*) : 20
;* Resource Partition :
;* A- side B- side
;* .L units 0 0
;* .S units 0 1
;* .D units 20* 20*
;* .M units 7 5
;* .X cross paths 5 6
;* .T address paths 20* 20*
;* Long read paths 5 1
;* Long write paths 0 0
;* Logical ops (. LS) 5 4 (.L or .S unit )
;* Addition ops (. LSD) 0 1 (.L or .S or .D unit )
;* Bound (.L .S .LS) 3 3
;* Bound (.L .S .D .LS .LSD) 9 9
;*
;* Searching for software pipeline schedule at ...
;* ii = 20 Schedule found with 3 iterations in parallel

```

Listing 15: Pipelined loop header in a CGTools-generated assembly listing

These are the important items to note in this listing:

- Loop Carried Dependency Bound and Partitioned Resource Bound The maximum of these numbers is the minimum number of clock cycles one instance of the loop will require in its current form. You can reduce these numbers by performing some of the optimizations listed in this guide.
- Loop Unroll Multiple This line will appear if the compiler is partially unrolling the loop to improve performance.

If a loop section instead displays `Disqualified loop`: then some of the conditions required to enable software pipelining have not been met:

- `-O2` or `-O3` optimizations must be enabled
- The loop cannot contain a function call. Make all called functions inline.
- The loop cannot contain any branches or jumps, often caused by large conditional statements
- Software pipelining will not work with nested loops; only the innermost loop will be pipelined. You should completely unroll the inner loop or refactor the algorithm so that the loop can be pipelined

For more information about pipelining and loop/branch optimization, see [Refactoring conditionals and branches](#).

12.40.7.6 C keywords

There are a few keywords in C that give the compiler additional information about the variables you declare and parameters you pass into functions. This allows the compiler to further optimize the code it is compiling, which can result in significant performance gains.

12.40.7.6.1 const

Effective use of `const` lets the compiler know whether pointers, scalars, or objects will remain constant in memory.

Let's add the `const` keyword to the filter function from our example of [The load-update-store pattern](#).

```

void
ProcessDirectFormII (
    const float * const input, // read - only
    float * const output, // read - write
    float * const state, // read - write
    const float * const coefs , // read - only
    int nsamp )
{
    // eB0 .. eB2 and eA0, eA1 are just integer enums to partition
    // the filter coefficients into A and B

    // ---- LOAD ----
    const float coefA0 = coefs [ eA0 ];
    const float coefA1 = coefs [ eA1 ];
    const float coefB0 = coefs [ eB0 ];
    const float coefB1 = coefs [ eB1 ];
    const float coefB2 = coefs [ eB2 ];

    float state0 = state [0];
    float state1 = state [1];

    // ---- UPDATE ----
    for (int i =0; i < nsamp; ++i)
    {
        const float output = input [i]* coefB0 + state0 ;
        state0 = input [i]* coefB1 + state1 - output * coefA0 ;
        state1 = input [i]* coefB2 - output * coefA1;
        output [i] = output;
    }

    // ---- STORE ----
    state [0] = state0;
    state [1] = state1;
}

```

Listing 16: Refactored filter algorithm with load-update-store pattern and `const` keyword applied.

It is especially important to note that the declaration of `const float output` was moved inside the loop. Why did we do this? Because we see that output is constant over an iteration of the loop, but it does change between iterations. By declaring it `const` inside the loop body we remove the data dependency that existed in output and allow the loop to optimize more effectively.

As demonstrated by this change to `const float output`, `const` is useful for manually breaking dependencies in DSP code. Variable re-use introduces unnecessary data dependencies in code, which can be avoided by using individual local `const` variables.

12.40.7.6.2 restrict

The `restrict` keyword tells the compiler that a specific pointer is not aliased, meaning that none of the memory locations accessed by the pointer are read or written to by any other variable within its local scope. This keyword is very important when optimizing TI code that involves pointers, as all AAX algorithms do due to the nature of the algorithm context structure.

`restrict` was introduced with the C99 standard. AAX plug-ins use the `AAX_RESTRICT` keyword, which is a cross-platform macro for the C99 standard `restrict`.

Note

Now that MSVC has added C99 support to its compiler, `AAX_RESTRICT` will eventually be deprecated in favor of the `restrict` keyword.

The following example demonstrates the use of `restrict` in our filter code.

```

void
ProcessDirectFormII (
    const float * const AAX_RESTRICT input,
    float * const AAX_RESTRICT output,
    float * const AAX_RESTRICT state,
    const float * const AAX_RESTRICT coefs ,
    int nsamp )
{
    // eB0 .. eB2 and eA0, eA1 are just integer enums to partition
    // the filter coefficients into A and B

    // ---- LOAD ----

```

```

const float coefA0 = coefs [ eA0 ];
const float coefA1 = coefs [ eA1 ];
const float coefB0 = coefs [ eB0 ];
const float coefB1 = coefs [ eB1 ];
const float coefB2 = coefs [ eB2 ];

float state0 = state [0];
float state1 = state [1];

// ---- UPDATE ----
for (int i =0; i < nsamp; ++i)
{
    const float output = input [i]* coefB0 + state0;
    state0 = input [i]* coefB1 + state1 - output * coefA0;
    state1 = input [i]* coefB2 - output * coefA1;
    output [i] = output;
}

// ---- STORE ----
state [0] = state0;
state [1] = state1;
}

```

Listing 17: Refactored filter algorithm with load-update-store pattern and `const` and `restrict` keywords applied.

Note

- This example applies `restrict` to the algorithm's input and output audio buffer pointers. These pointers do not alias each other in most algorithms, but this may not be the case for all algorithms and should be verified by the developer before applying `restrict`.
- The `restrict` keyword is somewhat redundant when used with the load-update-store pattern. This is because by asserting to the compiler that the pointers are not aliased, it should be able to partially do the load-update-store refactoring automatically. However, because some compilers have limited or no support for the `restrict` keyword, using the load-update-store pattern is still recommended.

12.40.7.6.3 Keywords to avoid

There are some keywords which do more harm than good, but are still being used either due to legacy code or developer superstitions. These keywords should not be used in AAX plug-ins.

- `register` The `register` keyword is a suggestion to the compiler that a certain variable will be accessed frequently and should be stored in a register rather than a memory location. Use this keyword only when you are sure that the compiler is placing a frequently-used variable in memory when it would be advantageous to keep it in a register. Note that the `register` keyword has no effect if the CGTools optimizations are enabled.
- `static` In C, the `static` keyword tells the compiler to initialize the variable at compilation time and retain the value between calls. Though there are some valid situations to use the `static` keyword, its use in AAX plug-ins on all platforms is extremely limited. One of its most "popular" uses, declaring local variables inside a function as `static` in order to achieve a type of global counter, should never be used in [AAX](#) algorithm code. If you are using `static` to make a local variable hold its variable across calls to a function, it is always preferable to either pass it in to the function as a modifiable parameter or declare it as a member variable of the method (if C++).

12.40.7.7 Data types

The TI C672x+ is a 32-bit floating point DSP platform, and has a few peculiarities that you should be aware of.

- Use `int` instead of `long` Integers of type `long int` are 40 bits wide on TI, and are very inefficient. Always use the `int` data type (or, even better, the C99-standard `int32_t`) instead.
- Use `float` instead of `double` Double-precision floating-point data types have a significant performance penalty on TI processors. Use `float` instead of `double` wherever possible, as long as this substitution does not affect signal integrity or cancellation.

- Use unsigned values when referencing memory. In general, explicitly typed pointers should always be used to reference memory. If you do have need of a generic memory representation, use an unsigned integer to avoid implicit conversion costs.

12.40.7.7.1 Unintended data type conversions

When developing for the TI platform it is important to keep an eye out for unintended type conversions, and especially for implicit double-precision instructions. The following points are helpful for both program efficiency and for future maintenance of the code, since they clarify the developer's understanding of how the code should operate, e.g. by specifying that a cast is occurring, and make it obvious that steps such as data type conversions are an intentional part of the algorithm.

- Explicitly declare constants as single-precision. For example, use `0.0f` instead of `0.0`. Often a compiler will be able to do this automatically at compile time, but it is better to be explicit with your intended precision.
- If any casts are required in your code, make them explicit. For example, `float output = (float)doubleVar` as opposed to `float output = doubleVar`.
- Use single-precision math.h functions (such as `fabsf()`) instead of the double-precision equivalents (`fabs()`).
- Do not directly reference memory addresses using integer data types; instead, use a pointer data type. If an integer data type is required, use an unsigned 32-bit type.

To help ensure that you are not violating these principles, always be aware of any warnings generated by the compiler. In particular, do not ignore warnings related to "implicit conversion from 'double' to 'float'" or "implicit conversion from 'double' to 'int"'; these warnings may indicate that you are declaring a double when a float would be just as good.

In the final stages of optimization, examine the generated assembly code to make sure there are no unintended double-precision instructions or memory accesses.

12.40.7.7.2 Additional data type optimizations

The AAX SDK includes cross-platform macros that can be used to convert two single-precision float loads to one double-precision load. The coefficient smoothing case study below includes an example use case for these macros.

```
const float * pTable = &SmoothCoefTable[address];
AAX_ALIGNMENT_HINT(pTable,8);
float firstCoef = AAX_LO(*pTable);
float secondCoef = AAX_HI(*pTable);
```

Listing 18: Example of using AAX macros for converting two `float` loads to one `double` load.

In this example the `AAX_ALIGNMENT_HINT` macro checks whether data is aligned on a 8-byte boundary, then the double word is loaded, and finally the `AAX_LO` and `AAX_HI` macros get the double word's first and second (`float`) parts.

If `SmoothCoefTable` consists of floats and is 8-byte aligned, then this scenario will work fine for loads when `address` is even. This raises the question about how to load double word from `&SmoothCoefTable[address]`, when `address` is odd. Since this kind of optimization is most useful for loading data from external memory, where the CPU savings of a single double word load vs two 32-bit loads is greatest, then one trick which can help is to trade off memory (as external memory is plentiful) for performance. Specifically, `SmoothCoefTable` can be organized in a such way that for every member of this table, except the first and the last ones, there will be two consequent entries.

```
const int32_t size = 4;
// instead of this classic variant...
const float SmoothCoefTable[size] = {
    -0.1, -0.2, -0.3, -0.4
}

// ...table can be organized this way
const float SmoothCoefTable[size*2 - 2] = {
    -0.1, -0.2,
```

```

-0.2, -0.3,
-0.3, -0.4,
-0.4,  0.0 /* last member is dummy */
}

```

Listing 19: Example of restructuring the table so that it can be easily used in the optimization scenario given above.

In this case the number of loads will be halved at the cost of doubling the size of the table. If the table is located in external memory then the additional memory requirement can be an excellent trade-off for the performance gained.

12.40.7.8 Case study: Efficient parameter smoothing at single and double precision

Coefficient smoothing ("de-zippering") can often be one of the most difficult parts of a plug-in to optimize for real-time operation. This is especially true in cases when full double-precision smoothing filters have been used in a plug-in's Native code, with the possibility of very small coefficients. In these cases it can be difficult to optimize the smoothing code while also satisfying requirements for audio data parity between the plug-in's Native and DSP configurations.

```

double * const AAX_RESTRICT deZipper = dzCoefsP->mDeZip [ch ][0];
const double * AAX_RESTRICT coefs = myCoefsP->mBiqCoefsBuf [0];

// Double - precision
for (int i = 0; i < eNumBiquads * eNumCoefs ; ++i)
{
    double dz = deZipper [i];
    dz += zeroCoef * ( coefs [i] - deZipper [i]);
}

```

Listing 20: Example of double-precision smoothing.

In this section we will describe three specific approaches that may be taken to perform optimized real-time smoothing without compromising sound quality.

12.40.7.8.1 Method 1: Clamped single-precision smoothing

The simplest approach for optimization of a double-precision smoothing filter is to replace it with modified single-precision smoothing. Unfortunately, we have found that this approach can lead to glitches and instability at higher sample rates when adjusting controls due to transient inaccuracies in the smoothing.

```

double * const AAX_RESTRICT deZipper = dzCoefsP->mDeZip [ch ][0];
const double * AAX_RESTRICT coefs = myCoefsP->mBiqCoefsBuf [0];

// Method 1 - single - precision
for (int i = 0; i < eNumBiquads * eNumCoefs ; ++i)
{
    float dz = deZipper [i];
    dz += zeroCoef * ( coefs [i] - deZipper [i]);

    // If the de -zip step is so small that the coefficient doesn't change then clamp
    // the value to the target to ensure we are using exactly the desired value .
    deZipper [i] = (dz == deZipper [i]) ? coefs [i] : dz;
}

```

Listing 21: Example of clamped single-precision smoothing.

12.40.7.8.2 Method 2: Mixed-precision smoothing

To resolve the stability issues at high sample rates, the state may be accumulated at double-precision. This results in mixed-precision operations that are much faster on TI DSPs than full double-precision calculations, though still slower than single-precision.

```

float * const AAX_RESTRICT deZipper = dzCoefsP->mDeZip [ch ][0];
double * const AAX_RESTRICT deZipState = dzCoefsP->mDZState [ch ][0];
const float * AAX_RESTRICT coefs = myCoefsP->mBiqCoefsBuf [0];

// Method 2 - partial double precision

```

```
# pragma UNROLL ( CBiquad::eNumCoefs )
for(int i = 0; i < eNumBiquads * eNumCoefs ; i++)
{
    double dz = deZipState [i];
    dz += zeroCoef * ((coefs [i]) - (deZipper [i]));
    deZipState [i] = dz;
    deZipper [i] = float (dz);
}
```

Listing 22: Example of mixed-precision smoothing.

12.40.7.8.3 Method 3: Loop unrolling and double-word memory accesses

Further performance gains can be made by unrolling the loop and using double word memory accesses. This code is faster, but is still not as fast as full single-precision.

```
float * const AAX_RESTRICT deZipper = dzCoefsP->mDeZip [ch][0];
double * const AAX_RESTRICT deZipState = dzCoefsP->mDZState [ch][0];
const float * AAX_RESTRICT coefs = myCoefsP->mBigCoefsBuf [0];

// Method 3 - partial double precision - unrolled with double-precision memory accesses for(int i = 0; i <
// (eNumBiquads * eNumCoefs); i +=2 )
{
    double dz0 = deZipState [i];
    double dz1 = deZipState [i+1];
    dz0 += zeroCoef * (AAX_LO ( coefs [i]) - AAX_LO ( deZipper [i]));
    dz1 += zeroCoef * (AAX_HI ( coefs [i]) - AAX_HI ( deZipper [i]));
    deZipState [i] = dz0;
    deZipper [i] = float (dz0);
    deZipState [i+1] = dz1;
    deZipper [i+1] = float (dz1);
}
```

Listing 23: Example of loop unrolling and double-precision memory accesses for smoothing optimization.

12.40.7.8.4 Coefficient smoothing example summary

- Full single-precision smoothing (method 1) is an excellent and simple solution for gain coefficients and other scalar values which are not extremely sensitive to coefficient quantization at small values. This method does not always reach the target value, so clamping should be used to ensure signal integrity.
- Mixed-precision smoothing (method 2) uses slightly more CPU, but gives full double precision accuracy. This approach should generally be used for EQs and other sensitive coefficients.
- Further low-level optimizations are also possible via manual loop unrolling and double-precision memory access (method 3).

12.40.7.9 Refactoring conditionals and branches

Note

For more detailed information on how to reduce or eliminate the use of branches in algorithms, see section 5.2 of the **Hand-Tuning Loops and Control Code on the TMS320C6000** guide provided by TI.

An important technique in refactoring algorithms to enhance loop performance is to reduce or eliminate conditionals and branches in code. The TI compiler focuses a lot of its optimization energy on keeping its pipeline full of inside loops. However, it cannot pipeline a loop if the one of the following is true:

- The loop contains a branch
- The loop contains a function call
- The loop is too long

To demonstrate this, we will again begin with an unoptimized example:

```

for ( int i = 0; i < numSamples ; ++i)
{
    if ( ! bypass )
    {
        const float filtOutput1 = input [i] * coef0 + state0 * coef1 ;
        const float filtOutput2 = filtOutput1 * coef2 + state1 * coef3 ;
        output [i] = filtOutput2 ;
    }
    else
    {
        output [i] = input [i];
    }
}

```

Listing 24: Another unoptimized filter algorithm.

Though trivial, this example illustrates the problem with conditionals inside of loops. In TI assembly, conditional code usually translates into code branches, which prevents loops from pipelining effectively see [Understanding CGTools-generated ASM files](#). Let's refactor the loop in our example to reduce the size of its conditional branch:

```

for (int i = 0; i < numSamples ; ++i)
{
    const float filtOutput1 = input [i] * coef0 + state0 * coef1 ;
    const float filtOutput2 = filtOutput1 * coef2 + state1 * coef3 ;
    output [i] = filtOutput2 ;

    if ( bypass )
    {
        output [i] = input [i];
    }
}

```

Listing 25: Filter algorithm with a refactored conditional branch.

At first, it may seem wasteful to perform the filter calculation if `bypass` will simply throw away the result. In reality, however, the opposite is true: as a real-time algorithm, this code is constrained by its maximum, worst-case cycle count. It is important to understand this point: essentially, the cycle count of the plug-in is always its worst-case performance.

By reducing the algorithm's maximum cycle count we are therefore reducing waste, even though we are increasing the plug-in's cycle count when it is bypassed. In fact, the ideal scenario for most algorithms is to use only one code path (and, consequentially, a single deterministic cycle count) despite the fact that this can result in worse performance for some specific states. To state this fundamental principle in a different way:

The performance of specific states in an AAX DSP algorithm is not relevant if there is another possible state with worse performance.

Going back to our optimized example, you may also notice that the conditional still exists. Doesn't this create a branch in the assembly code as well and prevent pipelining?

In the case of very brief conditionals such as this, the answer is usually no. On TI processors, most instructions can be executed conditionally, depending on the value of a control register. Thus, the single assignment (`output = input`) inside this conditional will reduce to a few conditional instructions without having to execute a branch. As a result, the TI compiler will be able to efficiently pipeline this loop.

That said, it is occasionally necessary to eliminate conditionals entirely. One effective solution for these situations is to execute the branched logic algorithmically rather than conditionally. To demonstrate this approach, here is our filter example again, this time with the the conditional completely eliminated from the loop:

```

for (int i = 0; i < numSamples ; ++i)
{
    const float filtOutput1 = input [i] * coef0 + state0 * coef1 ;
    const float filtOutput2 = filtOutput1 * coef2 + state1 * coef3 ;
    output [i] = (! bypass ) * filtOutput2 + bypass * input [i];
}

```

Listing 26: Filter algorithm with branching logic executed algorithmically.

This code is shorter and completely eliminates the conditional from inside the loop body. However, there is an associated cost in readability, in that it is not initially obvious how exactly `bypass` affects the output. This is of course a tradeoff that you will need to consider on a case-by-case basis. In general, we encourage you to consider

this technique only when you have verified in the assembly code that simply reducing the size of the conditional is not enough to achieve effective instruction pipelining.

Another useful technique for optimizing loops is to use `pragma MUST_ITERATE` and `pragma PROB_ITERATE` (see more about these pragmas in [Loop controls](#)), which help the compiler guess the number of iterations for the loop. It is extremely useful when you know the exact number of the iterations, and this number never changes during plug-in processing. For example, this is applicable for the loops which iterate through the audio samples in the input and output buffers. The number of input samples is always constant for an AAX DSP plug-in algorithm; the buffer length must be described with the option `AAX_eProperty_DSP_AudioBufferLength` for each DSP component in the plug-in's description.

The following code example shows an algorithm processing function template. For convenience, this function template takes the audio buffer length as a template parameter:

```
template<int kAudioWindowSize>
void AAX_CALLBACK
Example_AlgorithmProcessFunction( SExample_Alg_Context * const inInstancesBegin [], const void *
    inInstancesEnd)
{
    for (SExample_Alg_Context * const * walk = inInstancesBegin; walk != inInstancesEnd; ++walk)
    {
        SExample_Alg_Context* const AAX_RESTRICT contextP = *walk;
        const float * const AAX_RESTRICT inputP = contextP->mInputPP;
        float * const AAX_RESTRICT outputP = contextP->mOutputPP;

        #pragma MUST_ITERATE( kAudioWindowSize, kAudioWindowSize, kAudioWindowSize )
        for (int32_t i = 0; i < kAudioWindowSize; ++i)
        {
            outputP[i] = inputP[i];
        }
    }
}
```

Listing 27: Optimizing loop using pragma `MUST_ITERATE`.

Note that the audio buffer length property takes a `AAX_EAudioBufferLengthDSP` value. The values of this enum are set to the power-of-two for each buffer length, so in this case the `kAudioWindowSize` value would be set to match `2 << AAX_eProperty_DSP_AudioBufferLength` when compiling this algorithm callback into the TI DLL.

The same optimization can be used for the loops that iterate through input/output channels, as demonstrated by the `DemoDist` example plug-in.

12.40.7.10 Case study: pipeline refactoring in Avid's EQ3 and Dyn3 plug-ins

While optimizing the "stock" Pro Tools equalization and dynamics processors we came across many real-world optimization scenarios that will be applicable to a broad variety of plug-ins. In this section we will consider specific techniques that we used to enable software pipelining of these algorithms by the TI compiler, including an in-depth look at the pseudo-speculative execution approach used in our Dyn3 plug-in's polynomial gain calculation loop.

12.40.7.10.1 Move individual processing operations into separate loops

Oftentimes a sample-by-sample iterative loop that is not software pipelining can be broken up into individual loops that incrementally apply changes to the audio buffer. These smaller loops have a much better chance of being successfully pipelined by the compiler. In EQ3, moving our biquad audio processing stages to dedicated loops that do not include coefficient smoothing or other tasks resulted in large performance gains.

12.40.7.10.2 Avoid pipeline dependencies

The goal of the above optimization is to allow the compiler to successfully pipeline each iterative loop. However, even a pipelined loop may be optimized further. One of the best ways of optimizing loops is to keep the processor busy while pipeline dependencies are cleared.

For example, in EQ3 we found that it was better to perform the plug-in's input and output meter calculations in the same loop rather than separating them out into individual loops. This is because each meter calculation has a dependency on its previous value, which puts a dependency in the pipeline. Doing both at the same time gives the

process more to do while waiting for the next value. In Dyn3 we had similar results merging table lookup, attack, and release loops into a single iterative loop. As long as the loop is still successfully pipelined by the compiler, these "larger" loops tended to have much better performance due to the reduction in blocking dependencies.

12.40.7.10.3 Detailed example of loop optimization in Dyn3

At this point it will be helpful to go into greater detail about our optimizations for Dyn3's polynomial gain calculation loop, because the increase in performance was quite large and is fairly representative of other algorithms. The unoptimized code took 43 cycles to execute one iteration of the loop. After rearranging the code it now takes 6 cycles. The basic problem was numerous pipeline dependencies: the *Loop Carried Dependency Bound* was 42 cycles, yet the *Partitioned Resource Bound* was 4 cycles. In other words, if all of these dependencies were removed the loop could potentially execute in 4 cycles.

```

2760 ;* SOFTWARE PIPELINE INFORMATION
2761 ;*
2762 ;* Loop source line : 199
2763 ;* Loop opening brace source line : 200
2764 ;* Loop closing brace source line : 213
2765 ;* Known Minimum Trip Count : 4
2766 ;* Loop Carried Dependency Bound (^) : 42
2767 ;* Unpartitioned Resource Bound : 4
2768 ;* Partitioned Resource Bound (*) : 4
2769 ;*
2770 ;* Searching for software pipeline schedule at ...
2771 ;* ii = 42 Did not find schedule
2772 ;* ii = 43 Schedule found with 1 iterations in parallel
2773 ;* Done

for (int i =0; i< kAudioWindowSize ; i++) // cSmoothingBlockSize
{
    const float * smoothCoeffs = stateP -&t; mSmoothedPoly ;
    float logEnv = logEnvArray [i]; // logEnvArray [ fIdx +i];
    logEnv -= smoothThrLow ;

    if( logEnv >= 0.0 f) // In the knee
        smoothCoeffs += eCpdPolyOrder ;
    if( logEnv >= 0.0 f) // In the knee
        logEnv -= smoothThrLowDelta ;
    if( logEnv >= 0.0 f) // In the linear GR stage
        smoothCoeffs += eCpdPolyOrder ;

    const float filteredLogEnv = smoothCoeffs [ eCpdPolyCoeffsC ] +
        logEnv *( smoothCoeffs [ eCpdPolyCoeffsB ] +
        smoothCoeffs [ eCpdPolyCoeffsA ] * logEnv );
    filtLogEnvArray [i] = filteredLogEnv + smoothedMakeupGain ;
}

```

Listing 28: Dyn3's unoptimized polynomial gain calculation loop and asm listing.

- `logEnv -= smoothThrLow` *depends on the result of logEnvArray[i]*
- `if(logEnv >= 0.0f)` *depends on the result of logEnv -= smoothThrLow*
- `logEnv -= smoothThrLowDelta` *depends on the result of logEnv -= smoothThrLow*
- Thrid `if(logEnv >= 0.0f)` *depends on the result of logEnv -= smoothThrLowDelta*
- Second `smoothCoeffs += eCpdPolyOrder` *depends on the result of the first smoothCoeffs += eCpdPolyOrder*
- `logEnv*smoothCoeffs [eCpdPolyCoeffsB]` *depends on the result of logEnv -= smoothThrLowDelta*
- `smoothCoeffs [eCpdPolyCoeffs], etc.` *depend on the result of the second smoothCoeffs += eCpdPolyOrder*
- `filteredLogEnv+smoothedMakeupGain` *depends on the result of filteredLogEnv = smoothCoeffs [eCpdPolyCoeffsC]*
- `filtLogEnvArray[i]` *depends on the result of filteredLogEnv + smoothedMakeupGain*

And I don't think that even covers every case, but you get the idea. The bottom line is there is no way this loop can pipeline well. In contrast, here is the optimized code and listing file output once these dependencies have been removed:

```

2476 ;* Loop opening brace source line : 167
2477 ;* Loop closing brace source line : 179
2446 ;* Known Minimum Trip Count : 4
2482 ;* Loop Carried Dependency Bound (^) : 1
2483 ;* Unpartitioned Resource Bound : 4
2484 ;* Partitioned Resource Bound (*) : 4
2512 ;* ii = 6 Schedule found with 5 iterations in parallel

for (int i = 0; i < cProcessingBlockSize ; i++)
{
    float logEnv = logEnvArray [i];
    float logEnvThrHi = logEnv - smoothThrHigh ;
    const float gainSlope = smoothThrSlope +
        logEnv * smoothSlope ;
    const float gainKnee = smoothKneeC +
        logEnvThrHi *( smoothKneeB +
        smoothKneeA * logEnvThrHi );

    const bool bKnee = ( logEnv > smoothThrLow );
    const bool bSlope = ( logEnv > smoothThrHigh );

    float filteredLogEnv = bKnee ? gainKnee : 0.0f;
    filteredLogEnv = bSlope ? gainSlope : filteredLogEnv ;
    filtLogEnvArray [i] = filteredLogEnv ;
}

```

Listing 29: Dyn3's optimized polynomial gain calculation loop and asm listing

In this case gainSlope is only dependent on the loading of logEnv, so that can begin almost immediately. GainKnee must wait for logEnvThrHi, but gainSlope can be calculated during that time. bKnee and bSlope are also only dependent on logEnv, and start right away. The main dependency is filteredLogEnv which is dependent on bKnee and gainKnee and then bSlope and gainSlope. Anyhow, this is far fewer dependencies. Here is another version which runs in exactly the same number of cycles. (In fact, under the hood it may be creating the same asm code; we have not compared instruction-by-instruction.)

```

for (int i = 0; i < kAudioWindowSize ; i++)
{
    float logEnv = logEnvArray [i];
    float logEnvThrHi = logEnv - smoothThrHigh ;

    const bool bKnee = ( logEnv > thrLow );
    const bool bSlope = ( logEnv > thrHigh );

    float filteredLogEnv = bKnee ?
        kneeC + logEnvThrHi *( kneeB + kneeA * logEnvThrHi ) :
        0.0 f;
    filteredLogEnv = bSlope ?
        thrSlope + logEnv * slope :
        filteredLogEnv ;
    filtLogEnvArray [i] = filteredLogEnv ;
}

```

Listing 30: An alternative optimization for Dyn3's polynomial gain calculation loop.

12.40.7.10.4 But what about Native?

You might expect this altered code to execute well on a TI DSP but poorly on x86. However, keep in mind that a large degree of speculative execution is used on Intel's processors. This means that pipeline dependencies due to conditionals can be broken because multiple paths are executed. In these cases, only one of the results is used and the others are thrown away. In other words, if you saw pseudo code showing the literal execution of the unoptimized code above on Intel then it would probably look a lot like the optimized code. The lesson? For TI it is important to rearrange your code so that essentially it implements speculative execution as much as possible, and if applied correctly this optimization should not negatively impact your plug-in's native performance.

12.40.7.11 Case study: Additional optimization lessons from EQ3 and Dyn3

The pipeline optimization example above is just one example, and the following techniques also helped us achieve many-fold increases in performance. Note that many of these techniques are discussed in greater detail in the

sections above.

12.40.7.11.1 Watch the assembly listing

In the process of optimizing these plug-ins we found their asm listing files very helpful, especially the *Loop Carried Dependency Bound* and the *Partitioned Resource Bound* information. The listing file shows how many cycles the code is taking to execute, and we could make an estimate of how far away we were from the optimal implementation by seeing how well the pipeline is being utilized.

12.40.7.11.2 Divide processing tasks over multiple calls

In the old RTAS version of EQ3 the coefficients were updated (smoothed) every 8 samples. Initially, this was changed to every 4 samples in the AAX version in order to easily work with 4-sample blocks on HDX. However, we were able to achieve better results by adding "ping pong" logic that alternates between smoothing the first and second half of the coefficients on each pass. To make this work in our odd-banded EQ we had to pad the smoothing coefficients by one biquad's worth to make an even number of biquads, but regardless of this inefficiency we still achieved performance gains.

12.40.7.11.3 Eliminate branches that block pipelining

Eliminating large conditional branches is critical to optimal performance on TI. This can be an especially tempting pitfall for developers who are used to coding only for x86 processors.

Consider the "ping pong" optimization described above. This logic does not break pipelining because the conditional logic that checks the state of the flag does not result in a large branch; once the ping pong value is set, the exact same logic operates in every processing callback. If instead we used an if statement to determine which "side" should execute, this would prevent pipelining optimizations and would seriously impact performance.

12.40.7.11.4 Remove double-precision operations where they are not required

Here is some coefficient smoothing code from our pre-optimization EQ3 algorithm. This code was embedded in the inner biquad processing loop:

```
# pragma UNROLL ( CBiquad::eNumCoefs )
for (int k = 0; k < CBiquad::eNumCoefs; ++k)
{
    double &dz = deZipper[k];
    AAX::DeDenormal (dz);
    step[k] = zeroCoef * (coefs[k] - dz);
}

# pragma UNROLL ( CBiquad::eNumCoefs )
for(int k = 0; k < CBiquad::eNumCoefs; ++k)
{
    double nm1_dz = deZipper[k]; // read state
    nm1_dz += step[k];
    biquadCoefs[k] = static_cast< float > ( nm1_dz );
    deZipper[k] = nm1_dz ; // write state
}
```

Listing 31: Unoptimized coefficient smoothing in EQ3

To optimize this code, we converted the logic to use single-precision de-zipper values. However, this resulted in a sonic difference due to the fact that the smoothed coefficients would not necessarily ramp all the way to the correct target value. To solve that we added a conditional "clamp" that halts the smoothing once there is no difference between the 32-bit smoothed value and the target value. On examination of the assembler output, we found that this conditional pipelines very well.

```
# pragma UNROLL ( CBiquad::eNumCoefs )
for(int i = 0; i < (cMaxNumBiquadsWithPad / 2) * CBiquad::eNumCoefs; ++i)
{
    float dz = deZipper[i];
    dz += zeroCoef * (coefs[i] - deZipper[i]);
    deZipper[i] = (dz == deZipper[i]) ? coefs[i] : dz; // clamp
}
```

Listing 32: Optimized coefficient smoothing in EQ3

12.40.7.11.5 Make coefficients contiguous

We were able to achieve significant performance gains in iterative loops like the smoothing code shown above by ensuring that all of the coefficients that would be accessed by the loop are contiguous in memory. In addition, note that in the optimized code there is only one loop, which iterates `NumBiquads*NumCoefs` times. This optimization is possible due to the fact that each filter's coefficients are contiguous in the `coefs` array.

12.40.7.11.6 Use AAX_RESTRICT wherever applicable

We have found that the `restrict` keyword is vital for optimal performance on TI DSPs. For example, the parameter smoothing logic in our Dyn3 plug-in was reduced from 18 cycles to 3 cycles per loop iteration simply by the addition of this keyword to the applicable pointer variables.

For more information about the `restrict` keyword, see [restrict](#).

12.40.7.11.7 Be aware of shell overhead

In the TI Shell there is code that loops through every buffered coefficient FIFO before every sample buffer in order to swap the algorithm's context field pointers to a new set of coefficients if one is available. This uses a nominal number of cycles per buffered port, which can add up very quickly in small plug-ins.

For example, before our optimizations EQ3 used eight individual buffered coefficient blocks. On investigation, we found that the shell overhead from managing these buffers added up to be roughly equivalent to the algorithm's total processing cycles! To work around this we merged the 8 coefficient blocks into one large block. The trade-off of this optimization is that more work must be done on the host to re-generate and copy the whole coefficient state every time any parameter changes, so this is an optimization that should be applied only when appropriate for the individual plug-in. For example, before our optimizations EQ3 used eight individual buffered coefficient blocks. On investigation, we found that the shell overhead from managing these buffers added up to be roughly equivalent to the algorithm's total processing cycles! To work around this we merged the 8 coefficient blocks into one large block. The trade-off of this optimization is that more work must be done on the host to re-generate and copy the whole coefficient state every time any parameter changes, so this is an optimization that should be applied only when appropriate for the individual plug-in.

12.40.7.11.8 Watch for opportunities to merge or eliminate operations

Keep an eye out for unnecessary processing stages performed by your algorithm. Gain stages, phase toggles, and "dummy" coefficients are particularly good candidates for this kind of optimization. For example:

- In our EQ3 plug-in, we found that we could achieve significant performance improvement by merging the plug-in's input and output gain stages with the overall gain of the first and last biquads. As a side benefit, this reduced the total quantization noise in the algorithm.
- In our Dyn3 plug-in, we found that we were applying smoothing logic to filter coefficients that would always be zero.
- When we looked more closely at Dyn3 we found that we were also computing and discarding sidechain filter information for the LFE, which is not part of the sidechain

12.40.7.11.9 Read the TI documentation

There are many helpful optimization resources available from Texas Instruments. Out of all of the TI optimization documents we encountered, we found the *Hand-Tuning Loops and Control Code on the TMS320C6000* guide to be the most helpful and complete.

12.40.7.12 Optimization on the HDX platform

12.40.7.12.1 Interrupt latency

Besides the large latency due to context switching (lots of data file registers to store) and the pipeline (many stages), interrupts can be disabled around pipelined loops, which cannot be interrupted. This can be controlled with the

-mi=X compiler option, which will disallow unsafe pipelining for loops that are longer than X cycles. See TI's documentation (SPRU187O Section 2.12) for more details and references regarding this behavior.

12.40.7.12.2 External memory access

A loop which performs many reads and writes may require access to external memory. In this scenario, the loop may take 10's or even 100's of times longer to execute than the compiler expects it to!

There are two options for dealing with this:

1. Search and destroy these loops individually
 - Move all the data used by the loop to internal RAM.
 - Use HDX's DMA facilities for external memory accesses.
 - `#pragma FUNC_INTERRUPT_THRESHOLD` can be used to disable pipelining on a case by case basis.
2. For modules that are known to have these loops but are not worth hand optimizing, then turn off pipelined loop optimization altogether. (`-mu` aka `-disable_software_pipelining`).

Note

This is only a problem in the C67(0-2)x ISAx used on the HDX platform. In The C64xx and C674x ISA, there is an SPLOOP command which can buffer the branches within pipelined loops to allow them to be interruptable.

12.40.7.13 Code Composer Studio optimization tools

12.40.7.13.1 Compiler Consultant

The Compiler Consultant tool can be used to suggest additional optimizations.

To enable the Compiler Consultant in Code Composer Studio, do the following:

1. Set an optimization level of `-o2` or `-o3` (Found in CCSv4 under Build Options > Compiler > Basic)
2. Set the `-consultant`: Generate Compiler Consultant Advise switch (Found in CCSv4 under Build Options > Compiler > Feedback)

12.40.7.13.2 Optimization information file

Optimization information files can be generated in Code Composer Studio by selecting the option Build Options > Compiler > Feedback > Opt Info File. Optimization information files have an .nfo extension and are placed into the project's intermediate build products directory. In general, these files list function call-graph information and describe whether or not individual functions can be inlined.

12.40.8 Error Codes

The following appendices document error codes that are specific to plug-in hosting in Pro Tools HDX and other AAX platforms based on the TI DSP environment.

12.40.8.1 -138xx: DHM Core DSP errors

These errors relate to routing and assignment problems on Pro Tools HDX hardware. Plug-ins should never be able to trigger these error codes, which indicate low-level problems in the system.

Table 1: DHM Core DSP error codes

Value	Definition
-13801	ePSError_CTIIDSP_WrongSampleRate
-13802	ePSError_CTIIDSP_NoFreeStreams
-13803	ePSError_CTIIDSP_StreamCreation
-13804	ePSError_CTIIDSP_StreamDestruction
-13805	ePSError_CTIIDSP_InactiveStream
-13806	ePSError_CTIIDSP_StreamCorrupted
-13807	ePSError_CTIIDSP_QueueFull
-13808	ePSError_CTIIDSP_NullPointer
-13809	ePSError_CTIIDSP_WrongStreamID
-13810	ePSError_CTIIDSP_ImageError
-13811	ePSError_CTIIDSP_ResetError
-13812	ePSError_CTIIDSP_ImageVerify
-13813	ePSError_CTIIDSP_DSPAlreadyInBootOr
-13814	ePSError_CTIIDSP_TriggerInterrupt
-13815	ePSError_CTIIDSP_BufferSizeNot
-13816	ePSError_CTIIDSP_TimeoutWaitingForH
-13817	ePSError_CTIIDSP_SetUHPIError
-13818	ePSError_CTIIDSP_UHPINotReady

12.40.8.2 -140xx: AAX Host errors

These errors relate to logic failures in the AAX host software. These errors can be due to plug-in bugs or system configuration problems.

Table 2: AAX Host Software error codes

Value	Definition
-14001	kAAXH_Result_Warning
-14003	kAAXH_Result_UnsupportedPlatform
-14004	kAAXH_Result_EffectNotRegistered
-14005	kAAXH_Result_Incomplete
-14006	kAAXH_Result_InstantiationException
-14007	kAAXH_Result_UnknownException
-14008	kAAXH_Result_BadEffectComponents
-14009	kAAXH_Result_BadLegacyPlugInIDIndex
-14010	kAAXH_Result_EffectFactoryInitiated
-14011	kAAXH_Result_InstanceNotFoundWhen
-14012	kAAXH_Result_EffectFailedToRegister
-14013	kAAXH_Result_PlugInSignatureNot
-14014	kAAXH_Result_ExceptionDuring
-14015	kAAXH_Result_ShuffleCancelled
-14016	kAAXH_Result_NoPacketTarget
-14017	kAAXH_Result_ExceptionReconnecting
-14018	kAAXH_Result_EffectModuleCreation
-14019	kAAXH_Result_Accessing
-14020	kAAXH_Result_ComponentInstantiationPostponed

-14021	kAAXH_Result_FailedToRegister
-14022	KAAAXH_ResolveNameFailedToRegister
-14023	KAAAXH_ResolveWrongAddressAgainst
-14023	KAAAXH_PossibleSDKValueBaultAgainst
-14100*	KAAAXH_PossibleSDKValueIsArgumentValue
-14101*	kAAXH_Result_NameNotFoundInPage

*Overlaps with [-141xx: TI System errors](#) definitions

Table

12.40.8.3 -141xx: TI System errors

These errors relate to logic failures in the TI management software and generally indicate a failure in the HDX system services such as buffered message queues, context management, and callback timing.

Table 3: TI system error codes	
Value	Definition
-14101	eTISysErrorNotImpl
-14102	eTISysErrorMemory
-14103	eTISysErrorParam
-14104	eTISysErrorNull
-14105	eTISysErrorCommunication
-14106	eTISysErrorIllegalAccess
-14107	eTISysErrorDirectAccessOfFifo
-14108	ETISysErrorOutOfBounds
-14109	eTISysErrorPortTypeDoesNotSupport
-14110	ETISysAccessEIFOFull
-14111	eTISysErrorRPCTimeOutOnDSP
-14112	eTISysErrorShellMgrChip_SegsDont
-14113	ETISysAddrOnChipRPCNotRegistered
-14114	eTISysErrorUnexpectedBufferLength
-14115	eTISysErrorUnexpectedEntryPointName
-14116	eTISysErrorPortIDTooLargeFor
-14117	ETISysEBlocMixerDelayNotSupported
-14118	ETISysInnerShellFailedToStartUp
-14119	eTISysErrorUnexpectedCondition
-14120	eTISysErrorShellNotRunningWhen
-14121	ETISysEDrorFailedToCreateNewPI
-14122	ETISysEErorUnknownPIInstance
-14123	eTISysErrorTooManyInstancesFor
-14124	ETISysBufdeneBBressing
-14125	eTISysBadDSPID
-14126	eTISysBadPIContextWriteBlockSize
-14128	eTISysInstanceInitFailed
-14129	eTISysSameModuleLoadedTwiceOnSame
-14130	ETISysCouldNotOpenPlugInModule
-14130	eTISysCouldNotOpenPlugInModule
-14131	eTISysPlugInModuleMissingDependcies

-14132	eTISysPlugInModuleLoadableSegment←
-14133	eTISysMissingModuleLoadFailure
-14134	eTISysOutOfOnChipDebuggingSpace
-14135	eTISysMissingAlgEntryPoint
-14136	eTISysInvalidRunningStatus
-14137	eTISysExceptionRunningInstantiation
-14138	eTISysTIShellBinaryNotFound
-14139	eTISysTimeoutWaitingForTIShell
-14140	eTISysSwapScriptTimeout
-14141	eTISysTIDSPModuleNotFound
-14142	eTISysTIDSPReadError

12.40.8.4 -142xx: DIDL errors

These errors all relate to the dynamic library loading system that manages ELF DLL binaries on Pro Tools HDX hardware. For example, a `eDIDL_FileNotFound` error will be raised if the ELF DLL name specified by an Effect's Describe code does not match any DLL that is present in the plug-in's bundle.

Table 4: DIDL error codes

Value	Definition
-14201	<code>eDIDL_FileNotFound</code>
-14202	<code>eDIDL_FileNotOpen</code>
-14203	<code>eDIDL_FileAlreadyOpen</code>
-14204	<code>eDIDL_InvalidElfFile</code>
-14205	<code>eDIDL_ImageNotFound</code>
-14206	<code>eDIDL_SymbolNotFound</code>
-14207	<code>eDIDL_DependencyNotLoaded</code>
-14208	<code>eDIDL_BadAlignment</code>
-14209	<code>eDIDL_NotImplemented</code>

12.40.8.5 -144xx: HDX hardware errors

These errors relate to failures on the HDX hardware itself. Plug-ins should never be able to trigger these error codes, which indicate low-level problems in the system.

Table 5: HDX hardware error codes

Value	Definition
-14401	<code>eBerlinImageError</code>
-14402	<code>eBerlinImageWriteError</code>
-14403	<code>eBerlinInvalidArgs</code>
-14404	<code>eBerlinCantGetTMSChannel</code>
-14405	<code>eBerlinChunkWriteError</code>
-14406	<code>eBerlinChunkReadError</code>
-14407	<code>eBerlinInvalidReqID</code>
-14408	<code>eBerlinDSPInResetError</code>
-14409	<code>eBerlinDSPTimeOut</code>
-14410	<code>eBerlinIncorrectTdmCableWiring</code>
-14411	<code>eBerlinInvalidClock</code>

12.40.8.6 -145xx: DHM isochronous audio engine errors

These errors relate to failures within the HDX audio engine software. Plug-ins should never be able to trigger these error codes, which indicate low-level problems in the system.

Table 6: DHM isochronous audio engine error codes

Value	Definition
-14500	eDsiIsochEngineGenericError
-14501	eDsiIsochEngineWrongChannelNumber
-14502	eDsiIsochEngineTxRingFull
-14503	eDsiIsochEngineRxRingNotReady
-14504	eDsiIsochEngineWrongNumberOfSamples
-14505	eDsiIsochEngineUnrecognizedSample
-14506	eDsiIsochEngineUnsupportedSample
-14507	eDsiIsochEngineUnsupportedNumberOfSamples
-14508	eDsiIsochEngineUnsupportedSample
-14509	eDsiIsochEngineDMAAlreadyEnabled
-14510	eDsiIsochEngineDMAAlreadyDisabled
-14511	eDsiIsochEngineInterruptHandler
-14512	eDsiIsochEngineBadCardRecord
-14513	eDsiIsochEngineCantSetValueDuringStreaming
-14514	eDsiIsochEngineStreamingAlready
-14515	eDsiIsochEngineStreamingAlready
-14516	eDsiIsochEngineStreamingCantBe
-14517	eDsiIsochEngineUnsupportedSamples
-14518	eDsiIsochEngineCantSetSamplesPerPeriod
-14519	eDsiIsochEngineInterruptLoop
-14520	eDsiIsochEngineGlobalDMADisabled
-14521	eDsiIsochEngineActiveInterruptMask
-14522	eDsiIsochEngineSDIOErrors

12.40.8.7 -30xxx: Dynamically-generated error codes

Errors in the -30xxx range are dynamically generated codes, and thus the same failure point could generate a different error code depending on the order in which errors occurred. These kinds of error codes are used heavily by the TI Shell Manager, the host component that interacts with the on-DSP shell environment.

If one of these error codes is being generated by the TI Shell Manager (the most common case) then you should be able to get more information about the failure by enabling the following [DigiTrace](#) logging facility:

DTF_TISHELLMGR=file@DTP_NORMAL

or, within the DSH tool:

```
enable_trace_facility [DTF_TISHELLMGR, DTP_NORMAL]
```

This should result in a log with more information such as the name of the failing plug-in, the dynamically generated error code, and a string description of its meaning. Depending on the failure case, the DAE dish command `getlastdsploaderror` can also sometimes be used to retrieve the description string for a dynamically-generated error if it was the last error generated during the DSP loading operation.

Collaboration diagram for TI Guide:



12.41 Page Table Guide

How to map a plug-in's parameters to control surfaces.

12.41.1 Contents

- [Introduction](#)
- [Avid Control Surfaces](#)
- [Plug-In Page Table Guidelines](#)
- [Avid Center Section Page Tables](#)
- [EUCON Page Tables](#)
- [Implementing Page Tables](#)
- [Appendix A. Get Parameter Value Info](#)

12.41.2 Introduction

12.41.2.1 Control Surfaces Overview

A tactile, external hardware control surface can be used to control different aspects of an application such as Pro Tools or Media Composer. Users prefer purpose-built control surfaces for DAW manipulation due to the control surface's superior accessibility, tactile feel, ergonomics, and user feedback.

Avid provides several different control surface products designed to accommodate a wide variety of user needs and workflows. Most Avid control surfaces implement EUCON (Extended User Control), a high-speed open control protocol featuring high-resolution, responsive control over almost all software functions. Some other control surfaces, such as the C|24, implement a dedicated control protocol for direct integration with Pro Tools. Finally, Pro Tools includes support for Mackie's HUI protocol and can interface with third-party control surfaces that implement this protocol.

12.41.2.2 Page Tables Overview

When a control surface is attached to a DAW system running AAX plug-ins the surface can be used to manipulate plug-ins' parameters. Plug-ins define the mapping between their parameters and control surface encoders using *page tables*.

Abstractly, a page table is a static mapping of a plug-in's controls to the interface of the control surface. Since a plug-in may have many more controls than the control surface can accommodate at a given time, the controls may be split across several "pages" that the user can freely switch between.

More concretely, a set of page tables is simply a set of single dimensional arrays. Each slot of the array corresponds to a particular rotary encoder or push-button of the control surface. By inserting control indices into the elements of the array, a plug-in's controls are mapped to particular tangible controls of the CS. Page tables are stored as XML data created by the [Page Table Editor](#) application available as part of the [AAX SDK Toolkit](#) on the [My Toolkits and Downloads](#) page at [avid.com](#). The XML is referenced by the plug-in as a resource using a call to [AAX_IEffectDescriptor::AddResourceInfo\(\)](#).

The following sections describe the various interfaces that the supported control surfaces provide for modifying plug-in parameters. Later, the specifics of implementation of page tables is described.

12.41.3 Avid Control Surfaces

12.41.3.1 EUCON

12.41.3.1.1 Pro Tools | Control app and Pro Tools | Dock

The free Pro Tools | Control app provides a EUCON-enabled control surface for iPad. Combining Pro Tools | Control with Pro Tools | S3, Pro Tools | Dock, or Artist Mix hardware adds additional touch workflows and custom control. Pro Tools | Control app display controlling an EQ plug-in instance

Pro Tools | Dock connects with an iPad and the free Pro Tools | Control iOS app, providing intelligent control of audio and video projects. The app offers a host of touch controls and visual feedback. The Dock provides eight push-top, touch-sensitive Soft Knobs that interact with whatever knobset has been chosen in the Pro Tools | Control app. Select an EQ, plug-in, send, pan, or other item, and all parameters instantly map to the knobs for tweaking. Pro Tools | Dock with iPad and Pro Tools | Control app

When laying out plug-in parameters on its display, Pro Tools | Control uses the same page table layout as [Artist Control](#)

12.41.3.1.2 Pro Tools | S6

Pro Tools | S6 is a modular control surface solution for the most demanding audio mixing and production environments. Built on the same proven technology that is core to the industry-leading ICON and System 5 product families, S6 enables mixers to quickly turn around complex projects while swiftly handling last-minute changes. With its unparalleled ability to simultaneously control multiple Pro Tools and other EUCON-enabled DAWs over simple Ethernet, S6 also speeds workflows and enables network collaboration on a single integrated platform. Pro Tools | S6

The main touch screen display on S6 can be configured to show graphs representing a plug-in's frequency or dynamics response curves. To support this feature, a plug-in must implement [AAX_IEffectParameters::GetCurveData\(\)](#) for the applicable [AAX_ECurveType](#) selector.

12.41.3.1.3 Pro Tools | S3

Pro Tools | S3 is a compact, EUCON-enabled, ergonomic desktop control surface that offers a streamlined yet versatile mixing solution for the modern sound engineer. Like S6, S3 delivers intelligent control over every aspect of Pro Tools and other DAWs, but at a more affordable price. While its small form factor makes it ideal for space-confined or on-the-go music and post mixing, it packs enormous power and accelerated mixing efficiency for faster turnarounds, making it the perfect fit everywhere, from project studios to the largest, most demanding facilities. Pro Tools | S3

On the S3 control surface, the top-right eight encoder/OLED pairs may be dedicated to any plug-in instance. In addition, the S3 includes support for dedicated EQ and dynamics plug-ins: a user can select a particular plug-in as his EQ or dynamics processor and use the surface's EQ, Compressor/Limiter, and Expander/Gate selectors to bring up the plug-in in a separate group of eight encoders. This mode uses the plug-in's D-Control Center Section EQ or Dynamics page tables when mapping plug-in controls. In order to be selectable as the system's EQ or dynamics processor a plug-in must meet the criteria for an ICON center-section plug-in.

12.41.3.1.4 Avid Artist Series: Artist Control

The Artist Control can run as either a standalone device or connected to additional units to form a larger system for your favored DAW. EUCON Artist Series: Artist Control

The plug-in editing section for the Artist Control consists of 8 touch and velocity sensitive rotary encoders, and 8 touch screen switches. The rotary knobs are physically laid out as two groups of 4, vertically aligned and positioned next to the customizable LED-backlit touch-screen interface, with their corresponding touch screen switches right next to them. The alpha-numeric scribble strip and plug-in editing displays are 8 characters wide. The knobs and switches can be assigned using the [Av81](#) page table.

Mapping of plug-in parameters to the surface's various controllers can also be done through use of the ProControl or D-Control page tables, giving the Artist Control the ability to emulate various plug-in editing sections(D-Control's Center Section EQ/Dynamics Sections and Channel Strip) or as a dedicated plug-in editing section (ProControl). Consequently, if using the ProControl page table, plug-in developers will have access to 16 controls, rotary encoders numbered top to bottom, left to right, as 0 through 7, and touch screen switches numbered top to bottom, left to right, as 8 through 15.

12.41.3.1.5 Avid Artist Series: Artist Mix

The Artist Mix can run as either a standalone device or connected to additional units to form a larger system for audio mixing applications.

EUCON Artist Series: Artist Mix

The plug-in editing section for the Artist Mix consists of 8 touch and velocity sensitive rotary encoders, and 8 switches. The rotary knobs are physically laid out horizontally along the top of the control surface as a group of 8, located right below the display screen, with the switches located directly below the encoders (the "ON" buttons). The alpha-numeric scribble strip and plug-in editing display are 8 characters wide. The Artist Mix is mapped using the Av18 page table, with the recommendation that the most important parameter is listed first. Like the Artist Control, mapping of plug-in parameters to the surface's various controllers can be done through use of the ProControl and/or D-Control page tables. This allows the Artist Mix to emulate the various plug-in editing sections that pertain to the D-Control, as well as the ProControl's dedicated plug-in section. Both rotary encoders and switches are numbered right to left, as 0 through 7, and 8 through 15 respectively.

12.41.3.1.6 Avid Pro Series: System 5

The System 5 Avid series consists of digital audio mixing systems that can be configured with hundreds of channels. These systems are used specifically for audio post production and music applications. Channels have a 4 band EQ, dynamics, two filters, and consist of 8 rotary knobs and a touch-sensitive motorized fader.

The plug-in editing section consists of 8 knob/switch pairs. This section can take its mapping from the Av81 page table, with the recommendation that the most important parameter is last (closer to the operator). Like the Artist Series of EUCON controllers, mapping of plug-in parameters can also be accomplished through use of either Pro Control or D-Control page tables. When using these page tables, the plug-in editing section is numbered top to bottom, 0 through 7, and 8 through 15, respectively.

The System 5 can also be put into a mode where a single plug-in is mapped across 4 rows of rotary knobs across an entire 8-fader bank. This mode uses the Av48 page table.

12.41.3.1.7 Avid Pro Series: MC Pro

The MC Pro is a workstation control surface that is geared towards professional post production. As the professional version of Avid's Artist Series Artist Control, it delivers precise and fast control of your editing applications.

The plug-in editing section consists of 8 pairs of touch-sensitive rotary knobs and switches. The rotary knobs are equipped with LED display rings that are available for control of EQ, Dynamics or any type of control your plug-in may need. As with the Artist Control, plug-in parameters can be mapped with the EUCON Av81 page table or by the ProControl or D-Control page tables described below. The 8 knobs are placed as two vertical groups of 4, laid out on the left half of the control surface, and are numbered top to bottom, left to right.

12.41.3.2 Avid C|24 and ICON

12.41.3.2.1 C|24

C|24 is a hardware control surface allowing great flexibility and power to Pro Tools systems. The alphanumeric scribble strip and plug-in editing displays allow 4 characters each for the plug-in name and a control's name. Three characters are allowed for the control's value.

C|24

The plug-in editing section consists of 24 horizontally aligned rotary data encoders and 24 switches lined up under the encoders. (the image above shows only 12 so that more detail can be displayed.) The C|24 encoders and switches are set up as pairs.

In any given control pair, either the encoder or a mix of encoder + the switch is used depending on how the mapped parameter has been defined in the plug-in:

- A parameter defined as a discrete parameter is automatically assigned to the switch and encoder.
 - The switch will increment the value of the parameter each time it is pressed.

- If a discrete parameter has only two possible values, the switch's backlight will be illuminated when the parameter has a non-zero value ("On")
- A parameter defined as a continuos will automatically be assigned to the rotary encoder.

Therefore, whether the plug-in has all switches, all encoders, or a mix, there are 24 controls available per page on the C|24.

12.41.3.2.2 ICON: D-Control ES

D-Control is a modular hardware control surface that adds high-quality tactile mixing and editing capability to Pro Tools systems. D-Control is a high-end mixing system that includes a center section and one or more fader packs. A fader pack consists of 16 channel strips, displaying track and plug-in information. Further general information about D-Control and how it works is available from the "BuckleyBriefing.pdf" on the developer website.

D-Control has the potential of displaying plug-in controls in four different sections: Channel Strip, Custom Fader, Center Section EQ, and Center Section Dynamics. The Dynamics section actually makes use of two different page table types, while the other three each have one page table type. In all sections, scribble strips are 6-characters wide and do not support the Digidesign Extended character set (i.e. special characters).

- Channel Strip

Plug-ins' controls are displayed in the channel strip section of the D-Control. Each channel strip consists of 6 vertically aligned encoder/switch pairs. This gives a total of 12 controls per page, where the encoders (from top to bottom) are numbers 1-6, and the switches (from top to bottom) are numbered 7-12. Like ProControl, the lower numbered encoder is paired with the lower numbered switch. In the diagram to the right, the button labeled "B-M-P" will control the switch.

There are a couple of special considerations when making D-Control Channel Strip page tables. First, keep in mind that the strips are at the top half of the control surface. Therefore, it will be more convenient for the user if you place the most frequently used controls at the bottom rather than at the top. Second, when a control is present in the switch of an encoder / switch pair, the "Pre" light will be lit as an indication to the user of the switch's presence. However, the name of the control placed on the switch will not appear in the scribble strip, unless the user presses the Sw Info button. The value of the switch will appear when the switch is pressed, but the name does not appear. Therefore, it is not recommended that you place a switch next to an empty encoder. It is recommended that you either place a switch next to an encoder that makes sense (like a Gain encoder next to a Phase switch) or place the switch control in both the switch and encoder so that the user will see the name of the control. These are merely guidelines, and not hard and fast rules.

D-Control: Channel Strip

- Custom Fader

D-Control can be placed in a special mode called Custom Fader to allow more controls of a plug-in to be displayed at once. D-Control takes the currently selected plug-in and displays its controls across 8 of the 16 channel strips such that the plug-in's first control is in the lower left control, its eighth is in the lower right. If there is more than one page of controls, D-Control displays additional pages, up to six, in the rows above the first. If the user has more than one fader pack, the Custom Faders can spread to more sets of faders.

D-Control: Custom Fader

D-Control: Custom Fader Page Layout

To support this mode, a plug-in must include a page table with 16 controls (8 encoder/switch pairs) per page. See the diagram to see how the layout of several Custom Fader page table pages works.

Since the one page layout is similar to the ProControl layout (16 controls per page with 8 encoders and 8 switches), D-Control will use your plug-in's ProControl page tables as a default. If you are not happy with how your plug-ins controls appear in this manner, you can override this by creating a Custom Fader page table.

- Center Section EQ and Dynamics Sections

D-Control's center section contains two dedicated areas for EQ and Dynamics plug-ins. Only plug-ins falling into these categories (EQ, Compressors, Limiters, Expanders, and Gates) should implement page tables for these sections. If your plug-in does not fall into these categories, you can skip these page table types. The remainder of the descriptions of these types can be found below in the Center Section Page Tables section.

12.41.3.2.3 ICON: D-Command ES

D-Command ES is a more compact yet expandable console than the D-Control. Coming standard with 8 physical faders/channel strips (two encoders per strip), it is expandable to 40 faders/channels.

The plug-in editing section for the D-Command follows closely to that of the D-Control, with two encoders per channel strip. It provides dedicated Dynamics and EQ sections for plug-ins that support Dynamics and EQ plug-in mapping, as well as the ability to display plug-ins in a Custom Fader section as described in the D-Control section. All sections provide scribble strips that 6 alpha numeric characters wide.

12.41.3.3 VENUE

VENUE is a revolutionary line of digital live sound systems that deliver amazing sound quality, reliability, flexibility, and ease-of-use. These live sound systems come equipped with plug-in editing sections, and work together to deliver studio-grade sound and powerful performance. For more information about using AAX plug-ins with VENUE systems see the [VENUE Guide](#).

12.41.3.3.1 VENUE | S6L

VENUE | S6L is a modular system designed to take on the world's most demanding tours and events with ease. Offering unprecedented processing capabilities - with over 300 processing channels - S6L delivers unrelenting performance and reliability through its advanced engine design and backs it up with modern touchscreen workflows and scalability to meet any challenge. VENUE | S6L console

S6L uses the same modular components as [Pro Tools | S6](#), but uses a different set of encoder layouts on these components in order to best support mixing workflows in a live sound setting. When displaying plug-in parameters, S6L maps the parameters onto its CKM. The leftmost two columns on the CKM are reserved (one column for constant operations, one as a "spacer" row with no assignments), leaving six columns of knob cells available for plug-in parameters. S6L uses the 'Av46' 4x6 page table type to map plug-in parameters to the CKM knob cells in this mode.

If a 'Av46' page table is not available from the plug-in, S6L uses the plug-in's C|24 'FrTL' layout in order to map one plug-in parameter to each of the 24 available knob cells. This generally leads to a very sub-optimal layout of parameters on the surface, both because the C|24 page table is designed for a linear layout and because it results in only one parameter assigned per knob cell, leaving two of the available encoders unassigned. Therefore, Avid strongly recommends that all AAX DSP plug-ins which are compatible with S6L are updated to include the new 4x6 page table layout.

S6L also supports dedicated EQ and dynamics plug-ins: a user can select a particular plug-in as his EQ or dynamics processor and use the surface's EQ, Compressor/Limiter, and Expander/Gate selectors to display the plug-in's parameters using a fixed layout for the given plug-in type. This mode uses the plug-in's D-Control Center Section EQ or Dynamics page tables when mapping plug-in controls. In order to be selectable as the system's EQ or dynamics processor a plug-in must meet the criteria for an ICON center-section plug-in.

12.41.3.3.2 VENUE | S3L-X

The VENUE | S3L-X System is a modular live sound solution including an HDX-powered processing engine, scalable remote I/O, and an [S3](#) control surface VENUE | S3L-X System

When used as part of an S3L system, the top-right eight encoder/OLED pairs on the S3 control surface (the Global Control encoders) may be placed into Insert mode in order to map to any plug-in instance. When in Insert Mode, the Global Control encoders use the plug-in's 'PcTL' page tables to map parameters onto the surface encoders.

Like S6L, S3L also includes support for dedicated EQ and dynamics plug-ins: a user can select a particular plug-in as his EQ or dynamics processor and use the surface's EQ, Compressor/Limiter, and Expander/Gate selectors to bring up the plug-in on the top-left eight encoder/OLED pairs on the S3 (the Channel Control encoders.) This mode uses the plug-in's D-Control Center Section EQ or Dynamics page tables when mapping plug-in controls. In order to be selectable as the system's EQ or dynamics processor a plug-in must meet the criteria for an ICON center-section plug-in. See [Using Channel Control](#) in the [VENUE Guide](#) and also [Center Section Parameter Mapping on VENUE | S3L-X](#) below for more information about encoder mapping in Channel Control mode.

12.41.4 Plug-In Page Table Guidelines

This section is intended as a guide in setting up defined 'pages' using some general rules. However, due to the sheer number of variables, it is simply not possible to account for all scenarios. But by following these suggestions, an AAX plug-in developer should find these guidelines useful in setting up their own plug-in pages. Moreover, it is hoped that a consistent and somewhat standard mapping topology will be realized across the broad range of plug-ins and control surfaces.

Here, we are primarily concerned with the number of controls on a control surface (CS) available for plug-in editing; and secondarily with the layout of controls provided for plug-ins. Accordingly, we need a method of mapping a plug-in's control parameters to a CS, and we need to take into account the varying numbers of controls available on a CS. Using 'page tables', from a software point of view, a plugin's control parameters can be mapped to a CS. Each page of the page table describes which of the plugin's parameters will be accessible from the CS's controls that are used for plug-in editing. Multiple pages are needed in the case where a CS has fewer controls available than the actual number of controls on a plug-in. We begin by stating guidelines that should be followed when mapping a plug-in's control parameters to a CS. At the end of this chapter, you will find the technical details of creating page tables for your plug-in.

The following guidelines are for simple, generic control surfaces. More advanced CSs, such as the MackieHUI and ProControl, have some guidelines of their own which are listed after these.

12.41.4.1 General Guidelines

Map a plug-in's controls from left-top to right-bottom sequentially onto each page.

Follow the layout of the plug-in GUI as closely as possible, allowing the controls to sequentially map to the Control Surface in the order specified above. In so doing, the CS controls will match the plug-in GUI; in the sense that by counting the location of a given control on the PI GUI, one should be able to grasp the corresponding slider or pot on the CS.

Note that Master Bypass, located in the plug-in's floating inserts window, should nearly always be placed as the first control on the first page. The only time this guideline might not be followed is if the plug-in has a particularly favorable layout for the control surface, and where this placement of Master Bypass would disrupt it. Also, on some control surfaces a dedicated bypass is already provided, in which case theMaster Bypass should not be mapped into the page table.

Related control parameters should be grouped together on the same page.

Controls that are often 'adjusted' with other similar or related controls should be mapped to the same page. This enhances the users ability to tweak related parameters and alleviates unnecessary paging.

Related control parameters should not be split across pages.

This follows directly from the bullet above. If some closely related controls cannot all fit on the same page, it is better to leave some blanks (i.e., unused pots, sliders, or switches) and move onto the next page where they can be adjusted together.

As a hypothetical example, let's say a control surface has 5 sliders, and we are mapping an EQ PI with 6 parameters - a low, mid, and high frequency band which has gain for each band. It would be best to map them to the CS as follows on page 1, from left to right on the CS: low freq, low gain, mid freq, mid gain, blank. Then map the remaining two parameters onto page 2: high freq, high gain, blank, blank.

Equivalent left and right stereo parameters should remain on the same page.

Since adjusting the left or right parameter of a stereo PI has considerable impact on the sound field, it is important that equivalent left and right stereo controls remain on the same page. Contrast this to placing the left parameters on one page, and the right parameters on another which is not desirable. This rule also changes according to the controller's layout. As an example, the CS-10 has 6 pots for PI editing arranged in a matrix of 3 rows x 2 columns. From left to right, the pots in row 1 are numbered 1 and 4. In row 2 the pots are numbered 2 and 5. Finally, the pots in row3 are numbered 3 and 6. A layout for L/R controls should be mapped param1L = control 1, param1R = control

4, and so on.

Repeat control parameters on pages where it makes sense to do so.

In some situations, it is desirable to have access to the same control on many pages. For example, this might mean having an output and/or input gain control available on each page of an EQ PI - since EQs change the overall gain. This is especially desirable if there would otherwise be blanks (i.e., unused pots, sliders, or switches).

12.41.5 Avid Center Section Page Tables

"Center Section" page tables provide a mapping of plug-in parameters to dedicated functions. These page tables are used by D-Control/D-Command (ICON), D-Show (VENUE), and EUCON-enabled consoles to provide a consistent user experience when interacting with EQ and Dynamics plug-ins.

There are three Center Section page table types:

Table type	Plug-in category
'DgEQ'	EQ
'DgCP'	Compressor/Limiter (Dynamics)
'DgGT'	Expander/Gate (Dynamics)

Dynamics plug-ins that include both Compressor/Limiter and Expander/Gate processing should support both DgCP and DgGT page tables.

The control surfaces which use these page tables each use a different physical layout of parameter functions onto the surface. These layouts have been designed to provide an intuitive and consistent way to control EQ and Dynamics plug-ins in a way that is appropriate for the encoder layout on each surface. By adding these page tables to your EQ and Dynamics plug-ins, your plug-ins will map correctly to all products which use these tables.

12.41.5.1 Center Section Page Table Guidelines

It is important to note that Center Section page table types are different from all other page table types in a fundamental way:

Each slot in the page table is *pre-defined* for a specific type of control. Therefore, your plug-in must conform to this pre-defined layout.

The purpose of these tables is to give the user a standard interface for EQ and Dynamics plug-ins - no matter what particular plug-in they are using. It allows the user to quickly access the most common controls in their favorite EQ or Dynamics plug-ins. This is different from all other page table types because the only restriction on other page table types is that - for types that have dedicated discrete controls - you cannot place continuous controls in a dedicated switch. However, the user will also know that if there are controls they would like to access that are missing from the Center Section, they can access them through one of the other layouts available on the control surface.

You'll notice looking at the pictures of these sections in D-Control (below) that the only scribble strips are in the center of the section. Unlike the Channel Strip section, there is not a scribble strip to label each control. The control's purpose is physically printed on the control surface. This model of hard-coding "center section" plug-in functions to specific encoders is followed on VENUE systems and on EUCON-enabled control surfaces which use these table types as well. That is why it is imperative that your plug-in conform to the pre-defined layout.

Because of the strict definitions of the layouts, it may mean that 1) not all of the controls for your plug-in can fit in these sections, and that 2) there may be controls your plug-in does not have and therefore are blank in this view. For example, let's say your plug-in is a 10-band EQ that does not have individual Q controls on any of the bands. Such a plug-in will be forced to leave off some of its bands, even though all Q controls specific to the bands on the page table are empty. That is fine as there will be another way for the user to display the plug-in on the control surface that will include all of the controls. For example, on D-Control, a user can also view an EQ or Dynamics plug-in in both the Channel Strip and Custom Fader modes which will display all controls. The important point is this:

Do not assign a parameter to a Center Section page table slot that does not match the parameter's function.

If you do, the parameter's function will be mislabeled and will cause confusion for the user.

You should only implement one page for these Center Section layouts. This is different from the other page table types, where it is expected to implement as many pages as necessary to give access to all controls in the plug-in. In the case of the Center Section layouts, you should only define one page, except in the case when the plug-in has separate controls for each channel (Left, Right, Center, etc.).

More than one page in Center Section layouts is allowed *only if the plug-in has separate controls for each channel.*

For example, if your EQ plug-in allows the user to change the EQ differently for the left and right channels, then you would implement two pages for the DgEQ page table. You'll notice in the pictures below for the EQ and Dynamics sections of D-Control, there are buttons labeled for channel selections (L, LC, C, RC, R, etc.). The control surface will automatically map the pages to the buttons, according to the standard order for surround channels. For example, in a stereo EQ, Left controls should be in the first page, and Right controls in the second. D-Control will automatically map page 1 to the L button and page 2 to the R button.

Note

We cannot emphasize strongly enough that trying to fill in all of your controls into these layouts, whether it is by creating extra pages, or by filling in empty slots, will only serve to confuse the user. You must adhere strictly to the guidelines given for these sections.

12.41.5.1.1 EQ Center Section Page Table Guidelines

To demonstrate the guidelines for EQ Center Section tables, we will use the D-Control Center Section encoders. Since all control surfaces which use Center Section page tables use a similar approach with hard-coded layouts for these types, the guidelines in this section are equally applicable to all control surfaces.

Take a look at D-Control's EQ section more closely in the image below.

D-Control EQ Center Section.

It supports a maximum of seven bands of EQ, each a vertical column of controls and labeled from left to right as follows:

- HPF (High-Pass Filter, nominally a high-pass filter or low frequency notch filter)
- LF (Low Frequency, nominally a parametric EQ or low frequency shelf filter)
- LMF (Low-Mid Frequency, nominally a parametric EQ)
- MF (Mid Frequency, nominally a parametric EQ)
- HMF (High-Mid Frequency, nominally a parametric EQ)
- HF (High Frequency, nominally a parametric EQ or high frequency shelf filter)
- LPF (Low-Pass Filter, nominally a low-pass filter or high frequency notch filter)

Each of these bands has a Q/Slope control, Frequency control, and an In Circuit/ Out of Circuit button. Five of the bands have an additional Gain control. Four of the bands have an additional EQ type selector switch, each surrounded by a pair of EQ type LED's. Input and Output Level controls are also available, as is a multi-channel Link button in the middle section.

Note

If an EQ plug-in implements a band that does not have an In/Out Circuit control or a Type control, but wants the related LED's to light properly, please see the discussion of the `GetParameterValueInfo()` API below.

The topmost rotary encoders in the HPF, LF, HF, and LPF bands are not labeled but they are indeed Q / Slope controls. The EQ type selector switches located between the Q/Slope and Frequency knobs control the type of filter on that band. Thus they define the behavior of all controls in that band (and not just the unlabeled Q/Slope control).

The EQ page table indices map to the dedicated EQ Center Section hardware encoders as follows:

- Equalization 'DgEQ'
 1. High Pass - In Circuit switch
 2. High Pass - Type switch
 3. High Pass - Frequency
 4. High Pass - Q/Slope
 5. Low Filter - In Circuit switch
 6. Low Filter - Type switch
 7. Low Filter - Gain
 8. Low Filter - Frequency
 9. Low Filter - Q/Slope
 10. Low-Mid Filter - In Circuit switch
 11. Low-Mid Filter - Type switch
 12. Low-Mid Filter - Gain
 13. Low-Mid Filter - Frequency
 14. Low-Mid Filter - Q/Slope
 15. Mid Filter - In Circuit switch
 16. Mid Filter - Type switch
 17. Mid Filter - Gain
 18. Mid Filter - Frequency
 19. Mid Filter - Q/Slope
 20. High-Mid Filter - In Circuit switch
 21. High-Mid Filter - Type switch
 22. High-Mid Filter - Gain
 23. High-Mid Filter - Frequency
 24. High-Mid Filter - Q/Slope
 25. High Filter - In Circuit switch
 26. High Filter - Type switch
 27. High Filter - Gain
 28. High Filter - Frequency
 29. High Filter - Q/Slope
 30. Low Pass - In Circuit switch
 31. Low Pass - Type switch
 32. Low Pass - Frequency
 33. Low Pass - Q/Slope
 34. Input Gain
 35. Output Gain
 36. Multi-channel Link switch
 37. High Pass - Q/Slope alternate parameter
 38. Low Filter - Q/Slope alternate parameter
 39. Low-Mid Filter - Q/Slope alternate parameter
 40. Mid Filter - Q/Slope alternate parameter
 41. High-Mid Filter - Q/Slope alternate parameter
 42. High Filter - Q/Slope alternate parameter
 43. Low Pass - Q/Slope alternate parameter

Note

In order to use the "Q/Slope alternate parameter" functions in the EQ page table, a plug-in must respond to the [AAX_ePageTable_UseAlternateControl](#) selector in the [GetParameterValueInfo\(\)](#) method. When the band type is changed, the control surface will call into the plug-in with this selector to determine if the control in the "Alt" position should be used. Please see the discussion of the [GetParameterValueInfo\(\)](#) below.

If your plug-in supports fewer than seven simultaneous bands of EQ, you have some options for where to place them. We recommend the following placement guidelines so users have consistency with various EQ plug-ins.

- If you only have one band of EQ, use the LF band.
- If you have one to four fully parametric bands, use LF, LMF, HMF, and HF (starting from left to right). (Skip the MF band.)
- If you have up to two shelving filters and two parametric bands, use LF (LF shelf), LMF (para), HMF (para), and HF (HF shelf). (Skip the MF band.)

Note that this layout includes a few additional functions not in D-Control's EQ section. These extra functions are the "Low-Mid Filter - Type switch", "Mid Filter - Type switch", and "High-Mid Filter - Type switch" slots.

12.41.5.1.2 Dynamics Center Section Page Table Guidelines

To demonstrate the guidelines for Dynamics Center Section tables, we will use the D-Control Center Section encoders. As with the EQ Center Section tables above, all control surfaces which use Center Section page tables use a similar approach with hard-coded layouts for these types, the guidelines in this section are equally applicable to all control surfaces.

The D-Control Dynamics section is shown below. Keep in mind that the D-Control's Dynamics section displays page tables for both the Compressor/Limiter page table type and the Expander/Gate type. Therefore, the function of certain rotaries and switches differs depending on which page table type has been loaded. In these cases, there is a LED to indicate the current function of a rotary or switch.

D-Control Dynamics Center Section.

Several rotary encoders have alternate uses while others are always dedicated to one function. The two page tables' indices map to the dedicated Dynamics Center Section hardware encoders as follows:

- Compressor/Limiter 'DgCP'
 1. Threshold
 2. Ratio
 3. Attack Time
 4. Release Time
 5. Knee
 6. Make-up Gain
 7. High Pass - In Circuit / Out of Circuit switch
 8. High Pass - EQ Type switch (with notch and high-pass filter LEDs)
 9. High Pass - Frequency
 10. High Pass - Q/Slope
 11. Low Pass - In Circuit / Out of Circuit switch
 12. Low Pass - EQ Type switch (with notch and low-pass filter LEDs)
 13. Low Pass - Frequency
 14. Low Pass - Q/Slope
 15. External Key switch (middle section)
 16. Key Listen switch (middle section)

- 17. Input Gain
- 18. Output Gain
- 19. Multi-channel Link switch (middle section)
- 20. Depth (not included on ICON)
- Expander/Gate 'DgGT'
 - 1. Threshold
 - 2. Ratio
 - 3. Range
 - 4. Attack Time
 - 5. Release Time
 - 6. Hysteresis
 - 7. Hold
 - 8. High Pass - In Circuit / Out of Circuit switch
 - 9. High Pass - EQ Type switch (with notch and high-pass filter LEDs)
 - 10. High Pass - Frequency
 - 11. High Pass - Q/Slope
 - 12. Low Pass - In Circuit / Out of Circuit switch
 - 13. Low Pass - EQ Type switch (with notch and low-pass filter LEDs)
 - 14. Low Pass - Frequency
 - 15. Low Pass - Q/Slope
 - 16. External Key switch (middle section)
 - 17. Key Listen switch (middle section)
 - 18. Input Gain
 - 19. Output Gain
 - 20. Multi-channel Link switch (middle section)
 - 21. Knee (not included on ICON)

The compressor/limiter page table, DgCP, supports all the controls above except Range, Hysteresis, and Hold. (As you create the page table in the [Page Table Editor](#), this is clear.)

The expander/gate page table, DgGT, supports all the controls above except Knee and Makeup Gain. It does support both Ratio and Range. The user presses the Page button to select between them.

A plug-in can support both DgCP and DgGT page tables. Again, the user presses the Page button to select between them. If a plug-in supports both of these page tables and the DgGT page table includes support for both Ratio and Range controls, pressing the Page button will switch between all available controls as follows:

DgCP -> DgGT with Ratio -> DgGT with Range (and all other controls unchanged)-> DgCP -> ...

12.41.5.2 Center Section Parameter Mapping to Single-Column/Row Layouts

The following tables show the layout mapping and hard-coded names for center section page tables on EUCON surfaces which use single-column or single-row layouts for Eq and Dyn plug-in modes. The tables show the assignment of specific center section table indices to cells on the EUCON control surface. Each EUCON control surface cell includes three encoders: a rotary knob encoder and two push-button encoders. These tables use the following codes to indicate encoders in each control surface cell:

- Knob = Knob encoder
- Knob(Sel I) = Knob encoder with Sel active
- Knob(Sel O) = Knob encoder with Sel inactive
- *In* = In switch

Note

For center section page table layouts the "Sel" push-button encoder is only used to toggle the rotary encoder between two possible parameters. It is never mapped to a single discrete parameter in these layouts.

With some versions of EUCON, the cell mapping on the first page is "reversed" relative to the second page when the surface is laid out horizontally; the first page moves left to right through increasing cell indices, while later pages move right to left.

12.41.5.2.1 'DgEQ' PageTable - Equalizer

		Page 1								Page 2							
		Far from user / Left				Close to user / Right				Far from user / Left				Close to user / Right			
1	Function	8	7	6	5	4	3	2	1	16	15	14	13	12	11	10	9
1	H↔ P↔ F In/↔ Out														Knob		
2	H↔ P↔ F Type														In		
3	H↔ P↔ F Freq														Knob (Sel O)		
4	H↔ P↔ F Q														Knob (Sel I)		
5	Lo In/↔ Out									In							
6	Lo Type										In						
7	Lo Gain																
8	Lo Freq											Knob					
9	Lo Q											Knob (Sel O)					
10	Lo Mid In/↔ Out									In					Knob (Sel I)		

11	Lo Mid Type					<i>In</i>								
12	Lo Mid Gain				Knob									
13	Lo Mid Freq						Knob (Sel O)							
14	Lo Mid Q						Knob (Sel I)							
15	Mid <i>In/↔</i> Out										<i>In</i>			
16	Mid Type									<i>In</i>				
17	Mid Gain											Knob		
18	Mid Freq											Knob (Sel O)		
19	Mid Q											Knob (Sel I)		
20	Hi Mid <i>In/↔</i> Out		<i>In</i>			<i>In</i>								
21	Hi Mid Type					<i>In</i>								
22	Hi Mid Gain			Knob										
23	Hi Mid Freq						Knob (Sel O)							
24	Hi Mid Q						Knob (Sel I)							
25	Hi <i>In/↔</i> Out	<i>In</i>												
26	Hi Type		<i>In</i>											
27	Hi Gain	Knob												

28	Hi Freq	Knob (Sel O)						
29	Hi Q	Knob (Sel I)						
30	L↔ P↔ F In/↔ Out					Knob		
31	L↔ P↔ F Type				In			
32	L↔ P↔ F Freq					Knob (Sel O)		
33	L↔ P↔ F Q					Knob (Sel I)		
34	In- put Level							Knob
35	Out- put Level							Knob
36	Link							
37	H↔ P↔ F Q Alt						Knob (Sel I)*	
38	Lo Q Alt					Knob (Sel I)*		
39	Lo Mid Q Alt		Knob (Sel I)*					
40	Mid Q Alt						Knob (Sel I)*	

41	Hi Mid Q Alt			Knob (Sel I)*												
42	Hi Q Alt	Knob (Sel I)*														
43	L↔ P↔ F Q Alt								Knob (Sel I)*							

Notes

- The multi-channel link switch (index 36) is not mapped to any encoder in this layout
- The knob assignments marked with an asterisk will be assigned depending on the plug-in's response to [AX_IEffectParameters::GetParameterValueInfo\(\)](#) with the [AA�_ePageTable_UseAlternateControl](#) selector. If the plug-in provides [AA�_eUseAlternateControl_Yes](#) then the assignment marked with an asterisk will be used.

12.41.5.2.2 Both 'DgCP' and 'DgGT' PageTables - Multi-dynamics

If both Dynamics center section page table types are defined for the plug-in then EUCON control surfaces will use the following mapping.

Note

Some control surfaces may only partially follow this mapping. For example, the mapping of the Artist Mix control surface in Dyn mode uses two pages: it follows this mapping for encoder cells 1-8 on the first page and follows the '['DgCP'-only mapping](#)' for encoder cells 9-16 on the second page.

		Page 1								Page 2										
		Far from user / Left				Close to user / Right				Close to user / Left				Far from user / Right						
		Function	C	Thre-	8	7	6	5	4	3	2	1	16	15	14	13	12	11	10	9
Dg↔	C↔	P	1	C									Knob							
Dg↔	C↔	P	2	Ra-					Knob					Knob						
Dg↔	C↔	P	3	At-									Knob				Knob			

Dg← C↔ P 4	C Re- lease Time					Knob			Knob		
Dg← C↔ P 5	C Knee							Knob			
Dg← C↔ P 6	C Gain Mak									Knob	
Dg← C↔ P 7	H↔ P↔ F En- able										
Dg← C↔ P 8	H↔ P↔ F Type										
Dg← C↔ P 9	H↔ P↔ F Freq										
Dg← C↔ P 10	H↔ P↔ F Q										
Dg← C↔ P 11	L↔ P↔ F En- able										
Dg← C↔ P 12	L↔ P↔ F Type										
Dg← C↔ P 13	L↔ P↔ F Freq										
Dg← C↔ P 14	L↔ P↔ F Q										

Dg← G↔ T 6	X Hys- tere- sis													
Dg← G↔ T 7	X Hold													

	Page 3								Page 4								
	Close to user / Left				Far from user / Right				Close to user / Left				Far from user / Right				
Dg← C↔ P 1	Function C Thre- old	24	23	22	21	20	19	18	17	32	31	30	29	28	27	26	25
Dg← C↔ P 2	C Ra- tio																
Dg← C↔ P 3	C At- tack Time																
Dg← C↔ P 4	C Re- lease Time																
Dg← C↔ P 5	C Knee																
Dg← C↔ P 6	C Gain Mak																
Dg← C↔ P 7	H↔ P↔ F En- able																Knob

Dg← C↔ P 8	H↔ P↔ F Type								In			
Dg← C↔ P 9	H↔ P↔ F Freq								Knob (Sel O)			
Dg← C↔ P 10	H↔ P↔ F Q								Knob (Sel I)			
Dg← C↔ P 11	L↔ P↔ F En- able								Knob			
Dg← C↔ P 12	L↔ P↔ F Type								In			
Dg← C↔ P 13	L↔ P↔ F Freq								Knob (Sel O)			
Dg← C↔ P 14	L↔ P↔ F Q								Knob (Sel I)			
Dg← C↔ P 15	Ext Key								Knob			
Dg← C↔ P 16	Key List- ten								Knob			
Dg← C↔ P 17	In- put Gain								Knob			
Dg← C↔ P 18	Out- put Gain								Knob			

Dg← C↔ P 19	Link												
Dg← C↔ P 20	Dept												
Dg← G↔ T 1	X Thre- old	Knob											
Dg← G↔ T 2	X Ra- tio	Knob											
Dg← G↔ T 3	X Rang				Knob								
Dg← G↔ T 4	X At- tack		Knob										
Dg← G↔ T 5	X Re- lease			Knob									
Dg← G↔ T 6	X Hys- tere- sis					Knob							
Dg← G↔ T 7	X Hold						Knob						

Notes

- Neither table's multi-channel link switch ('DgCP' index 19 and 'DgGT' index 20) is mapped to an encoder in this layout
- The 'DgGT' filter, external key, and gain parameters (indices 8 through 19) are not mapped to any encoders in this layout

12.41.5.2.3 'DgCP' PageTable - Compressor/Limiter

If the plug-in defines a 'DgCP' page table but does not define a 'DgGT' page table then EUCON control surfaces will use the following mapping.

		Page 1*								Page 2							
		Far from user / Left				Close to user / Right				Far from user / Left				Close to user / Right			
		8	7	6	5	4	3	2	1	16	15	14	13	12	11	10	9
1	Function C																
2	Threshold C Ratio							Knob									
3	C Attack Time									Knob							
4	C Release Time										Knob						
5	C Knee	Knob															
6	C Gain Make		Knob			Knob											
7	H↔ P↔ F												Knob				
8	H↔ P↔ F Type												In				
9	H↔ P↔ F Freq												Knob (Sel O)				
10	H↔ P↔ F Q												Knob (Sel I)				
11	L↔ P↔ F En-able												Knob				

12	L↔ P↔ F Type														In		
13	L↔ P↔ F Freq														Knob (Sel O)		
14	L↔ P↔ F Q														Knob (Sel I)		
15	Ext Key															Knob	
16	Key List- ten																Knob/In
17	In- put Gain			Knob													
18	Out- put Gain				Knob												
19	Link																
20	Dept										Knob						

Notes

- If no parameter is defined at the Ratio parameter index then the Makeup Gain parameter will be mapped to the Ratio parameter's normal spot in order to increase usefulness of the first-page mapping.
- Pro Tools versions prior to Pro Tools 11.1 use a different layout for the first page of this table type

12.41.5.2.4 'DgGT' PageTable - Expander/Gate

If the plug-in defines a 'DgGT' page table but does not define a 'DgCP' page table then EUCON control surfaces will use the following mapping.

		Page 1								Page 2									
		Far from user / Left				Close to user / Right				Far from user / Left				Close to user / Right					
		Function X	Thre- old	8	7	6	5	4	3	2	1	16	15	14	13	12	11	10	9
1												Knob							

2	X Ratio						Knob					
3	X Range			Knob								
4	X Attack Time				Knob							
5	X Release Time					Knob						
6	X Hysteresis	Knob										
7	X Hold		Knob									
8	H↔ P↔ F Enabled							Knob				
9	H↔ P↔ F Type								In			
10	H↔ P↔ F Freq								Knob (Sel O)			
11	H↔ P↔ F Q								Knob (Sel I)			
12	L↔ P↔ F Enabled									Knob		
13	L↔ P↔ F Type									In		
14	L↔ P↔ F Freq									Knob (Sel O)		

15	L P F Q										Knob (Sel I)	
16	Ext Key											Knob
17	Key List- ten											Knob/ <i>In</i>
18	Input Gain										Knob	
19	Out- put Gain										Knob	
20	Link											

12.41.5.3 Center Section Parameter Mapping in S6 Expand Mode

In addition to supporting single-row and single-column EQ and Dynamics layouts as described above, S6 also includes an Expand Mode for EQ and Dynamics plug-ins which allows the targeted plug-in's EQ or Dynamics center section mapping to be displayed across an entire CKM module.

12.41.5.3.1 'DgEQ' PageTable - Equalizer

EQ layout in S6 Expand Mode

12.41.5.3.2 'DgCP' and 'DgGT' PageTables - Compressor/Limiter and Expander/Gate

Dynamics layout in S6 Expand Mode

12.41.5.4 Center Section Parameter Mapping on VENUE | S3L-X

Built-in processing parameters are mapped to the Channel Control encoders on VENUE | S3L. For more information about Channel Control mode in S3L-X see [Using Channel Control](#) in the [VENUE Guide](#).

12.41.5.4.1 'DgEQ' PageTable - Equalizer

Channel Control Encoder	Knob	Sel Switch	In Switch
1	Low Gain		Low EQ band in/out
2	Low Freq/Q	Toggles Freq/Q	Low EQ band type
3	LoMid Gain		LoMid EQ band in/out
4	LoMid Freq/Q	Toggles Freq/Q	
5	HiMid Gain		HiMid EQ band in/out
6	HiMid Freq/Q	Toggles Freq/Q	
7	High Gain		High EQ band in/out
8	High Freq/Q	Toggles Freq/Q	High EQ band type

12.41.5.4.2 'DgCP' and 'DgGT' PageTables - Compressor/Limiter and Expander/Gate

Channel Control Encoder	Knob	Sel Switch	In Switch
1	Threshold level		Comp/Lim or Exp/Gate in/out
2	Ratio		
3	Attack		
4	Knee		
5	Release		
6	Gain level		
7	Key HF	Key Listen	Filter in/out
8	Key LF	Key In	Filter in/out

12.41.6 EUCON Page Tables

Plug-ins should implement specific EUCON page tables to take advantage of EUCON-specific features and layouts. By writing EUCON-specific page tables, your plug-in is able to re-define both the in/out button as well as the select button per EUCON control cell on compatible surfaces.

Host Compatibility Notes Pro Tools versions prior to Pro Tools 11.1 use plug-ins' ProControl and ICON page tables (Dynamics, EQ, Channel Strip, Custom Fader, etc.) to map plug-in parameters to EUCON-enabled surfaces, so be sure that your plug-ins also implement these page tables correctly so that users with earlier versions of Pro Tools can have the best possible experience when using your plug-ins.

Note

The legacy PeTE editor will remove all EUCON sections from any page table XML file that it saves. Developers should no longer use PeTE for AAX page table editing. If you do use PeTE, it is important to *always back up your page table file* before editing the file in PeTE.

12.41.6.1 Specification

EUCON page tables use modern formatting, making them more intuitive to implement than non-EUCON tables. Here are some example lines from a EUCON page table:

```
<PageTable type='Av18' pgSz='24' >
  <Page num='1'>
    <Cell row='1' col='1' knobID="Knob1" inOutButtonID="Button1A" selectButtonID="Button1B" />
    <Cell row='1' col='2' knobID="Knob2" inOutButtonID="Button2A" selectButtonID="Button2B" />
    <Cell row='1' col='3' knobID="Knob3" inOutButtonID="Button3A" selectButtonID="Button3B" />
    ...
  </Page>
  ...
</PageTable >
```

The EUCON PageTable element includes a series of Cell sub-elements. The attributes of each Cell sub-element are as follows:

- **row** - the cell's row position, ordered furthest to nearest
- **col** - the cell's column position, ordered left to right
- **knobID** - the parameter ID associated with the knob in question
- **inOutButtonID** - the parameter ID associated with the "In" button next to the knob in question. This must be a discrete parameter.

- `selectButtonID` - the parameter ID associated with the "Sel" button next to the knob in question. This can be a discrete or continuous parameter. If it is a continuous parameter then pushing "Sel" will toggle the knob associated with the `selectButtonID` and `knobID` parameters.

12.41.6.2 Types

Av81	Av18	Av48	Av46
<code>type='Av81'</code>	<code>type='Av18'</code>	<code>type='Av48'</code>	<code>type='Av46'</code>
8 rows	1 row	4 rows	4 rows
1 column	8 columns	8 columns	6 columns
<code>pgsz='24'</code> (3 elements)	<code>pgsz='24'</code> (3 elements)	<code>pgsz='96'</code> (3 elements)	<code>pgsz='72'</code> (3 elements)
placed in vertical sets of	placed in horizontal sets	placed in 4x8 knob cell	placed in 4x6 knob cell
Most important	Most important controls	Layout should be <code>Expander</code>	Layout should be similar
parameters should be placed in highest-numbered rows (closest to user)	should be placed in lowest-numbered columns (left)	Map to GUI	Layout in GUI

12.41.6.3 Conventions

- To map a single discrete parameter to an encoder cell, assign the parameter to both the knob and the In switch. Assigning the parameter to the cell's knob will ensure that the parameter name and value is always displayed on the surface. The user will be able to edit the parameter using either the rotary encoder or the In switch.
- When assigning discrete parameters, always prefer to use the In switch over the Sel switch. For cells with one continuous and one discrete parameter, users will always expect the discrete parameter to be assigned to the In switch rather than the Sel switch. In other EUCON modes (besides plug-in editing) the In switch is always used to enable/disable parameters, while the Sel switch is usually used to toggle between functions for the cell's rotary encoder.

12.41.6.4 Requirements

- All parameter IDs used in the EUCON page tables must be defined with a `Ctrl_ID` element within the `ControlNameVariations` element
- Every `Cell` with at least one parameter assignment must include a `knobID` assignment. It is not valid to assign either `inOutButtonID` or `selectButtonID` without also assigning the cell's `knobID`.
- For a given `knobID`, the same parameters must be assigned to the `selectButtonID` and `inOutButtonID` switches across all EUCON page tables
- Every knob cell assignment set (Rotary+Sel+In assignment) used in the '`Av48`' table must be exactly replicated somewhere in the '`Av81`' table
- '`Av48`' tables may contain no more than two pages

12.41.7 Implementing Page Tables

12.41.7.1 Page table XML specification

This section includes a rough specification for plug-in page table XML. Whenever possible, we encourage developers to use the [Page Table Editor](#) tool to generate plug-in page tables rather than writing or editing the page table XML by hand.

The page table XML format contains three main tags:

- [PageTableLayouts](#) tag
- [ControlNameVariations](#) tag
- [Editor](#) tag

12.41.7.1.1 PageTableLayouts tag

This section provides a static mapping of control elements to page table layouts. Multiple layouts may be provided in this section, e.g. in cases where different control sets are used by different plug-in Types.

Each layout includes a complete set of page table descriptions. There are multiple kinds of page tables, each of which may have multiple pages. At minimum, each layout must include a `PageTable` with `type='pgTL'` and `pgsz='1'`. This is the default page table, and the order of the control elements that it describes must match the order in which the corresponding parameters are added to the plug-in itself.

Here is an excerpt from the `PageTableLayouts` section in Avid's Eleven plug-in page tables demonstrating non-EUCON `PageTable` elements. For the EUCON `PageTable` element specification, see [Eucon page table specification](#).

```
<PageTableLayouts>
    <Plugin manID='Digi' prodID='ElvF' plugID='ELFr'>
        <Desc>Eleven Free 1 &gt; 1 by Avid Technology, Inc.</Desc>
        <Layout>PageTable Free</Layout>
    </Plugin><!--manID='Digi' prodID='ElvF' plugID='ELFr'-->
    <Plugin manID='Digi' prodID='Elvn' plugID='ELVr'>
        <Desc>Eleven 1 &gt; 1 by Avid Technology, Inc.</Desc>
        <Layout>PageTable 1</Layout>
    </Plugin><!--manID='Digi' prodID='Elvn' plugID='ELVr'-->

    <!-- ... -->

    <PTLayout name='PageTable 1'>
        <PageTable type='BkCS' pgsz='12'>
            <Page num='1'>
                <ID>Mic Type</ID>
                <ID>Cab Type</ID>
                <ID>Amp Type</ID>
                <ID>Master</ID>
                <ID>Gain 2</ID>
                <ID>Gain 1</ID>
                <ID>Mic Axis</ID>
                <ID>Cab Type</ID>
                <ID>Amp Type</ID>
                <ID> </ID>
                <ID> </ID>
                <ID>Bright Switch</ID>
            </Page><!--num='1'-->
            <Page num='2'>
                <!-- ... -->
            </Page><!--num='2'-->
            <Page num='3'>
                <!-- ... -->
            </Page><!--num='3'-->
        </PageTable><!--type='BkCS' pgsz='12'-->

        <!-- ... -->

        <PageTable type='PgTL' pgsz='1'>
            <Page num='1'>
                <ID>Master Bypass</ID>
            </Page><!--num='1'-->
            <Page num='2'>
                <ID>Input Level</ID>
            </Page><!--num='2'-->
            <Page num='3'>
                <ID>Output Level</ID>
            </Page><!--num='3'-->
            <Page num='4'>
                <ID>Gate Threshold</ID>
            </Page><!--num='4'-->

            <!-- ... -->
        </PageTable><!--type='PgTL' pgsz='1'-->
    </PTLayout><!--name='PageTable 1'-->
    <PTLayout name='PageTable Free'>
        <!-- ... -->
    </PTLayout><!--name='PageTable Free'-->
</PageTableLayouts>
```

The sub-tags for this section are as follows:

- **Plugin element**
A high-level description of a single plug-in Type.

The `manID`, `prodID`, and `plugID` attributes must match the corresponding Manufacturer, Product, and Type IDs for each plug-in Type that is described in the XML file. Multiple `Plugin` elements may be included in a single XML file.

The `Plugin` element has two sub-elements:

1. The `Desc` sub-element provides a brief description of the plug-in Type. This information is used only by the [Page Table Editor](#) application and does not affect operation of the plug-in in Pro Tools.
2. One `Layout` sub-element is used to bind the plug-in Type to a particular `PTLayout` (see below.)

- `PTLayout` element

A complete control mapping, including a full set of `PageTable` descriptions.

- `PageTable` sub-element - A single page table mapping, with controls specified across multiple `Page` elements as in the example above.

`PageTable` elements have the following attributes:

1. `type` defines the particular device that will use the `PageTable`. `type` may be one of:
 - * `PgTL` - Generic page tables (any size)
 - * `PCTL` - ProControl, VENUE, and fall-back EUCON page table (size 16)
 - * `MkTL` - Makie HUI page table (size 8)
 - * `HgTL` - 002/003 and Command|8 page table (size 8)
 - * `FrTL` - Control 24, C|24, and fall-back S6L page table (size 24)
 - * `BkCS` - ICON Channel Strip (size 12)
 - * `BkSF` - ICON Custom Fader (size 16)
 - * `DgGT` - ICON dynamics section (Gate/Expander) (size 20)
 - * `DgCP` - ICON dynamics section (Compressor/Limiter) (size 19)
 - * `DgEQ` - ICON EQ section (size 43)
 - * `Av81` - EUCON 8x1 section (size 24)
 - * `Av18` - EUCON 1x8 section (size 24)
 - * `Av48` - EUCON 4x8 section (size 96)
 - * `Av46` - EUCON 4x6 section (size 72)
2. `pgsz` defines the number of controls per page in the page table. Most page table types require a specific size, as noted above. The generic `PgTL` page tables may be of any size, and multiple `PgTL` page tables may be provided. However, each plug-in must provide a `PgTL` page table of size 1 that includes all of the plug-in's automatable parameters, in the order in which they are added to the plug-in.

12.41.7.1.2 ControlNameVariations tag

This section includes information about the names of each control in the plug-in. Here is an excerpt from our Eleven plug-in page tables which demonstrates the basic format of this section:

```
<Ctrl ID='Amp Bypass'>
  <name type='PgTL' sz='1'>AB</name>
  <name type='PgTL' sz='4'>AByp</name>
  <name type='PgTL' sz='5'>A Byp</name>
  <name type='PgTL' sz='6'>AmpByp</name>
  <name type='PgTL' sz='7'>Amp Byp</name>
</Ctrl><!--ID='Amp Bypass'-->
<Ctrl ID='Amp Type'>
  <name type='PgTL' sz='1'>AT</name>
  <name type='PgTL' sz='3'>Amp </name>
  <name type='PgTL' sz='5'>AmpTp</name>
  <name type='PgTL' sz='6'>AmpTyp</name>
  <name type='PgTL' sz='7'>Amp Typ</name>
</Ctrl><!--ID='Amp Type'-->
<Ctrl ID='Bright Switch'>
  <name type='PgTL' sz='1'>Brt </name>
  <name type='PgTL' sz='6'>Bright</name>
</Ctrl><!--ID='Bright Switch'-->
```

The sub-tags used are as follows:

Ctrl element with **ID** attribute - The identifier of the control that is being identified.

The identifiers for all parameters in the page table must and should match the IDs used for the control both in the **<PageTableLayouts>** layouts and in the **Editor** tag's **DiscCtrls** sub-tag (see below). See [Parameter identifiers](#) for more information about parameter identifiers.

name element - The desired abbreviated name of the control for display on control surface UIs, which may have limited display space available.

The **typ** parameter allows you to choose which page table type the given abbreviation will be specific to. For example, a name provided with **typ='HgTL'** would only appear on Command|8, 002, and 003 hardware. As with all other tags, the name associated with **typ PgTL** (the generic page table identifier) will be used if no other name is given for the specific page table that is being loaded.

The **sz** parameter defines the size of the control surface display for which the given name is appropriate. The control surface will use the name associated with the largest size that will fit on its display.

To reduce the number of names that must be specified, abbreviated names can be given that are longer than their specified size. These names will be truncated when necessary. For example, a control surface that could accommodate two characters on its display would use the size-1 name for the "Bright Switch" control above, since 1 is the largest number less than or equal to 2 from the provided set of name sizes (in this case, 1 and 6.) The size-1 name, '**Brt**', has four characters in it. Therefore, it would be truncated down to '**Br**' to fit onto the control surface's display.

12.41.7.1.3 Editor tag

The last of the three main sections in an XML plug-in page table is the **Editor** section. This section is used by the [Page Table Editor](#) application and you should not need to modify its contents. However, if you are encountering problems modifying your plug-in in the Page Table Editor then you may wish to verify that all plug-ins and controls are properly identified within the **PluginList** and **DiscCtrls** sub-tabs, respectively. Here is the **Editor** section from the Eleven page table XML:

```
<Editor vers='1.3.7.1'>
  <PluginList>
    <TDM>
      <PluginID manID='Digi' prodID='Elvn' plugID='ELVt'>
        <MenuStr>TDM: Eleven, 1 in X 1 out</MenuStr>
      </PluginID><!--manID='Digi' prodID='Elvn' plugID='ELVt'-->
    </TDM>
    <RTAS>
      <PluginID manID='Digi' prodID='Elvn' plugID='ELVr'>
        <MenuStr>RTAS: Eleven, 1 in X 1 out</MenuStr>
      </PluginID><!--manID='Digi' prodID='Elvn' plugID='ELVr'-->
      <PluginID manID='Digi' prodID='ElvF' plugID='ELFr'>
        <MenuStr>RTAS: Eleven Free, 1 in X 1 out</MenuStr>
      </PluginID><!--manID='Digi' prodID='ElvF' plugID='ELFr'-->
    </RTAS>
  </PluginList>
  <DiscCtrls>
    <CtrlID>Amp Bypass</CtrlID>
    <CtrlID>Amp Type</CtrlID>
    <CtrlID>Bright Switch</CtrlID>
    <CtrlID>Cab Bypass</CtrlID>
    <CtrlID>Cab Type</CtrlID>
    <CtrlID>Master Bypass</CtrlID>
    <CtrlID>Mic Axis</CtrlID>
    <CtrlID>Mic Type</CtrlID>
  </DiscCtrls>
</Editor><!--vers='1.3.7.1'-->
```

12.41.7.2 Parameter identifiers

The **ID** tags/arguments in a page table must reference parameters which are exposed by the plug-in's [AAX_IEffectParameters](#) implementation via methods such as [GetParameterIndex\(\)](#).

There are two supported ways to identify parameters in a page table:

- By the parameter's ID (preferred)

- By the parameter's 31-character name (legacy)

A single page table may only reference the plug-in's parameters using one of these two approaches; a page table file may not reference one parameter by its name and another by its ID, and it may not reference one parameter by its name in one location but by its ID in another location.

If a plug-in will change its parameters' names at run time then the parameter identifiers used in the page tables must reference parameters by ID.

12.41.7.3 Creating page tables using the AAX Plug-In Page Table Editor

Warning

The legacy PeTE tool has been deprecated. Developers should now use the AAX Plug-In Page Table Editor tool for all AAX plug-in page table editing.

Page tables can be created and edited using the AAX Plug-In Page Table Editor application available on the Developer website. The Page Table Editor generates an XML file that can be used in both Windows and Macintosh plug-in projects.

The Page Table Editor will make use of the Parameter IDs that are defined in a plug-in, and will associate them with the 'Plugin' tags in the XML file. Using the DemoDist sample plug-in included in the AAX SDK, you'll see that there is one Plugin entry for each plug-in type in the binary.

Note

For compatibility between your AAX and corresponding RTAS or TDM plug-ins, make sure the 4 character IDs for [AAE_eProperty_ManufacturerID](#), [AAE_eProperty_ProductID](#), [AAE_eProperty_PluginID_Native](#), and [AAE_eProperty_PluginID_AudioSuite](#) are identical to the legacy SDK's counterpart.

```
// Type, product, and relation IDs
const AAX_CTypeID cDemoDist_ManufactureID ='AVID ';
const AAX_CTypeID cDemoDist_ProductID = 'DmDE ';
const AAX_CTypeID cDemoDist_TypeID_AS = 'DmAS ';
const AAX_CTypeID cDemoDist_TypeID_MonoNative = 'DmRT ';
const AAX_CTypeID cDemoDist_TypeID_StereoNative = 'DsRT ';
const AAX_CTypeID cDemoDist_TypeID_MonoTI = 'DDT1 ';
const AAX_CTypeID cDemoDist_TypeID_StereoTI = 'DDT2 ';
```

Listing 1: DemoDist Plug-In IDs

```
.
.

<Plugin manID ='AVID ' prodID ='DmDE ' plugID ='DmRT '>
<Desc > DemoDist 1 -&gt; 1 by Avid Technology , Inc .</ Desc >
<Layout > PageTable 1</ Layout >
</ Plugin ><!-- manID ='AVID ' prodID ='DmDE ' plugID ='DmRT '-->
<Plugin manID ='AVID ' prodID ='DmDE ' plugID ='DsRT '>
<Desc > DemoDist 2 -&gt; 2 by Avid Technology , Inc .</ Desc >
<Layout > PageTable 1</ Layout >
</ Plugin ><!-- manID ='AVID ' prodID ='DmDE ' plugID ='DsRT '-->
<Plugin manID ='AVID ' prodID ='DmDE ' plugID ='DDT1 '>
<Desc > DemoDist 1 -&gt; 1 by Avid Technology , Inc .</ Desc >
<Layout > PageTable 1</ Layout >
</ Plugin ><!-- manID ='AVID ' prodID ='DDT1 '-->
<Plugin manID ='AVID ' prodID ='DmDE ' plugID ='DDT2 '>
<Desc > DemoDist 2 -&gt; 2 by Avid Technology , Inc .</ Desc >
<Layout > PageTable 1</ Layout >
</ Plugin ><!-- manID ='AVID ' prodID ='DDT2 '-->
.
.
.
```

Listing 2: DemoDist XML

For more information about the AAX Plug-In Page Table Editor tool, see the ReadMe file which accompanies the application.

12.41.7.4 Verifying Page Table Layouts: The Hidden Pop-Up Menu

You can verify the page tables created in your plug-in in Pro Tools with a "hidden" developer debug Page Tables popup menu. To verify the page tables, first include the YourPageTables.r file in your project and compile the plug-in. Then, after launching Pro Tools, instantiate the plug-in, hold down the Commandkey (Ctrl-key in Windows) and mouse-click the Automation button in the plug-in window to display the menu.

- Category

The Category menu item has a submenu listing the names of possible categories. Any category that the plug-in belongs to will have a check mark next to it. In addition, appended to the name of the category is an indication of whether that category can be bypassed, and if so, the control number (#) and control name of the associated bypass control. Also appended is the number of the first page on which a control associated with the category can be found. This page number is based on whatever the current page table type is selected in the "TableType" menu item. For example: Delay (can bypass, control #9, Master Bypass) (first page #1)

- Table Type

Sets the type of control surface page table.

- Page Size

Sets the number of controls per page. The identifier "custom" is shown next to page sizes that have been specifically implemented (which at minimum, should appear next to page sizes of 5, 6, 8, and 16!). The identifier "default" is shown next to page sizes that do not have specific support in your page table file. Note that the Mackie and ProControl table type will automatically set this to 8 and 16, respectively.

- Control Name Length

Sets the number of characters to be displayed in the control's name (shown in the Page menu below). The identifier "expected length" appears next to lengths that should be specifically addressed (3, 4, 5, 6, 7, 8, and 31). If you use the XML page table system, the names can be specified in the [Page Table Editor](#) application. Otherwise, the function GetControlNameOfLength() is responsible for providing names with these lengths.

- Control Value Length

Sets the number of characters to be displayed in the control's value (shown in the Page menu below). The identifier "expected length" appears next to lengths that should be specifically addressed (4, 5, 6, 7, 8, and 31; also, 3 is used for ProControl switch states). The plug-in Library call GetValueString() is responsible for providing values with these lengths.

- Highlighted Page

Highlights the selected page in the plug-in window in Pro Tools.

- Highlight Color

Sets the highlight color. The highlight color can be: red, green, blue or yellow. Note that at minimum, plug-in's should support these four colors of highlighting!

- Page X

The actual page table layouts are shown here. The following information can be seen in this menu item.

- The control's name, as returned from the XML page tables. If the table type is Mackie, then the length will be 4 (unless overridden by changing the "Control Name Length" menu item). If the table type is ProControl, the length will be 3 (again, unless overridden, but usually there is no point in doing so). Also, with ProControl set as the table type, the special ProControl symbols will appear here if they are part of the name (however, note that they are small and can be difficult to see). Finally, if the table type is default, the name will be shown with the number of characters as specified in the "Control Name Length" menu item.
- The control's value is shown next. Both the Mackie and ProControl table types will automatically set the control's value length to 4 and 3, respectively. Otherwise, this can be set with the "Control Value Length" menu item. GetValueString() is responsible for providing this 'value' information.
- The number of control steps is shown next for continuous and discrete controls. For example, a discrete control will appear as: "(discrete: N steps)", where N is the # of steps of the control.

- If "(NoL)" is displayed, this simply means that it is a new plug-in, and supports either the XML page tables or the GetParameterNameOfLength() function call. If "(NoL)" does not appear, then the plug-in is older and you should not expect the control names or values to be optimized - since they are created by just truncating the longer values that the older plug-ins return.
- If "(highlight)" is displayed, this means that the string will be reverse highlighted when displayed on hardware controllers that support it. Not all hardware controllers utilize reverse highlighting.
- Next, orientation flags for each plug-in control will be displayed. The format is: "(Value: xxx yyy)," where xxx is either "BMin" (kDAE_BottomMinTopMax) or "TMin" (kDAE_TopMinBottomMax); and yyy is either "LMin" (kDAE_LeftMinRightMax) or "RMin" (kDAE_RightMinLeftMax). This text identifies the value of the control's orientation flags, as returned by GetParameterOrientation().
- Also shown is the radial LED encoder-display mode (as returned by GetControlOrientation()) assigned to each control (this radial LED surrounds each encoder). The possible modes are: "spread" kDAE_RotateMode, "wrap" kDAE_RotaryWrapMode, "boost" kDAE_RotaryBoostCutMode, or "dot" kDAE_RotarySingleDotMode, along with either "RMin" kDAE_RotaryRightMinLeftMax, or "LMin" kDAE_RotaryLeftMinRightMax - indicating which side the minimum AAE value is being mapped to.

12.41.7.5 Testing new XML resources

To quickly test your changes to a plug-in's page table XML file, you can do the following:

- Save the XML file with a higher version number than the last saved file
- Copy the XML file to the Plug-ins folder and drop it next to your plug-in
- Change the name of the file to *PluginFilename.xml*. *PluginFilename* must match exactly the plug-ins filename. For example, if your plug-in's file name is RectiFi.aaxplugin, the XML file must be called RectiFi.xml
- Launch Pro Tools. AAE will find the file, and as long as its version number is higher than the XML compiled into the plug-in, it will load the file's page tables.

12.41.7.6 Control Highlighting Scheme

Note that plug-in controls that are currently controllable on any page of a plug-in will be highlighted in blue when they are the active page on a control surface. Therefore, it is very important to implement the highlighting of controls in your plug-in. Plug-in highlighting is also used with automation. In general, four common colors should be implemented:

- Red: Write automated
- Green: Read automated
- Blue: Accessible on control surface (stays blue if control is also read automated)
- Yellow: Accessible on control surface and write automated

You can use the "hidden" popup menu above to test that your color schemes are working properly.

12.41.7.7 Control Numbering Layouts

Most of the advanced control surfaces have both rotary encoders (knobs) and switches. The knobs and switches are handled differently by different surfaces. C|24 has a set of 24 rotary encoders and 24 switches for plug-in editing, and 003 has 8 encoders and 8 switches, all set up in pairs. However, both of these control surfaces automatically assign a control with only two possible values (e.g., "on" or "off") to a switch, while a control with three or more possible values, whether it is discrete or continuous, is automatically assigned to the rotary encoder. Therefore, no distinction is made between control numbers for switches and control numbers for encoders.

The following tables show how the control numbers are arranged for control surfaces which have distinctions between their encoders and switches.

Encoders	Switches
1	7
2	8
3	9
4	10
5	11
6	12

Table 12.1: Table 2: D-Control Channel Strip - Numbering Layout

Encoders	Switches
1	9
2	10
3	11
4	12
5	13
6	14
7	15
8	16

Table 12.2: Table 3: D-Control/Pro-Control Custom Fader Mode - Numbering Layout

12.41.7.8 Alphanumeric Displays

With the advent of the newer advanced control surfaces (CS), plug-ins now have the opportunity to provide information to the user via alphanumeric displays on the CS. Unfortunately, the displays are limited in the number of characters that can be shown. For instance, on the Mackie HUI, nine characters are provided for each plug-in control, allowing only four characters for the control name, and four characters for the control value, and one for a space between them. On ProControl, eight characters are provided for each plug-in control, with three characters allocated to displaying a control name, and four characters for its control value. On C|24 and 003, four characters are provided for the plug-in name, control name, and the control value. For the control value, these four characters include a +/- and/or any necessary unit abbreviations (ex: K, s, dB, etc.). D-Control and Command|8 provide six characters for the plug-in name, control name, and control value.

In order to display meaningful information in these short character strings, plug-ins will have to optimize the strings that they return to AAE. The dispatcher calls MapControlValToString(), which in turn calls GetValueString(), is used to obtain these control value strings. Typically in the past, the requested length argument of both these member functions, i.e., maxChars in MapControlValToString() or maxLength in GetValueString() has been ignored; but, from here on out, they should be carefully examined and used to create an optimum and meaningful string for any requested length.

To prevent the code from becoming unwieldy, as in the case of trying to provide strings for all requested lengths, a minimum number of expected lengths should be specifically addressed. The expected value lengths are: 4, 5, 6, 7, 8, and 31. In addition, ProControl switch states are a maximum length of 3 (i.e., when dealing with the state of a switch for ProControl, provide this information in 3 characters or less). Therefore, whenever possible, a plug-in should return the most meaningful string that will fit in any particular requested length, and at minimum, handle the expected lengths. If a plug-in does not have custom code to handle a particular requested length, it can round the length down to the next smaller expected length and use the code that it has for it. For example, a request of 9 characters should be converted to an expected request of 8 characters.

Please note: Since truncating a long string to fit within the requested length will not provide meaningful results in most cases, plug-ins must specifically provide code for deriving useful strings for the expected lengths where applicable.

Since the smaller lengths, especially 4 and 5 characters, are usually too short to display a plug-in's true full value including units, some decisions will have to be made about how to suitably shorten them. The following provides some general guidelines.

If needed, and in order of precedence, try to:

- Remove spaces: 13 Hz becomes 13Hz

- Use common abbreviations for units: 16 seconds to 16sec, or 16 s, 156 Hertz to 156Hz
- Drop the units entirely: 1832 Hz to 1832
- Round the value: 173.3 ms to 173

Here is a table depicting a typical example in more detail. The expected lengths are shown in the left most vertical column.

Alphanumeric Characters

While creating the above strings, the requested length argument passed in (maxChars in MapControlValToString(), and maxLength in GetValString()) should be strictly adhered to! The string returned should be no greater than the requested number of characters. Furthermore, the developer should assume that the buffer passed into this function is only as large as the requested length. Any intermediate string processing should be done in temporary local buffers and only when you have the final string should you copy back to the buffer that was passed in, making sure you copy no more than the requested number of characters, plus the Null character or Length byte, as appropriate; since in general, MapControlValToString() uses C strings and GetValString() uses Pascal strings.

Also, in an effort to further help prevent buffer overruns, two new functions have been added to the PI library file SliderConversions.cp: SmartAppendNum() and SmartAppendXNum(), which includes a maximum length argument and should be used in place of FicAppendNum() and FicAppendXNum() from FicBasics.cpp.

Finally, note that ProControl and HUI will sometimes utilize 5 characters to display its value when a sign is involved. For instance, if the number is -100, the negative sign will appear in the space separating the control name from the control value. Pro Tools will take care of this conversion as long as all of the expected lengths are properly provided for. For C|24 and 003, this is not the case - only four characters are allowed, so the +/- must be a part of those four characters.

HUI, ProControl, C|24, and 003 all provide alphanumeric displays for visual feedback, with the primary purpose being plug-in parameter editing. Functions in the plug-in library are provided that allow customized parameter strings to be created for use on the display. Specifically, these functions are: GetControlNameOfLength() and Get←ValueString(). Fortunately, the details of writing to the display are taken care of by the application. Therefore, the plug-in developer only needs to be concerned with providing meaningful display strings for all plug-in parameters that are controllable. Whether using the XML or legacy page table system, GetControlNameOfLength() should return the long version (31 characters maximum) of the plug-in's control names. Short versions of plug-in control names are stored in the XML file, edited with the [Page Table Editor](#) application. If you are also using legacy page tables to support versions of Pro Tools prior to 6.4, the short names should also be coded in the GetControlNameOfLength() function.

AAE clients like Pro Tools call GetControlNameOfLength(), but if AAE finds XML data stored in the plug-in, it gets the information from there rather than calling into the plug-in. GetControlNameOfLength() is responsible for providing the parameter "names" used on the display. As with parameter "value" strings, it is important to carefully create parameter names that are meaningful in the limited space allocated to displaying parameter names. On ProControl, string lengths of three characters will be used for the parameter name strings, where four characters will be used on the HUI, C|24, and 003. D-Control and Command|8 use six characters for control names. In general, lengths of 3, 4, 5, 6, 7, 8, and 31 should be specifically addressed in the Page Table Editor or GetControlNameOfLength(). We begin next, by looking at some concrete examples.

12.41.7.9 ProControl Display

ProControl also provides an alphanumeric display; however, there are some significant differences that need to be addressed. Shown is a generic representation for the ProControl display. The image below shows what users will usually be viewing on ProControl's displays, which are the current Encoder/Value settings. Figure 13 shows the current switch settings when the user toggles to the Switch/State display mode. ProControl will only display one of three views at any given time.

Pro-Control Encoder Display

ProControl Switch Display

As with the HUI, meaningful strings will have to be provided in the limited available lengths. ProControl, C|24 or 003 value strings require no special treatment other than what has already been stated above for HUI, since all

of these control surfaces display their values in 4 digits/characters, as returned by `GetValueString()`. The biggest difference between the HUI display and the Avid control surfaces' displays is that the Avid control surfaces can display symbols.

Let's take a moment to explain how the displays of C|24 and 003 work. Both of these surfaces have a four character LED display located above each encoder/switch pair. When in Channel mode, these scribble strips show the plug-in name (if any) on that channel. When that plug-in is selected, the display switches to show the controls for the plug-in. Now each display shows the name of the control. The display automatically switches to the current value of the control when the encoder or switch is moved or pushed.

12.41.8 Appendix A. Get Parameter Value Info

12.41.8.1 Overview

```
AAX_Result GetParameterValueInfo ( AAX_CParamID iParameterID, int32_t iSelector, int32_t* oValue)
```

`GetParameterValueInfo()` is implemented at the Data Model's [AAX_CEffectParameters](#) level, and will allow the app to query a plug-in for the "meaning" of its parameter values. It was designed as a general purpose mechanism that will find additional uses with new selectors in the future. It is used:

- to ensure the EQ and Dynamics sections' EQ type selector LEDs light appropriately for a given band's filter type. We want the appropriate EQ type LED to light for each band's filter type -whether or not your band can switch between filter types.
- to ensure the state of the EQ section's In buttons matches the values of the associated plug-in controls. When each band's In button is lit, it must mean that the EQ is active/on/enabled. When it is unlit, it must mean that the EQ is off/disabled/bypassed. (Some plug-ins have EQ bypass controls (On = bypass); others have EQ In buttons (On = On). We can derive "meaning" from the control regardless of control value.)
- similarly, to ensure the state of the Dynamics section's Filt In buttons matches the values of the associated plug-in controls. When each band's Filt In button is lit, it must mean that the EQ is inactive/on/enabled. When it is unlit, it must mean that the EQ is off/disabled/bypassed.

12.41.8.2 Implementation

`GetParameterValueInfo()` will be of type [AAX_EParameterValueInfoSelector](#) ([AAX.Enums.h](#)) and will allow for future queries on parameter value "meaning."

```
enum AAX_EParameterValueInfoSelector
{
    AAX_ePageTable_EQ_Band_Type = 0,
    AAX_ePageTable_EQ_InCircuitPolarity = 1,
    AAX_ePageTable_UseAlternateControl = 2
};
```

Listing 3: Parameter Selector Enums

Results (not return values) passed back by `GetParameterValueInfo()` will be of one of these two types:

```
enum AAX_EQBandTypes
{
    AAX_eEQBandType_HighPass = 0,
    AAX_eEQBandType_LowShelf = 1,
    AAX_eEQBandType_Parametric = 2,
    AAX_eEQBandType_HighShelf = 3,
    AAX_eEQBandType_LowPass = 4,
    AAX_eEQBandType_Notch = 5
};
```

Listing 4: EQ Band Type Enums

```
enum AAX_EQInCircuitPolarity
{
    AAX_eEQInCircuitPolarity_Enabled = 0,
    AAX_eEQInCircuitPolarity_Bypassed = 1,
    AAX_eEQInCircuitPolarity_Disabled = 2
};
```

Listing 5: EQ Circuit Polarity Enums

```
enum AAX_EUseAlternateControl
{
    AAX_eUseAlternateControl_No = 0,
    AAX_eUseAlternateControl_Yes = 1
};
```

Listing 6: Alternate Control Enum

Please see [AAX.Enums.h](#) for more information.

To add support for this method, you must to override [AAX_CEffectParameters::GetParameterValueInfo\(\)](#) from within your plug-ins Data Model. For a given [AAX_CParamID](#), selector, and parameter value, you must pass back a result (not a return value) denoting the "meaning" of that parameter value. If a parameter has no meaning in the context of the given selector, you should return [AAX_ERROR_UNIMPLEMENTED](#).

One point to note, is that an EQ or Dynamics plug-in may have a band of EQ that does not include an EQ type selector control. The band is just always, say, a HPF. There's no control to map to the EQ type selector button in the page table and therefore no obvious control for which you would pass back [AAX_eEQBandType_HighPass](#) in [GetParameterValueInfo\(\)](#). What is the solution? Well, the other controls on that band (Frequency, Q/Slope, Gain, In) know what type of band they're on. Thus the plug-in should pass back the relevant EQ_Band_Types enum in [GetParameterValueInfo\(\)](#) for all controls on a given band of EQ. This also applies to key filter bands in your Dynamics plug-ins. In this way, the EQ type selector LEDs in both the EQ and Dynamics sections will always light appropriately.

The second exception applies to EQ or Dynamics plug-ins that have a band of EQ that does not include an In Circuit / Out of Circuit control for that band. In this case, the band is always In Circuit (or On) and the other controls for this band should return [AAX_eEQInCircuitPolarity_Enabled](#) as the result for [GetParameterValueInfo\(\)](#) when the [AAX_ePageTable_EQ_InCircuitPolarity](#) selector is passed in. Code snippets for [GetParameterValueInfo\(\)](#) from the EQ III 7-Band AAX plug-in is provided below:

Note

The logic for the In Circuit / Out of Circuit LED is available in Pro Tools 6.7 and higher.

```
// ****
// METHOD:  GetParameterValueInfo
// ****
AAX_Result EQIII_7_Parameters::GetParameterValueInfo ( AAX_CParamID iParameterID,
    int32_t iSelector, int32_t* oValue) const
{
    const AAX_IParameter * parameter = mParameterManager.GetParameterByID( iParameterID );

    if ( !parameter )
        return AAX_ERROR_INVALID_PARAMETER_ID;

    if ( iSelector == AAX_ePageTable_EQ_Band_Type )
    {
        if ( parameter->Name() == EQIII_HPF_Type )
        {
            switch( parameter->GetStepValue() )
            {
                case 0 /*Notch*/: *oValue = AAX_eEQBandType_Notch; break;
                case 1 /*HiPass*/: *oValue = AAX_eEQBandType_HighPass; break;
                default: *oValue = -1; return AAX_ERROR_UNIMPLEMENTED; break;
            }
            return AAX_SUCCESS;
        }
        else if ( parameter->Name() == EQIII_LF_Type )
        {
            switch( parameter->GetStepValue() )
            {
                case 0 /*Peak*/: *oValue = AAX_eEQBandType_Parametric; break;
                case 1 /*Shelf*/: *oValue = AAX_eEQBandType_LowShelf; break;
                default: *oValue = -1; return AAX_ERROR_UNIMPLEMENTED; break;
            }
        }
    }
}
```

```

        return AAX_SUCCESS;
    }
    else if (parameter->Name() == EQIII_HF_Type)
    {
        switch (parameter->GetStepValue())
        {
            case 0 /*Peak*/: *oValue = AAX_eEQBandType_Parametric; break;
            case 1 /*Shelf*/: *oValue = AAX_eEQBandType_HighShelf; break;
            default: *oValue = -1; return AAX_ERROR_UNIMPLEMENTED; break;
        }
        return AAX_SUCCESS;
    }
    else if (parameter->Name() == EQIII_LPF_Type)
    {
        switch (parameter->GetStepValue())
        {
            case 0 /*Notch*/: *oValue = AAX_eEQBandType_Notch; break;
            case 1 /*LoPass*/: *oValue = AAX_eEQBandType_LowPass; break;
            default: *oValue = -1; return AAX_ERROR_UNIMPLEMENTED; break;
        }
        return AAX_SUCCESS;
    }
}
else if (iSelector == AAX_ePageTable_UseAlternateControl)
{
    if ( (parameter->Name() == EQIII_HPF_Type) ||
        (parameter->Name() == EQIII_HPF_Q) ||
        (parameter->Name() == EQIII_HPF_Slope) )
    {
        const AAX_IParameter* typeParam = mParameterManager.GetParameterByID(
EQIII_HPF_Type_Ch);
        *oValue = (typeParam->GetStepValue() == 0 /*Notch*/) ?
AAX_eUseAlternateControl_No :
AAX_eUseAlternateControl_Yes;
        return AAX_SUCCESS;
    }
    else if ( (parameter->Name() == EQIII_LPF_Type) ||
        (parameter->Name() == EQIII_LPF_Q) ||
        (parameter->Name() == EQIII_LPF_Slope) )
    {
        const AAX_IParameter* typeParam = mParameterManager.GetParameterByID(
EQIII_LPF_Type_Ch);
        *oValue = (typeParam->GetStepValue() == 0 /*Notch*/) ?
AAX_eUseAlternateControl_No :
AAX_eUseAlternateControl_Yes;
        return AAX_SUCCESS;
    }
}

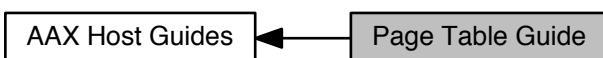
return AAX_ERROR_UNIMPLEMENTED;
};

```

Listing 7: EQ III GetParameterValueInfo()

As you'll notice, the EQ III plug-in has separate controls for Q and Slope in its HPF and LPF bands, depending on whether the band is set to notch or band pass. When the Band Type control is set to notch, the continuous Q control is used. When the Band Type is set to Hi/Lo Pass, the discrete Slope control is used. In the page table, the HPF / LPF Q controls are set to the "Q or Slope" in the [Page Table Editor](#) application, and the HPF / LPF Slope controls are set to the "Q or Slope Alt". Then, with the [GetParameterValueInfo\(\)](#) implementation above, the control surface properly swaps the controls when the band type control is changed. In other words, when the function is called with the [AAX_ePageTable_UseAlternateControl](#) selector, if the band type control is set to notch, then [AAX_eUseAlternateControl_No](#) is returned in the result and the control surface puts the Q control at that position. If the band type is set to pass, then [AAX_eUseAlternateControl_Yes](#) is returned and the Slope is placed at that position.

Collaboration diagram for Page Table Guide:



12.42 DigiTrace Guide

How to add tracing to your plug-ins and view logging from the plug-in host.

12.42.1 On this page

- [What is DigiTrace?](#)
- [DigiTrace quick start guide](#)
- [DigiTrace log files](#)
- [Configuring DigiTrace](#)
- [Bonus features](#)
- [Adding traces to an AAX plug-in](#)
- [Advanced DigiTrace configuration](#)
- [Compatibility](#)
- [Additional Information](#)

12.42.2 What is DigiTrace?

DigiTrace is a logging tool used by many Avid audio applications. DigiTrace provides high-performance, real-time tracing capabilities and can help you debug hard-to-isolate problems in real-time code. Pro Tools and other Avid audio products are instrumented with DigiTrace, and it is easy to add DigiTrace logging to your AAX plug-ins.

This document outlines how to use DigiTrace, both as a developer to add trace instrumentation to your code and as an end user to view or record trace instrumentation for an instrumented application.

12.42.2.1 What does DigiTrace do?

DigiTrace generates encrypted logs on users' systems. These log files can be decrypted via the DigiTraceDecryptor application that is included in the DigiTrace Tools package.

By default, DigiTrace logs basic information including details about the system, software, component versions, and any errors that are encountered. By using a simple configuration text file, DigiTrace can be easily configured to provide additional logging information such as plug-in loading details. Here are some examples of how you can use DigiTrace:

- You can use DigiTrace in your plug-ins when you need a convenient, high-performance logging solution.
- You can use the default DigiTrace logs that Pro Tools generates to help you understand problems that your plug-ins encounter when running on Pro Tools.
- You can add DigiTrace statements and stack traces to your released plug-ins in order to help you troubleshoot end-user issues more quickly.
- You can (and should!) submit DigiTrace logs when reporting bugs and other Pro Tools issues to Avid.

12.42.3 DigiTrace quick start guide

This section provides quick steps for the following common tasks:

- [Find and decrypt DigiTrace log files](#)

- Configure DigiTrace for AAX plug-in logging
- Configure DigiTrace for plain-text output
- Add tracing to a plug-in

12.42.3.1 Find and decrypt DigiTrace log files

DigiTrace log files are placed into a common logs directory. The specific directory that is used depends on the version of DigiTrace - see [Where are DigiTrace log files stored?](#)

By default, the version of DigiTrace that is installed with Avid audio products generates logs in an encrypted format with the extension ".dlog". Developer builds of Pro Tools and other applications are configured to generate plain-text logs.

You can convert .dlog files to plain-text using the DigiTraceDecryptor tool that is included in the DigiTrace Tools package available for download from the Avid developer portal. To decrypt a log using this tool, simply drag-and-drop the .dlog file onto the tool. You can also set this tool as the default application for opening .dlog files in your OS, which will allow you to decrypt and open .dlog files directly.

12.42.3.2 Configure DigiTrace for AAX plug-in logging

You must customize the DigiTrace configuration to enable extra logging, such as debug logging for [AAX](#) plug-ins.

DigiTrace uses a plain-text configuration file to enable custom logging. This file uses the suffix ".digitrace" and is located within or beside the application. For example, the configuration files for Pro Tools are located at:

- OS X: Pro Tools.app/Contents/Resources/config.digitrace
- Windows: C:\Program Files\Avid\Pro Tools\ProTools.digitrace

To configure DigiTrace to print logs from [AAX_TRACE](#) or [AAX_TRACE_RELEASE](#) macros in [AAX](#) plug-ins, add the following line to the .digitrace configuration file for the application:

DTF_AAXPLUGINS=file@DTP_LOWEST

If a config.digitrace file does not already exist for a DigiTrace-enabled application then you can create it to enable DigiTrace. For more information about customizing the DigiTrace configuration and enabling different levels of debug logging, see [Configuring DigiTrace](#).

12.42.3.3 Configure DigiTrace for plain-text output

In order to be able to view streaming log output in real time, DigiTrace must be configured for plain-text output. This is the default configuration for developer builds of Pro Tools and other Avid audio applications.

To configure shipping applications for plain-text log output, you must replace the application's installed DigiTrace library with a development version of the DigiTrace library. Development builds of DigiTrace are included in the DigiTrace Tools package. Search for "Digitrace.framework" on OS X or "DigiTrace.dll" on Windows and replace the installed shipping version of the library with the developer version from the DigiTrace Tools package to configure the application for plain-text output.

Note that the developer version of DigiTrace may output logs to a different directory than the shipping version. In general, developer builds of DigiTrace will place log files in a directory next to the instrumented application. For example, developer builds of Pro Tools will output logs to a logs directory placed adjacent to the Pro Tools application bundle rather than in the user's Library/Logs/Avid folder.

12.42.3.4 Add tracing to a plug-in

To easily add tracing to an [AAX](#) plug-in, use the [AAX_TRACE](#) or [AAX_TRACE_RELEASE](#) macros. Logging from the "release" macro will be enabled for all builds of the plug-in, whereas logging from the "standard" macro will only be enabled in Debug builds of the plug-in.

12.42.4 DigiTrace log files

The default logging in Avid audio applications includes data that can be useful in many different troubleshooting situations. For example:

- Information about the user's system configuration
- A complete list of loaded components and libraries. (If the user has an old or incompatible version of your plug-ins installed on his system, you will know about it!)
- Crash logs in the event of a system failure

In addition, you can add DigiTrace logging code to your plug-ins, helping you examine potential issues in the way your plug-in is running on a user's computer even when you cannot reproduce the issue locally.

12.42.4.1 Where are DigiTrace log files stored?

12.42.4.1.1 Log directory

DigiTrace logs are stored in a log files directory on the user's system. The default location of this directory depends on the operating system and on which version of DigiTrace is being used:

- Pro Tools / DigiTrace 11.0 and later: ~/Library/Logs/Avid/ (OS X) C:\Program Files\Avid\Pro Tools\Logs (Windows)
- Pro Tools / DigiTrace 9.0 - 10.3: ~/Library/Logs/AvidLogFiles/ (OS X) C:\Users\[username]\AvidLogFiles\ (Windows 7 or Windows 8) C:\Documents and Settings\[username]\AvidLogFiles\ (Windows XP)
- Pro Tools / DigiTrace 8.0.3 - 8.5: [Pro Tools application directory]/LogFiles/ (All configurations)

This default log directory can be overridden in the DigiTrace config file. See [Advanced DigiTrace configuration](#) for more information.

12.42.4.1.2 Log file names

By default the log file will be given a time-stamped name in the format <AppName>_YYYY_MM_DD_HH_MM↔_SS.dlog. This timestamp represents the system time when the log was created. Like the log directory, this log file name can be changed using the DigiTrace configuration. See [Advanced DigiTrace configuration](#) for more information.

12.42.4.2 Monitoring DigiTrace logs

12.42.4.2.1 Log files

You can of course view a log file by opening it periodically. In addition, assuming that DigiTrace is [configured for plain text output](#), you can also constantly monitor a log file in a "streaming" manner. This is possible using standard Unix tools included with OS X or with Cygwin on Windows. In fact, this approach usually works better than telling DigiTrace to use console output due to buffering of the console output.

- For basic real-time monitoring of a single file, use `tail:tail -f /path/to/digitrace/logs/the↔_logfile.txt`
- For real-time monitoring of the most recent file in the log file directory, use a combination of `tail` and `ls`:

12.42.4.2.2 Console

Console behavior is quite different between OS X and Windows

Windows On Windows, traces sent to the console go to the system debugging console. The only way to view the console output is to be running with an attached debugger.

OS X On the Mac, console traces are sent to stdout console, which shows up in a few places:

- If you're running in a debugger, the debug console will display stdout output, including DigiTrace messages
- If you're not in the debugger, you can view the output in the Console app (/Applications/Utilities/Console). For Pro Tools, look under ~/Library/Logs/Avid/Pro Tools.X.log in the log list. Note that these messages are not displayed in the "All Messages" log.
- Alternately, you can manually look at the log output, again using the `tail` command, e.g. `tail -f "~/Library/Logs/Avid/Pro Tools.0.log"`

12.42.4.3 Log file formatting

Here is the beginning of an example DigiTrace log:

```
*** Digidesign Session Trace for: /Applications/Pro Tools 11.0.2 3PDev.app (pid=0x5aff, version=11.0.2d626)
*** Starting Timestamp: Tuesday, January 7, 2014 4:10:57 PM Eastern Standard Time (89706938666 uS)
*** System Details: OS Version: 10.8.5, CPU Speed: 2.7 GHz, Architecture: Intel 64 bit, Num Processors: 8 log
*** DigiTrace Config File: /Applications/Pro Tools 11.0.2 3PDev.app/Contents/Resources/config.digitrace
*** Facilities to trace:

DTF_INSTALLED_COMPONENTS@DTP_NORMAL(0e0d)

Time(us),Tid,Facility,Name : Debug Message
-----
89707181683,00c07,0e0d: Pace eden lib version: 2.0.0, r22343 (2.0.0.22343), [...]
89707220338,00c07,0e0d: ShoeTool_Init - shoe tool installed version is 6.000 [...]
89707220374,00c07,0e0d: ShoeTool_IncreaseAIOLimits - var=46, newVal=512, cur [...]
89707220380,00c07,0e0d: ShoeTool_IncreaseAIOLimits - var=47, newVal=512, cur [...]
```

The log file consists of a header followed by a series of log statements. Each log statement includes the following information:

- Time(us) - The time the message was logged, in microseconds since the machine was started.
- Tid - The thread ID of the thread that logged the message.
- Facility - The Facility ID of the facility that's logging the message.
- Name - This is the config name added to all facilities included by this config file. This can be used to group all facilities related to a feature set, for instance. If not set, this is not included.
- Debug Message - This is the actual string passed to the trace facility.

12.42.5 Configuring DigiTrace

You can configure DigiTrace to include or exclude specific traces using the config.digitrace configuration file. This file is plain text and includes a single configuration command on each line.

This is the basic format for a command used to enable tracing for a single **facility**:

`facility=[console@minimum console logging priority],[file@minimum file logging priority]`

Here are some examples:

- `DTF_APP_VERSION=file@DTP_LOW`

- DTF_PLUGINS_3P=file@DTP_LOW, console@DTP_URGENT
- DTF_ASSERTHANDLER=console@DTP_URGENT
- DTF_DAE_MEM=console@DTP_URGENT, file@DTP_LOWEST

For more information about special configuration commands, see [Advanced DigiTrace configuration](#).

12.42.5.1 Trace facilities

Trace facilities are used by DigiTrace to determine whether or not the given trace statement should be displayed. Trace facilities allow the user to filter trace statements at the component level.

12.42.5.2 Trace priorities

Trace priorities are used by DigiTrace to determine whether or not the given trace statement should be displayed. DigiTrace specifies five trace priorities:

- DTP_LOWEST
- DTP_LOW
- DTP_NORMAL
- DTP_HIGH
- DTP_URGENT

The DigiTrace configuration file specifies minimum trace priorities. For example, if a trace statement uses DTP_LOW and DigiTrace is configured to use DTP_NORMAL as the minimum trace priority, then the trace statement will not be sent to the output target. In general, the DTP_LOWEST priority setting will populate the trace output with the most verbose information while the DTP_URGENT setting will output only the most high level details.

12.42.5.3 Useful DigiTrace facilities

This section includes descriptions of several facilities that are used in Pro Tools and other Avid audio products. The logging provided by these facilities may be helpful when diagnosing plug-in issues.

- DTF_AAXPLUGINS This is the standard facility for [AAX](#) plug-ins. This facility will only log traces that are present in [AAX](#) plug-ins themselves, not traces in any hosting code. Plug-ins may use the [AAX_TRACE](#) or [AAX_TRACE_RELEASE](#) macros to log to this facility.

Note

Disabling the DTF_AAXPLUGINS facility will slightly reduce the overhead of trace statements and chip communication on HDX systems.

- DTF_AAXHOST at DTP_NORMAL Logging from the main [AAX](#) host component. Use a lower priority for additional [AAX](#) Host tracing.
- DTF_PLUGINS at DTP_LOW Miscellaneous plug-in operations, including page table logging, preset directory errors, and DLL loading and unloading
- DTF_TIPPLUGINS at DTP_NORMAL Logging for HDX plug-in algorithm handling details such as packet management and private data field state reset. Use DTP_LOW for deeper tracing.
- DTF_TISHELLMGR at DTP_HIGH Logging from the HDX RTOS
- DTF_DAE_HOSTDEVICE at DTP_URGENT Performance logging from the real-time audio render thread. See [Real-time AAE performance logging with DigiTrace](#)

- **DTF_DAE_ERRORS** at **DTP_NORMAL** or **DTP_LOW** Information about any errors that occur in AAE. Use **DTP_LOW** to enable stack traces.
- **DTF_ASSERTHANDLER** at **DTP_NORMAL** or **DTP_LOW** Similar to **DTF_DAE_ERRORS**: Information about any asserts that fail. Use **DTP_LOW** to enable stack traces.
- **DTF_THREAD_NAMES_AND_PRIORITIES** at **DTP_NORMAL** or **DTP_LOW** Allows you to look up a thread's debug name from its ID on the standard trace line. Thread names and IDs will be traced as they are created, and you can then use that ID to resolve the thread name of later trace statements. Use **DTP_NO_RMAL** for just names and IDs, and **DTP_LOW** to include priorities.
- **DTF_PACESUPPORT** at **DTP_NORMAL** Plug-in digital signature logging, with some diagnostics for digital signature verification failures.
- **DTF_ADC** at **DTP_NORMAL** Delay compensation logging, including host accounting for plug-in latency.
- **DTF_AUTOMATION** at **DTP_LOW** Parameter touch and release logging.
- **DTF_AUDIOSUITE** at Logging of events specific to AudioSuite plug-in instances.

12.42.6 Bonus features

12.42.6.1 Real-time AAE performance logging with DigiTrace

Pro Tools 11 and higher includes logging for audio render callback performance. To enable this logging, enabled the **DTF_DAE_HOSTDEVICE** facility at **DTP_URGENT**. This facility will enable logging of real-time audio render thread metrics around any render errors that occur.

Here is an example of a performance log:

```
Int(LL): hstEr=0, ioEr=0, dif=2665(2891,23129), tot=2517(1648,2317), in=51(58,83),
clbk=2158(1508,2158), out=108(99,164), offset=[12], mxW=1101(com.avid.aax.eleven.free)
```

The different values included in this log are:

- hstEr
- ioEr
- dif
- tot
- in
- clbk
- out
- offset

A log of 'x=a (b,c)' means that the (x) value (e.g. tot) for the interrupt was (a) us, the running average was (b) us, and the maximum value encountered was (c) us. Therefore, in the example above:

- 1648 us average total time was spent in each interrupt
- 2517 us was spent in this interrupt
- Eleven Free was the longest worker in this interrupt

In practice, it is difficult to precisely log this information during an error. This is due to changes in the interrupt pattern and scheduling when the audio engine is halted. In order to account for this, the performance logging will print out logs for several interrupts around when any error occurs. The actual audio engine error (hstEr) may be reported for a "junk" interrupt cycle that is spuriously logged during this halt process.

12.42.6.2 Adding signposts to the DigiTrace log at run-time

Use the shift-'~' key combination to add a "Trace Flag" line into the DigiTrace log. This allows you to add a "signpost" line to the log right when an important event happens, or before/after an important operation, so that it is easier to find the important details when inspecting the log later.

```
176603608088,2b603,0016: DSK_PrePrimeDiskTask::PrePrimeDiskTask - finish
176603961348,00307,0000: Trace Flag 3 (diff prev: 2.40s, diff start: 7.18s)
176605252296,00307,0000: Trace Flag 4 (diff prev: 1.29s, diff start: 8.47s)
176605252779,00307,0f09: 2016-07-19 23:36:32.499 PTC_Mgr::Idle() -- performing task (Websocket Base)
176606039830,00307,0000: Trace Flag 5 (diff prev: 0.79s, diff start: 9.26s)
```

Each trace flag signpost includes the text "Trace Flag" and the diff (in seconds) from both the previous trace flag and the first trace flag which was triggered during the current run of the app.

These lines will be printed regardless of the current DigiTrace configuration settings.

Host Compatibility Notes This feature is available in Pro Tools 12.6 and higher

12.42.7 Adding traces to an AAX plug-in

12.42.7.1 Basic AAX logging

Standard `printf`-style logging from [AAX](#) plug-ins is very easy. This feature is built into the [AAX](#) specification and is exposed to plug-ins via the [AAX_TRACE](#) and [AAX_TRACE_RELEASE](#) macros. For more information about basic logging, see the documentation for those macros.

Note

To enable basic [AAX](#) logging via these macros, the `DTF_AAXPLUGINS` trace facility must be enabled.

12.42.7.1.1 Tracing for AAX DSP

The [AAX_TRACE](#) and [AAX_TRACE_RELEASE](#) macros, as well as [AAX_ASSERT](#), are cross-platform and are supported for use in [AAX](#) DSP algorithms. For more information about tracing from [AAX](#) DSP algorithms, see the [Tracing](#) section in the [TI Guide](#).

12.42.7.2 Advanced DigiTrace logging features

As a developer, you can use several advanced macros to extend the functionality of DigiTrace logging in your plug-in beyond the simple `printf`-style features provided by [AAX_TRACE](#). The full DigiTrace macro suite includes macros for stack traces, very long traces, or even the ability to dump a block of memory to the log.

Note that these advanced features are only available on the host system. They are not currently available from algorithms running on embedded hardware.

12.42.7.2.1 What files do I include in my project?

To add advanced DigiTrace instrumentation to your source code you must:

1. Include `DigiTrace.h` in the file where you are going to put your trace statements.
2. Compile `CDigitraceAccess.cpp` into your project

If you have problems including the DigiTrace header file, try moving it to the top of the file that you're including it into. DigiTrace has no other dependencies and should be safe to include into any component.

12.42.7.2.2 What do I do if I encounter problems compiling or linking?

Should you run into linker errors or other problems after adding the DigiTrace header file, please go through the following items and verify that each is included in your project:

- The CDigitraceAccess.cpp file automatically searches for and loads the DigiTrace.dll (Windows) or DigiTrace.framework (Mac) component and ensures that the appropriate function pointers are initialized. If you receive linker errors, the missing symbols are likely in this file. Note that your project will need the path to CDigitraceAccess.h in order to compile this file.
- If you receive an include file error for DigiPragmas.h then you will need to add the header's path to your project's search paths. This file is included in the most recent DigiTrace Tools packages but may not be included in some older packages.

12.42.7.2.3 Macros

DigiTrace provides five core macros for trace output, which are:

- TRACE_R - for general printf style tracing. Subject to a total line limit of 256 chars.
- TRACE_PUTS_R - prints an arbitrary length buffer, splitting it up into clean lines based on line breaks. No formatting.
- STACKTRACE_R - for stack trace printing. See below.
- MEMTRACE_R - for memory buffer hex tracing
- FXTRACE_R - for automatic function entry and exit tracing

12.42.7.2.4 Debug vs. release macros

All of the macros listed above are general-use macros, which generate output in both Debug and Release builds. They each have a debug-only variant which excludes the trailing "<TT>_R</TT>". Like [AAX_TRACE](#), these debug-only versions compile to a noop in release builds.

The use of debug-only macros is not usually necessary due to the fact that release traces are encrypted and hidden from end users (but not from other developers.) As a best practice, we recommend using the "_R" version of a macro whenever possible. The debug versions of the macros should only be necessary in special circumstances where you specifically do not want to compile the code into release builds.

For more information about tracing in release builds see [Security concerns](#)

12.42.7.2.5 Syntax

Adding a DigiTrace statement to your code is as easy as making a single function call thanks to DigiTrace's predefined macros. The basic macro syntax is:

```
MACRO_NAME( TRACE_FACILITY_NAME [| TRACE_PRIORITY_LEVEL], MESSAGE_STRING )
```

Here is a code sample:

```
bool my_function(char* data_buffer, int data_buffer_len)
{
    FXTRACE_R( DTF_PLUGINS_3P, "my_function" ); // Automatically trace function entry and exit.
    // You need only to specify a trace facility.

    OSERr err = FrobinateBuffer(data_buffer, data_buffer_len);
    if( noErr != err )
    {
        TRACE_R( DTF_PLUGINS_3P | DTP_HIGH, "Couldn't frobnicate the buffer: %s", OSERrToString(err) );
    }
    else
    {
        int cBytesToTrace = 64;
        MEMTRACE( DTF_PLUGINS_3P | DTP_LOW, "Data after frobination", data_buffer, cBytesToTrace );
    }
}
```

12.42.7.2.6 Generating stack traces

The `STACKTRACE_R` macro is very useful for getting stack traces of important events in the code like throwing errors (which can be thrown from many locations). One particularly useful feature of this macro is that it allows you to specify a facility and priority for the `printf` part of the stacktrace, e.g. `DTP_NORMAL`, and another one for the stacktrace step, e.g. `DTP_LOW`. See `DigiTrace.h` for a full list of macros and their documentation.

12.42.7.2.7 Turning off tracing in a specific file

You can explicitly disable tracing in an instrumented file in the build by defining the `MTurnDbgTraceOff` symbol at the top of the file.

12.42.7.3 Security concerns

Unless you provide your own logging encryption, DigiTrace logs are not secure and should not be used to store any sensitive information.

Logs generated by Pro Tools release builds on users' systems are encrypted. This is primarily for the sake of avoiding confusion in our user community, since DigiTrace logs can be cryptic and potentially misleading for users who are not familiar with our code.

Avid and other third-party developers will see your plug-ins' release trace statements if they load your plug-ins with the appropriate trace facilities enabled. We highly recommend that you keep this in mind when developing your trace statements, both in order to prevent confusion (see the formatting guidelines in the [AAX_TRACE_RELEASE](#) documentation) and in order to maintain the security of your code.

12.42.8 Advanced DigiTrace configuration

The basic configuration command to enable tracing for a facility is described above in [Configuring DigiTrace](#).

There are also additional commands that can be added to the DigiTrace configuration file for more advanced configurations.

12.42.8.1 Configuration command format

- All DigiTrace configuration commands are listed in the configuration file with the form `<token>=<value>`
- Any blank line or line beginning with a '#' character is ignored
- Tokens are not case sensitive
- If there are repeated tokens in a file, the last token wins

12.42.8.2 Advanced configuration commands

- `FileTracingDir = { DIRPATH }`
 - Default: `USE_RELATIVE_PATH`
 - Custom log file directory. e.g. "C:\MyTraceDir" on Windows or "~/MyTraceDir" on OS X.
 - If `DIRPATH == USE_RELATIVE_PATH` then the output trace file directory will be created next to the target application.
- `Append = { true | false }`
 - Default: `false`
 - Append to file. If `true`, the output of this trace will be appended to any existing log file with the same name. Otherwise, this trace will overwrite an existing log file with the same name.
- `LogFileLimit = { LIMIT }`

- Default: no limit
 - Limits the number of log files kept around for this config file to the specified number.
 - If set to an integer value, DigiTrace will delete the oldest log file(s) until there are only N most recent log files in the output folder.
 - If you rename an output file so it does not have the standard prefix, it is never deleted by this option.
 - Does not work with the "append" option
- TraceQueueSize = { small | medium | large }
- Default: small
 - This controls the amount of memory allocated to the trace queue. You probably won't need to change it.
- BeQuiet = { true | false }
- Default: false
 - "Quiet" mode. If set to true, this configuration option makes all trace output occur without any decoration (i.e. no timestamps, no thread id, no process id, etc.).
 - This mode may be useful for some types of real-time vector tracing or for configuring formatted logs for post-processing with a text editor.
- FileId = { FILEID }
- Default: none
 - If set, this string is included in the filename created by DigiTrace for trace output files.
 - Does not work with the "append" option
- Name = { NAME }
- Default: none
 - If set, this string is included in every trace for all the facilities that are enabled in this config file.
 - You can have one of these per config file.

12.42.8.3 Dynamically changing the DigiTrace configuration

DigiTrace config files can be loaded dynamically, which means that you can add new configs while the instrumented application is running. Below are the details surrounding dynamic loading:

- In debug builds, this will happen each time the app comes to the foreground.
- The API only does anything if something has changed in your config files that will result in different tracing of some sort. If nothing has changed, the overhead to make the call is < 1ms, and current tracing is not affected.
- If something has changed, the changes are merged in to the existing config objects in memory. Active log files are not interrupted when possible. This takes around 200ms (mostly because of a sleep command that lets threads finish tracing things). Traces that happen in threads during this changeover will be dropped.
- No code in the actual tracing commands was changed.
- You can change any of the attributes of the trace facilities (priority level, file or console, etc).
- You can add new config files on the fly, or if you start with no config file, you can add one on the fly.

12.42.9 Compatibility

DigiTrace is an internal testing and troubleshooting tool. Although we will try to provide up-to-date documentation so that third-party developers can use this tool, we may need to change the way that DigiTrace works at some point and so we cannot make any promises regarding forwards-compatibility.

At the time of this writing:

- DigiTrace is fully compatible with Pro Tools 8.0.3 and later. DigiTrace is installed by default with compatible Pro Tools shipping versions and with some Pro Tools Development Builds.
- DigiTrace is compatible with all versions of the DAE dish in the DSH environment. Tracing must be explicitly enabled in the dsh executable by placing a config.digitrace config file next to the executable.

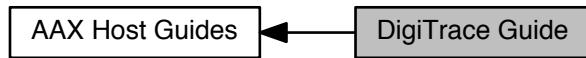
If you notice significant bugs or other problems with DigiTrace in any Pro Tools release then we encourage you to report the issues to us on the developer forum. We may not be able to address issues immediately, but your feedback is appreciated.

12.42.10 Additional Information

12.42.10.1 Confidentiality

As with all information provided in the [AAX SDK](#), the information provided in this documentation is confidential and is bound by the terms of your NDA with Avid. You may provide customized DigiTrace configuration files to end users in order to generate useful debugging information on their systems. However, you may not provide users with decrypted DigiTrace logs or with other details provided in this DigiTrace documentation.

Collaboration diagram for DigiTrace Guide:



12.43 DSH Guide

How to test basic functionality of AAX plug-ins using DSH test tool.

12.43.1 Contents

- [What is DSH and how it works](#)
- [Basic set of commands of the DAE dish](#)
- [Basic plug-in tests](#)
- [Debugging and tracing in DSH](#)
- [Scripting interface and batch profiling](#)

12.43.2 What is DSH and how it works

DigiShell is a software tool that provides a general framework for running tests on Avid audio hardware. As a command-line application, DigiShell may be driven as part of a standard, automated test suite for maximum test coverage. DSH supports loading all types of AAX plug-ins except AS, and is especially useful when running performance and cancellation tests of AAX-TI types. DigiShell is included in Pro Tools Development Builds as dsh.exe (Windows) or as dsh in the CommandLineTools directory (Mac).

After it is launched, DigiShell waits for a command name and parameters to be entered via stdin; command results are output via stdout. DigiShell parses its input as command name, followed by a single space, and then command parameters. The command parameters are expected to be a yaml-encoded string. Here are two examples of strings in compact (single-line) yaml format:

- A hash containing lists in compact yaml syntax { key1: [val1, val2], key2: [val3, val4] }
- A list of two lists [[PIO, 0, 1], [DSP, 1, 1]]

12.43.3 Basic set of commands of the DAE dish

DigiShell has built-in commands for getting help, creating a DigiTrace configuration and loading DigiShell modules known as "dishes". One can see the command list by running the help command without any parameters. Passing a command name as a single string parameter to the help command will give a more detailed command description.

The default installation of DigiShell includes a few dishes, including the DAE dish. This dish loads the AAE audio engine and can be used for loading and testing basic functionality of plug-ins. The DAE dish can also be used to load a plug-in for basic debugging purposes, and provides a higher-weight debugging environment than the full Pro Tools application.

Another dish supplied with DSH, the aaxh dish, provides a lower-level hosting environment for [AAX](#) plug-ins. This dish loads a dedicated [AAX](#) host component without audio routing logic or other audio engine responsibilities. In this guide we will focus on loading and running plug-ins using the DAE dish, but we encourage you to also explore the commands available in the aaxh dish and to learn how to exercise your plug-ins in that environment.

12.43.3.1 Loading plug-ins in DSH

The following commands can be used to load and configure the DAE dish:

- `load_dish DAE` Loads the DAE dish

- `init_dae 48000` This command is optional. It configures AAE to work at a specific sample rate. By default it will work at 44100 Hz.

Loading the DAE dish into the DigiShell environment with the built-in `load_dish` command will extend the set of available commands. Among them there is a `run` command. The `run` command can be used in two ways:

- Execute with no arguments: List all plug-in configurations which are available to AAE
- Execute with arguments specifying a particular plug-in configuration: Load a plug-in instance using the specified configuration

You can also search for the id and spec of the specific plug-in with the `findpi` command, which takes a plug-in's name or part of it as an argument, and then searches through the whole list of available plug-ins using this pattern.

Figure 1: DSH command for loading plug-ins.

If the plug-ins was instantiated successfully, then DSH will list all its parameters, just like on the screenshot. If instantiation fails, then DSH in most cases will output the error code, although it is not always obvious what this error code means. Here is the list of possible reasons of some failures:

- -9060 failed to load DSP Hybrid plug-in
- -14140 IO interface is not connected
- -7050 not enough resources for instantiating plugin
- -14378 plug-in exceeded memory limits
- -14003 something is wrong with your HDX card

-30xxx errors are dynamically-generated and can indicate different failures. Failures due to plug-ins exceeding the cycle limit of the DSP CPU will often appear as -30xxx errors. See [-30xxx: Dynamically-generated error codes](#) in the [TI Guide](#) for more information.

Note

`run` command works for Native and DSP plug-ins, but not for the Audio Suite ones. Also it will fail for DSP Hybrid plug-ins. To be able to instantiate them, one should run `acquiredeck` command.

There are several DAE dish commands for operating with plug-ins' instances:

- `getcurrentinstance` Returns the index of the current instance. The counting starts from 0 for the first instance that has been instantiated, and increments by one for every next instance.
- `getinstanceproperties` Returns the effect name for the Native plug-ins, and much more detailed info for the DSP instances.
- `setcurrentinstance` Sets the instance with the given index as the current instance.

12.43.3.2 Working with HDX card from DSH

One of the benefits of the DAE dish in DigiShell is that it has direct access to the shell environment that loads DSP plug-ins. The included facilities for retrieving load error information from the DSP manager can be very helpful for debugging DSP plug-in load failures. For example, you can use the following DAE dish commands to determine what resource requirement is preventing additional instances from loading onto a single DSP:

1. `reservetidsp all` Reserves all unused DSPs in the system
2. `unreservetidsp 0` Frees the first DSP for plug-in allocation
3. `getlastdsploaderror` Retrieve the text of the error that was generated when the final Effect instance attempted to load
4. `getdspinfo 0` Returns detailed info about the DSP chip with the given index. By executing this command you can figure out whether particular chip is in use currently, which plug-ins are instantiated there, how many resources they consume and how many resources are still available.

Figure 2: Info about the DSP chip with the given index.

12.43.3.3 DAE dish tips

- With the standard configuration, the system's AAX plug-ins folder will be used by DSH and DTT. To override this, create a folder named "Plug-Ins" next to the DTT and CommandLineTools directories. While that directory exists, AAE will only scan the plug-ins in the new Plug-Ins folder.

12.43.4 Basic plug-in tests

There are some basics tests that can be performed for AAX plug-ins in DSH. Among them are instantiation test, measuring of amount of processor cycles that DSP plug-in may consume on different settings, cancellation test and so on.

12.43.4.1 Cycle count performance test

Use the DAE.cyclesshared command in the DAE dish to profile a DSP algorithm's cycle count performance. This command measures both the shared and the per-instance cycles used by a plug-in, both of which must be reported to the host. This command also includes the option to load a custom plug-in preset so that various algorithm code paths may be exercised. It is important to report the maximum possible number of cycles that the plug-in may need, so that it had enough resources, even in the worst case. Otherwise one can obtain noise and clicks in the output audio on the extreme plug-in's settings.

The full syntax of this command is as follows: `cyclesshared <index -- spec -- {key:value, key:value, etc.}> index - Index of the plug-in as listed by the DAE.run command spec - Plug-in I↔D triplet in array, e.g. [AVID', 'DmGn', 'DGDT'] key:value hash options: idx - Index of the plug-in spec - Plug-in ID triplet array run_cached: <true -- false> - Whether to use cached code when measuring. Defaults to false. load_preset: <filename> - Load the specified preset for each instance before measuring performance adjust_controls: <true - false> - Randomly change the plug-in's parameter state before running the test`

Examples:

- `cyclesshared 21`
- `cyclesshared ['AVID', 'DmGn', 'DGDT']`
- `cyclesshared {spec: ['AVID', 'DmGn', 'DGDT'], load_preset: "mySettings.tfx"}`
- `cyclesshared {spec: ['AVID', 'DmGn', 'DGDT'], adjust_controls: true}`
- `cyclesshared {idx: 21, run_cached: true} Do not use cached measurements for reported cycle counts!`

Normal output of this command should look like this:

Figure 3: Normal cyclesshared command output.

Sometimes during the development process it may happen that this test fails, and the reason of such a failure can be different:

1. Plug-in exceeded the DSP chip's memory limit

Figure 4: Plug-in exceeded the memory limit.

2. Plug-in exceeded the processors cycles budget

- The number of instance and shared cycles looks acceptable, but expected number of plug-in's instances that can be instantiated on the chip/card at different sample rates is zero. Resultant cycle count can be used for calculating how much plug-in has exceeded the limit, and how much it should be optimized.

Figure 5: Plug-in exceeded the processor cycles budget.

- If plug-in exceeds the processor's cycles budget too much, then cyclesshared test will most likely output the warning that is highlighted with orange color on the screenshot below. Also the number of both instance and shared cycles will be shown as zero or one.

Figure 6: Plug-in exceeded the processor cycles budget very much.

3. Plug-in processing is not balanced.

If some big parts of code, which do not really depend on the specific plug-in settings, are located under condition structures, and if they make plug-in to do more calculations in one case and less in another case, then that means that plug-in's processing is not balanced. This may cause some problems, because it is hard to predict when this or that condition may become true and how much the amount of processor's cycles that plug-in needs will increase. So it is better to remove such conditional blocks and to perform those calculations every time, even if their result is not really needed in particular cases.

To indicate such situations the correlation coefficient can be used. If its value close to zero, then plug-in has the described problems.

Figure 7: Plug-in processing is not balanced.

12.43.4.1.1 Performance profiling and test signals

Some algorithms' performance characteristics are program-dependent, and in such cases use of the the cycles command alone may not be sufficient. To route a test signal to your plug-in while measuring cycles, use of the cycles command along with the load_wav_file command in the DAE dish. The basic approach is as follows:

- Use single-buffer manual processing, rather than continuous
- Split your test signal into several pieces, with each piece to be processed using different settings
- Loop on:
 - Load or adjust the PI's settings,
 - process the next piece of audio while measuring cycles

Example:

1. piproctrigger manual set to single-buffer processing
2. load_wav_file "testaudio_pt1.wav"
3. load_wav_file "testaudio_pt2.wav"
4. load_wav_file "testaudio_pt3.wav" load audio buffers; take note of return ...
5. run <mypluginIndex>
6. load_settings "mySavedSettings.tfx" load the settings OR control [1, 24] # set controls directly
7. cycles b1 measure cycles while processing first file
8. load_settings "mySavedSettings2.tfx" load the next settings
9. cycles b2 measure cycles while processing second file ... etc.

12.43.4.2 Cancellation test

When porting plug-ins from RTAS to AAX platform, or from 32-bit to 64-bit architecture, or from Native to DSP, it may be useful to compare output of two plug-in's versions to make sure that it is still the same and nothing has been broken. For this purpose DSH cancellation test can be used.

In the simplest case, when both plug-ins are present in the same version of Pro Tools (Native and DSP version of the same plug-in for example), then `diff` command can be used to perform the test: `diff [<spec1>, <spec2>, <frames>]` which reports the peak difference in the output amplitude of plug-ins `<spec1>` and `<spec2>` after processing `<frames>` frames of a 1 kHz full-scale sine wave. The maximum difference will be provided in dB.

Another way to perform the cancellation test is to process audio with each plug-in separately manually and to compare the result after that. This scenario allows you to load custom input audio file and special plug-in settings:

1. `piproctrigger manual` This command should be run for DSP plug-ins before loading them. When this option is set, DSP plug-in will start process audio only after piproc command is called. Otherwise it will start processing right after the instantiation process.
2. `run 81 Laoding plug-in`
3. `load_settings "/Users/settings/pitch_settings.tfx"` Loading settings file
4. `load_wav_file "/Users/audio_files/mono_file.wav"` Wav file will be loaded in a buffer, or in several buffer if it has more than one channel. Command will output references to the buffers, like `b1, b2 ...`

Note

It is not recommended to choose very long audio files for the DSP processing, since the test is very slow, and processing of 10 sec audio file may take up to 1 min depending on the complexity of the plug-in's algorithm.

5. `bclone b1` The easiest way to create the output buffer of the same size is to copy the input buffer. Command will also output references to the resultant buffers.
6. `piproc [b1, b2]` Command which actually is doing passes the input audio through the current plug-in, and is recording the result to the output buffer (which is `b2` here). For stereo plug-in command will look like `piproc [[b1,b2], [b3,b4]]`
7. `bfsave [b2, "/Users/saved_buffer"]` Output buffer can be stored to the disk. This may be needed for the cases, when one wants to compare the output of plug-ins, which can not be loaded in the same instance of the DSH. So the output of one plug-in can be saved to disk and then loaded later in the another instance of DSH. Also output buffer can be saved as .wav file with `save_wav_file` command.
8. `bfload "/Users/saved_buffer"` Saved buffer can be loaded again by this command. It will output the reference to the newly created buffer.
9. `bacmp [b1,b2]` This command will compare the contents of the buffers `b1` and `b2`. So it can compare the output buffers of two plug-ins and thus make the cancellation test.

12.43.5 Debugging and tracing in DSH

DSH provides a lighter-weight debugging environment than the full Pro Tools application. So it should be easier to step through the description code of the plug-in there, rather than in PT, because DSH does significantly less initialization work than PT during the loading process.

Also DSH is very useful in situations, when one wants to debug the plug-in's algorithm on a specific audio buffer. The only way to follow plug-in's algorithm work step-by-step on the certain piece of audio is to debug the `piproc` command.

DSH supports tracing, which is based on Avid's DigiTrace. To enable trace logs in DSH, one should create a `dsh.digitrace` config file for it and put it next to `dsh` executable file. It can be the same as `.digitrace` file for the Pro Tools. DSH has built-in commands to generate a DigiTrace config file. The `clear_trace_config` command creates (or clears if it already exists) a DigiTrace config file. The `enable_trace_facility` command enables logging of a specified facility/priority pair.

Note

On the Mac, DigiShell must be relaunched before a new DigiTrace configuration will take effect.

12.43.6 Scripting interface and batch profiling

DigiShell can be scripted using DishTestTool, a Ruby-based command line tool. More details can be found in [DTT Guide](#).

Collaboration diagram for DSH Guide:



12.44 DTT Guide

How to automate different test scenarios for DSH.

12.44.1 Contents

- [What is DTT](#)
- [How to run tests and suites in DTT](#)
- [Writing DTT scripts](#)
- [Logging in DTT and debugging DTT scripts](#)
- [Working with DTT test suites](#)

12.44.2 What is DTT

DishTestTool (DTT) is a Ruby-based command line tool, which provides the ability to script and thus automate DSH test scenarios. It is included in the DigiShell Tools package in the /DTT directory.

Note

Ruby is installed by default on OS X. On Windows, you will need to install Ruby and add it to your PATH variable manually. For information regarding Ruby version compatibility with a specific build of DTT, see the ReadMe.txt file in /DTT/sources.

The DTT folder consists of:

- *Sources*
 - DTT core
 - *scripts* folder - place all DTT script files here
 - By default, this folder includes a few example scripts demonstrating basic DTT operations and plug-in testing steps:
 - * *DSH_SigCancellation.rb* - script for the cancellation test
 - * *DSH_TI_CycleCounts* - script for performing cycle count test
 - * *SuiteGenerator.rb* - generates suites for the DSH_SigCancellation and DSH_TI_CycleCounts tests
 - *suites* folder - place your DTT suite files here
 - *run_test.command* (on Mac) or *run_test.bat* (on Windows) - command file for running tests
 - *run_irb.command* (on Mac) or *run_irb.bat* (on Windows) - command-line interpreter

12.44.3 How to run tests and suites in DTT

run_test is the main DTT execution program. *run_test* is able to execute Ruby scripts which have been placed in the *scripts* folder within the DTT directory.

- *run_test.command -l* - lists all the available scripts and suites
- *run_test.command 1* - runs test by number
- *run_test.command DSH_SigCancellation* - runs a test by name. Pay attention that the name of test should be without (!) extension.

- `run_test.command -script DSH_SigCancellation -a sample_rate=48000 -a threshold=-80` - runs a test with test script arguments, which are specified using the `-a` option

Figure 1: running DTT tests.

For more information about script arguments, see [Describing and using input arguments of the script](#)

12.44.4 Writing DTT scripts

Most of the DTT scripts require `DigiShell`, which allows them to run `dsh` and execute different `dsh` commands. Each script should be represented in the form of class, which inherits `Script` class, and also each script must have at least two elements: `self.inputs` section, where all the input arguments of the test should be described, and `run` method, which is the main body of your script.

```
require 'DigiShell'

class ScriptSample < Script
  def self.inputs
    return {}
  end

  def run
    return pass("Well, it didn't explode. So that's something.");
  end
end
```

Listing 1: Skeleton of the script

12.44.4.1 Describing and using input arguments of the script

The available parameters and their values for a script are listed in the static `self.inputs` routine. Input arguments must be organized in the form of a hash map which is returned from this routine.

```
def self.inputs
  return {
    :sample_rate      => [44100, [44100, 48000, 88200]],
    :path_to_tfx     => ['none'],
    :threshold        => [-96],
  }
end

@dsh.init_dae(sample_rate)
```

Listing 2: Describing input arguments for the script and using them

Hash entries should be in the following format: `:arg_name => [default_value, [range of allowed values]]`

These arguments can be used then by just calling them by name, like in the example above with `sample_rate` argument.

12.44.4.2 Writing body of the script

The body of the script must be enclosed in the body of the `run` method of the script class. As far as most DTT tests need DSH, in the example below it's shows how to create a DSH instance in the script and how to use it then. DSH instance can be created with `DigiShell.new` method, which requires `DigiShell` module, as has already been said. Then all the DSH commands become available as methods of the DSH instance, and input arguments can be passed to these command as input arguments for the methods, i.e. in parentheses `dsh.load_dish("DAE")`. Also it's recommended to handle possible exceptions that may occur during the execution of the code, and to make sure that DSH has been closed, if it was instantiated on the moment of the failure.

```
def run
begin
  dsh = DigiShell.new(target)
  dsh.load_dish("DAE")
```

```

dsh.init_dae(sample_rate)
dsh.close

return pass("Well, it didn't explode. So that's something.")

rescue Exception => e
  # make sure to close down dsh before returning
  if (dsh)
    dsh.close
  end

  return fail(e)
end
end

```

Listing 3: Example of the body of the script.

12.44.5 Logging in DTT and debugging DTT scripts

DTT tool has logging, and all the logs collected in the Logs folder, which is located in root directory of the DTT. DTT creates a separate folder for each test and names these folders with the corresponding names + the time when the particular test has been executed. For example:

DSH_SigCancellation_20131225_185146_0001

Inside each folder there are several log files:

- *xxx.html* – contains info about input & output of the test in a fancy form (tables)
- *xxx_c.txt* – contains the list of the DSH commands that have been executed
- *xxx_d.txt* – DSH output
- *xxx_i.txt* – info about your system
- *xxx_l.txt* – standard output
- *xxx_v.txt* – verbose output

12.44.5.1 Interactive mode

There is an option to run DTT in interactive mode using interactive ruby shell (irb). When running in this mode, DTT creates a shell which is an extended version of the standard Ruby interpreter. Besides the standard functionality, it knows how to work with DTT classes and can give hints on their methods. In particular, the DTT interactive mode shell knows how to work with DigiShell.

To run DTT in interactive mode, go to the DTT folder and launch the `run_irb` program. At this point you will send ruby commands to dsh through the pipe in YAML format:

```

t = Target.new # creates an instance of Target. In this case target is a local machine, though we have a
               # possibility to run test on remote machine.
dsh = DigiShell.new(t) # creates an instance of DigiShell(aka launching dsh binary)
dsh.load_dish('DAE') # Loads 'DAE' dish
dsh.help('init_dae') # Requests help from dsh for 'init_dae' command
dsh.init_dae
plugins = dsh.run #returns an array of plug-ins and writes them to plugins var.
plug-ins[0] #reaching first plug-in from the list
#... whatever you want to do

```

Listing 4: Running DTT in interactive mode

12.44.6 Working with DTT test suites

Suites are files which contain the list of DTT scripts that should be run, and parameteres for these tests. These files should be created in YAML format. The list of the tests should be preceded by `tests:` line. Then tests to be run should be described as a map with the following members:

- name: - name of the test
- enabled: - determines whether test will be run or skipped
- args: - input parameters of the test

The input parameters of the test should be organized as a hash map. That means that all keys should start with ":" and look like ":plugin_spec:".

Also suite may contain a section with the general parameters like:

- verbose: `false` which determines whether the output of the test in the console will be verbose or not.
- timeoutFactor: `"16.0"` which defines the time period, after which test will exit in case it stuck on the execution of the certain piece of code.

Example of the suite:

```
suitesettings:
  verbose: false
tests:
#
# Cycle counting test
#
- name: DSH_TI_CycleCounts
  enabled: true
  args:
    :plugin_spec: 'Digi,Pich,Psmm'
    :sample_rate: 48000
```

Listing 5: Example of the DTT test suite

Note

All the suites files should have an extension .gss

12.44.6.1 Autogeneration of the suites

Sometimes it is necessary to generate the suites for the particular script for all the plug-ins from the bundle and/or for different sample rates. In this case instead of the copy-paste, which may lead to some mistakes and errata, suitegenerator can be used. This is a special script, which takes as arguments the name of the script(s), for which the suites should be generated, and the list of their input parameters. Strange as it may sound, this data should be formed as a suite. Script itself is available as SuiteGenerator.rb along with other scripts in the DTT.

Note

`SuiteGenerator.rb` can generate suites only for the two tests: `DSH_SigCancellation` test and `DSH_TI_CycleCounts` test

Here is the example of how to use this script to generate the suites for all the plug-ins from the 'D-Verb' bundle for all the sample rates the cancellation test AAX Native vs AAX DSP, and for the cycle counts test:

```
tests:
# Generate suite for Cancellation test: AAX Native vs AAX DSP
- name: SuiteGenerator
  args:
    :plugin_name: 'D-Verb'
    :path_to_audio_files: /Volumes/G_Audio/GS_Test_Resources/audio/
    :path_to_presets: /Volumes/G_Audio/GS_Test_Resources/PL_Settings/
    :test_script: DSH_SigCancellation
  enabled: true
# Generate suite for Instance Count test
- name: SuiteGenerator
  args:
    :plugin_name: 'D-Verb'
    :path_to_presets: /Volumes/G_Audio/GS_Test_Resources/PL_Settings/
    :test_script: DSH_TI_CycleCounts_se
  enabled: true
suitesettings:
  verbose: false
  timeoutFactor: "16.0"
```

Listing 6: Example of how to use SuiteGenerator script for generating suites for all the plug-ins from the 'D-Verb' bundle for the all sample rates for the cancellation and for the cycle counts test.

To generate the suites, one should run this suite for the SuiteGenerator as an ordinary suite by executing the `run_test.command <the name of the suite for the SuiteGenerator>` command. All the suites will be generated into the single file, which will be located inside the suites folder, and will be called like:

```
dspVSnative(optional)_<the name of the plug-in>_<the name of the test>.gss
```

Examples:

- TRIM_DSH_TI_CycleCounts.gss
- dspVSnative_TRIM_DSH_SigCancellation.gss

Collaboration diagram for DTT Guide:



12.45 Extensions

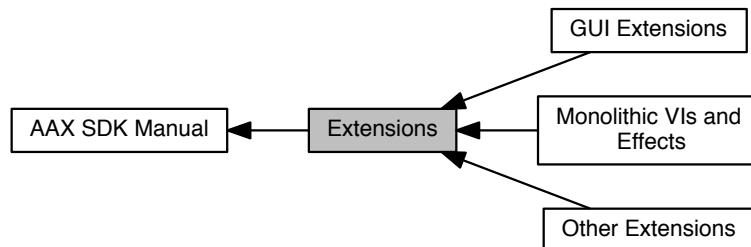
12.45.1

Extensions to the AAX SDK.

Documents

- [GUI Extensions](#)
GUI Extensions for the AAX SDK.
- [Monolithic VIs and Effects](#)
Extension of the `AAX_CEffectParameters` class for monolithic VIs and effects.
- [Other Extensions](#)

Collaboration diagram for Extensions:



12.46 GUI Extensions

GUI Extensions for the AAX SDK.

12.46.1 About the SDK's GUI Extensions

The code and projects in the SDK's Extensions/GUI/ directory demonstrate how to extend the AAX SDK's GUI programming interface using a variety of popular GUI frameworks, including:

- Native Cocoa
- Native Win32
- VSTGUI
- JUCE

These projects do not represent core functionality of the AAX SDK, but rather they serve as examples of how plug-in GUIs can be written to the AAX specification using a variety of different approaches.

12.46.2 Notes

- The VST and JUCE GUI Extension library projects use a macro value to resolve file paths to the installed framework directory. This macro is defined in a Visual Studio property sheet on Windows and as a custom project variable on Mac. Because this macro will not be resolved on Mac until compilation, the Xcode GUI will not be able to find the included files. However, the projects should build successfully once the macros are updated to point to the correct directory.
- The JUCE GUI Extension code in this SDK was written using version 1.51 of the JUCE framework
- Several VSTGUI-based headers with an "_ext" filename are provided along with the SDK's GUI Extensions code. These headers are slightly modified versions of the corresponding headers that are distributed with VSTGUI. The headers have been modified for use with our example plug-ins because we had several problems when using the VSTGUI SDK for 64 bit. For example, we encountered conflicting typedefs redefinitions like int32_t etc., which are preprocessed out of the vstguibase_ext.h file. These headers should not be required to build a 32-bit plug-in and they were only added in our transitional AAX SDK, version 1.5.

Collaboration diagram for GUI Extensions:



12.47 Monolithic VIs and Effects

Extension of the [AAX_CEffectParameters](#) class for monolithic VIs and effects.

This extension to [AAX_CEffectParameters](#) adds some conveniences for Virtual Instrument (VI) plug-ins and for other plug-ins that use a monolithic processing object, i.e. an object that combines state data with the audio render routine in a single object.

- The [RenderAudio](#) method provides a direct audio processing callback within the data model object. Perform all audio processing in this method.
- The [StaticDescribe](#) method establishes a generic MIDI processing context for the Effect. Call this method from the plug-in's [Description callback](#) implementation.
- The [AddSynchronizedParameter](#) method provides a mechanism for synchronizing parameter updates with the real-time thread, allowing deterministic, accurate automation playback. For more information about this feature, see [Fixing timing issues due to shared data](#)

Note

This convenience class assumes a monolithic processing environment (i.e. [AAX_eConstraintLocationMask_DataModel](#).) This precludes the use of [AAX_CMonolithicParameters](#)-derived Effects in distributed-processing formats such as AAX DSP.

[AAX_CMonolithicParameters](#) Collaboration diagram for Monolithic VIs and Effects:



12.48 Other Extensions

12.48.1

MIDI logging utilities

- void [AAX::AsStringMIDIStream_Debug](#) (const [AAX_CMidiStream](#) &*inStream*, char **outBuffer*, int32_t *inBufferSize*)

Filesystem utilities

- bool [AAX::GetPathToPlugInBundle](#) (const char **iBundleName*, int *iMaxLength*, char **oModuleName*)
Retrieve the file path of the .aaxplugin bundle.

12.48.2 Function Documentation

12.48.2.1 void AAX::AsStringMIDIStream_Debug (const AAX_CMidiStream & *inStream*, char * *outBuffer*, int32_t *inBufferSize*)

Print a MIDI stream as a C-string

Sets an empty string in release builds

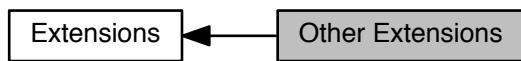
12.48.2.2 bool AAX::GetPathToPlugInBundle (const char * *iBundleName*, int *iMaxLength*, char * *oModuleName*)

Retrieve the file path of the .aaxplugin bundle.

Parameters

in	<i>iBundleName</i>	<ul style="list-style-type: none"> • OS X: The <code>CFBundleIdentifier</code> value set in the plug-in's .plist file • Other: This parameter is ignored
in	<i>iMaxLength</i>	
out	<i>oModuleName</i>	A preallocated buffer of size <i>iMaxLength</i>

Collaboration diagram for Other Extensions:



12.49 Supplemental Information

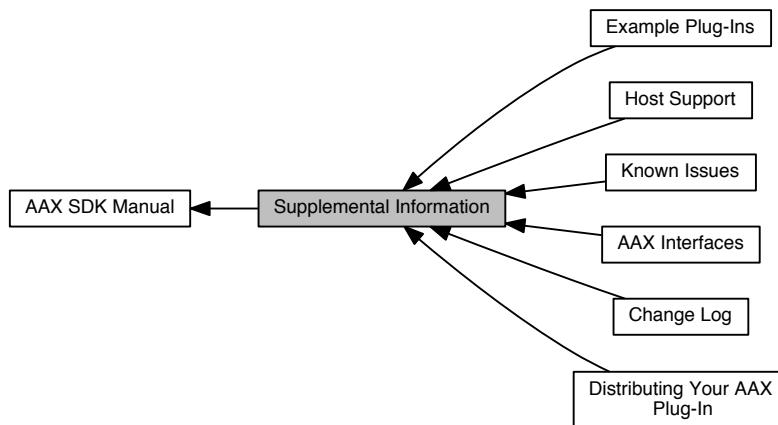
12.49.1

Supplemental documents beyond the scope of the AAX SDK.

Documents

- [Distributing Your AAX Plug-In](#)
Details about packaging and distributing your AAX plug-ins.
- [AAX Interfaces](#)
Full list of AAX interfaces.
- [Host Support](#)
Supported features in each AAX host.
- [Known Issues](#)
A list of known bugs affecting AAX plug-ins.
- [Change Log](#)
Changes between AAX SDK versions.
- [Example Plug-Ins](#)
Descriptions of the SDK's example plug-ins.

Collaboration diagram for Supplemental Information:



12.50 Distributing Your AAX Plug-In

Details about packaging and distributing your AAX plug-ins.

12.50.1 Contents

- [The finishing touches](#)
- [Building your plug-in installer](#)
- [Testing your plug-in](#)
- [Selling your plug-in](#)

12.50.2 The finishing touches

You've completed your main development work and your new AAX plug-in is nearly ready to ship! Now it's time to put the polish on your release.

12.50.2.1 Check and finalize page tables

After development has completed on your plug-in, we recommend that you check and finalize the plug-in's page tables using the [Page Table Editor](#) tool. It can be easy to forget to update the plug-in's page tables after making changes to the plug-in's list of parameters or to other aspects of the plug-in during development. To check for problems, open and view the plug-in's page tables for every layout in the editor app. Verify that the plug-in parameters are arranged properly for each control surface and that the list of available parameters in each layout is correct.

Correct and complete page tables are an important part of the user experience for many AAX plug-in users, and your users will appreciate your attention to detail here!

12.50.2.2 Create factory presets

Each AAX plug-in may be bundled with a set of factory presets. These presets will be made available to users through the host application's plug-in preset management UI.

Plug-in factory presets are stored as .tfx settings files. These files can be generated from any AAX host application which supports plug-in preset management. For example, in Pro Tools it is possible to create a new .tfx settings file by following these steps:

1. Create an instance of your plug-in in a Pro Tools session
2. Manually apply the desired preset settings
3. Choose "Save Settings As..." from the Presets drop-down menu in the plug-in window header

Once you have saved your desired factory presets as .tfx files onto your system you can package them with your plug-in bundle in *.aaxplugin/Contents/Factory Presets. Any presets found in this directory will be copied to the plug-in settings location for the running instance of Pro Tools when Pro Tools scans the plug-in on launch. See [.aaxplugin Directory Structure](#) for more information about supported sub-directories within the .aaxplugin bundle.

The feature for automatically copying factory presets from the .aaxplugin bundle to the plug-in settings directory on the user's system is supported by Pro Tools 11 and later and by all versions of Media Composer with AAX plug-in support.

Plug-in installers for 32-bit plug-ins supporting Pro Tools 10.3.5 and earlier must copy the settings to the plug-in settings folder when the plug-in is installed.

These are the paths for plug-in settings used by Pro Tools and Media Composer versions which support 32-bit AAX plug-ins:

- Mac: /Library/Application Support/Digidesign/Plug-In Settings
- Win: C:\Program Files(x86)\Common Files\Digidesign\DAE\Plug-In Settings

For more information about using plug-in presets in the various AAX hosts, see the following pages in the documentation for each host:

- [Pro Tools](#)
- [Media Composer](#)
- [VENUE](#)

12.50.2.3 Sign your plug-in

Pro Tools requires that all AAX plug-ins be signed with a digital signature. The certificate authority for this signature is PACE Anti-Piracy, Inc. and all AAX plug-ins for Pro Tools must be signed with the digital signing tools from PACE. See the [Digital signature](#) section in the [Pro Tools Guide](#) for more information about this requirement.

12.50.3 Building your plug-in installer

Your plug-in installer should place all .aaxplugin bundles into the system's AAX Plug-Ins directory:

- OS X: /Library/Application Support/Avid/Audio/Plug-Ins
- Windows (32-bit plug-ins): C:\Program Files (x86)\Common Files\Avid\Audio\Plug-Ins
- Windows (64-bit plug-ins): C:\Program Files\Common Files\Avid\Audio\Plug-Ins

This directory is searched recursively, so AAX plug-ins may be installed into sub-directories. For example, you may install all AAX plug-ins into a new sub-directory labelled with your manufacturer name.

12.50.3.1 Installing Track Presets

The Track Presets feature in Pro Tools allows users to recall entire tracks, or entire sets of tracks, and to add specific track data such as insert chains, sends, and routing. For example, if a user doesn't know in advance what vocal chain they may want to use, they can begin tracking, and then instantiate a whole set of inserts with stored settings from an existing track preset by clicking on an insert selector and finding that preset.

You are encouraged to create your own track presets and provide them to users in your installers. For example, if you sell plug-in bundles then you may wish to provide users with Track Presets demonstrating useful combinations of multiple plug-ins from the bundle, or if your plug-ins involve some "boilerplate" routing configuration then you can provide a multi-track Track Preset with this routing already established.

Installation Location

Track Presets are stored in the Pro Tools documents folder. Use these locations for default installation

- Mac ~/Documents/Pro Tools/Track Presets
- PC: C:\Users\[username]\Documents\Pro Tools\Track Presets

This location is indexed automatically by Pro Tools.

All of the Track Preset files which you install should be added to a folder with the name of your company. This will ensure that your Track Presets appear as expected in the preset menus in Pro Tools:

- *Pro Tools Documents Folder*

- /Track Presets
 - * · *Name of your company*

Tagging

A default tags dictionary is available from the [My Toolkits and Downloads](#) page at [avid.com](#). These are not the only tags you can use, but any of these that you do use will be increasing the value and usability of the default set included with Avid products. Using this shared dictionary will ensure that your users can quickly find your Track Presets. Workflow Considerations

- Audibility
 - If you want a track to be heard automatically then route that track to the Monitor Path. If a user is using a Monitor Path the track preset will be instantiated and audible immediately.
 - Track Data to Recall
 - In most cases a Track Preset will be created exactly as a user wants to recall it. The available Track Data to Recall from a preset is quite broad though, so you should consider what default import settings make the most sense for each of your presets.
- Here are some ideal default settings for a generic single track plug-in focused preset:
- Plug-in Format Conversion
 - Format conversion for plug-ins is designed to work if formats are enumerated correctly and available. This would take place for instance when recalling inserts from a stereo track preset to a 5.1 track preset
 - most often this should just work if your plug-in is available in all/most formats.
 - Including Avid Stock Plug-ins
 - If you wish to include any stock Avid plug-ins in your presets for any reason, stick to these plug-ins that are automatically installed by Pro Tools to be as sure as possible that your end user will be able to fully recall the preset:
 - * *AutoPan; BF-76; Channel Strip; Click II; Dither; Down Mixer; D-Verb; Dynamics III; Eleven Lite; EQ III; InTune; Invert/Duplicate; LoFi; Master Meter; Maxim; ModDelay III; Normalize-Gain; Pitch Shift Legacy; Pitch II; RectiFi; Reverse/DC Removal; SansAmp PSA-1; SciFi; Signal Generator; Time Shift; Time Adjuster; Trim; VariFi*
- The following Virtual Instruments are installed separately but come for free with paid Pro Tools versions:
- * *Boom; DB-33; Mini Grand; Structure Free; Vacuum; Xpand!2*

12.50.4 Testing your plug-in

The AAX Plug-In Burnthrough Grid document describes a number of test cases and workflows for multiple AAX plug-in hosts. This document is available for download as part of the AAX SDK Toolkit on the [My Toolkits and Downloads](#) page at [avid.com](#).

12.50.5 Selling your plug-in

12.50.5.1 Avid Marketplace

Through the Avid Marketplace, Alliance partners can easily promote and sell their solutions, with all onboarding, licensing and e-commerce managed by a powerful new system. Seller registration takes minutes and the product submission process is simple and backed by responsive support. Products sold on the Marketplace all benefit from Alliance Partner Certification.

Get your AAX Plug-In ready for sale on Avid Marketplace by following these steps:

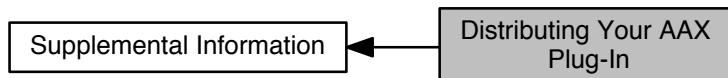
- *Get Certified* - On your "Avid|My Account" page, scroll down to "My Developer Account" and select "My Certifications."
- *Get your Avid Certified Connectivity Developer (ACCD) certification for AAX* - This is a test of your mastery of the AAX Toolkit and the process of working with Avid to deliver your plug-ins to market. Anyone who has reviewed the AAX Toolkit materials in reasonable depth should easily pass. There is a sitting fee for this exam that covers costs of administration, developer support, test services and access to release builds of Pro Tools. A credential is deposited to your account upon passing. Good luck!
- *Explore Avid Marketplace* - Review the [Avid Marketplace description](#) and learn about this valuable and expanding offering. E-mail us at partners@avid.com with your questions.
- *Sign up* - Register as a Seller (sometimes referred to as a "vendor") by following the link from the "My Developer Account" page and selecting "Access the Seller Portal."
- *Prepare your submission* - Gather the plug-in and other information required to onboard as described in the Onboarding FAQ. Your experience will be easier if you collect these items in advance.
- *Send us your Product* - Submit your products and other required information for testing and publication on the Avid Store!

12.50.5.2 In-App Purchase

In-App Purchase provides a direct path to purchase your products from directly within the AAX host application. For example, when a user opens a session which contains unavailable plug-ins, In-App Purchase can be used to prompt the user to purchase the plug-ins immediately.

See [this article](#) for more information about how to add support for In-App Purchase to your on-boarded AAX plug-ins. Additional documentation regarding In-App Purchase is available under the "In-App Purchase Tools" section of the [AAX SDK Toolkit](#) downloads page in your [avid.com](#) account.

Collaboration diagram for Distributing Your AAX Plug-In:



12.51 AAX Interfaces

Full list of AAX interfaces.

12.51.0.1 Interfaces Implemented by the AAX Host

These interfaces are implemented by the AAX Host. References to the host-managed objects are provided to the plug-in through accessor methods, most commonly [IACFUnknown::QueryInterface\(\)](#).

Class [AAX_IAutomationDelegate](#)

Class [AAX_ICollection](#)

Class [AAX_IComponentDescriptor](#)

Class [AAX_IController](#)

Class [AAX_IDma](#)

Class [AAX_IEffectDescriptor](#)

Class [AAX_IHostProcessorDelegate](#)

Class [AAX_IHostServices](#)

Class [AAX_IMIDINode](#)

Class [AAX_IPrivateDataAccess](#)

Class [AAX_IPropertyMap](#)

Class [AAX_ITransport](#)

Class [AAX_IViewContainer](#)

12.51.0.2 Interfaces Implemented by the AAX Plug-In

These interfaces must be implemented by the AAX plug-in. Default implementations are provided in the AAX Library via the `AAX_C` classes. Plug-in classes may inherit from the `AAX_C` classes to override the default behavior.

Class [AAX_IEffectDirectData](#)

Class [AAX_IEffectGUI](#)

Class [AAX_IEffectParameters](#)

Class [AAX_IHostProcessor](#)

12.51.0.3 Interfaces internal to the AAX SDK

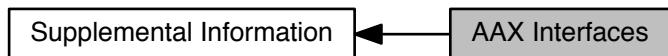
These classes and interfaces are used internally within the AAX Library. References to objects implementing these classes are never passed between the plug-in and the AAX Host, and the AAX Host has no knowledge of these classes.

Class [AAX_IParameter](#)

Class [AAX_IParameterValue](#)

Class [AAX_ITaperDelegateBase](#)

Collaboration diagram for AAX Interfaces:



12.52 Host Support

Supported features in each AAX host.

12.52.1 Host Support

These tables list AAX host support for various AAX interfaces, as well as support for general features. The tables include the version number for the earliest version of each Avid host software which supports the given interface or feature.

The earliest version of each host to support AAX plug-ins is:

- Pro Tools 10.0
- Media Composer 8.1
- VENUE 4.1

For more information about versioning in AAX, including how to check for host support of a particular interface, see [The Avid Component Framework \(ACF\)](#).

12.52.1.1 Platform Support

	Pro Tools	Media Composer	VENUE	
AAX Native	10.0	8.1	<i>none</i>	
AAX DSP	10.0	<i>none</i>	4.1	
AAX Hybrid	11.0*	<i>none</i>	<i>none</i>	
Offline processing (AudioSuite)	10.0	8.1**	<i>none</i>	
ProcessProc / data model co-location	10.0	8.1	<i>none</i>	
Monolithic topology	10.0	8.1	<i>none</i>	
Native processor architecture	10: x86/i386 11+: x86-64	8.1+: x86-64	4.1: x86/i386 4.5+: x86-64	

Note

Pro Tools 11.0 supports AAX Hybrid processing for real-time plug-ins only. Support for AudioSuite processing for AAX Hybrid is supported starting in Pro Tools 11.1.

Media Composer 8.5 and higher support both multichannel and mono AudioSuite processing. Earlier versions of Media Composer support mono only.

12.52.1.2 Describe Interfaces

	Pro Tools	Media Composer	VENUE	
AAX_IACF↔ Collection	10.0	8.1	4.1	
AAX_IACF↔ Component↔ Descriptor	10.0	8.1	4.1	

V2	11.0	8.1	4.5?	
V3	12.8	<i>none</i>	5.6	
AAX_IACFEffectDescriptor	10.0	8.1	4.1	
V2	11.0	8.1	4.5?	
AAX_IACFPropertyMap	10.0	8.1	4.1	
V2	11.0	8.1	4.5?	
V3	12.9	<i>none</i>	5.6	
AAX_IACFDescriptionHost	12.8	<i>none</i>	<i>none</i>	
AAX_IACFFeatureInfo	12.8	<i>none</i>	<i>none</i>	

12.52.1.3 Run-Time Interfaces

	Pro Tools	Media Composer	VENUE	
AAX_IACFAutomationDelegate	10.0	8.1	4.1	
AAX_IACFController	10.0	8.1	4.1	
V2	11.0	8.1	<i>none</i>	
V3	12.4	8.6	<i>none</i>	
AAX_IACFEfectDirectData	10.0	8.1	4.1	
AAX_IACFEfectGUI	10.0	8.1	4.1	
AAX_IACFEfectParameters	10.0	8.1	4.1	
V2	11.0	8.1	4.5?	
V3	11.2	8.1	<i>none</i>	
V4	<i>none</i>	<i>none</i>	5.6	
AAX_IACFHostProcessor	10.0	8.1	<i>none</i>	
V2	12.0	<i>none</i>	<i>none</i>	
AAX_IACFHostProcessorDelegate	10.0	<i>none</i>	<i>none</i>	
V2	11.0	<i>none</i>	<i>none</i>	
V3	12.0	<i>none</i>	<i>none</i>	
AAX_IACFHostServices	10.0	8.1	4.1?	
V2	12.0	8.6	<i>none</i>	
V3	12.8.3	<i>none</i>	<i>none</i>	
AAX_IACFPrivateDataAccess	10.0	8.1	4.1	
AAX_IACFTransport	10.0	8.5 (partial)	<i>none</i>	
V2	10.3.7	8.5 (partial)	<i>none</i>	
AAX_IACFViewContainer	10.0	8.1	4.1	

V2	12.0.1	<i>none</i>	<i>none</i>	
AAX_IACFPage↔ Table	12.8	<i>none</i>	5.7	
V2	12.8.2	<i>none</i>	5.7	
AAX_IACFPage↔ TableController	<i>none</i>	<i>none</i>	5.7	

12.52.1.4 Features

	Pro Tools	Media Composer	VENUE	
Surround Stem Formats (3-8 channels)	10.0	8.1	<i>none</i>	
7.1.2 Stem Format	12.8	<i>none</i>	<i>none</i>	
Ambisonics Stem Formats	12.8.2	<i>none</i>	<i>none</i>	
Plug-in type conversion	10.3.8, 11.0*, 11.1	<i>none</i>	<i>none</i>	
Auxiliary Output Stems	10.0	<i>none</i>	<i>none</i>	
Sidechain Inputs	10.0	8.5	<i>none</i>	
MIDI	10.0	<i>none</i>	<i>none</i>	
Automation recording and playback	10.0	<i>none</i>	<i>none</i>	
Plug-in presets	10.0	8.4	4.1	
External control surfaces	10.0	8.1	<i>none</i>	

12.52.2 Host Compatibility Notes

See also

[Compatibility Notes](#) in the [Pro Tools Guide](#) document

Member [AAX_CMidiPacket::mIsImmediate](#)

This value is not currently set. Use `mTimestamp == 0` to detect immediate packets

Member [AAX_CParameter< T >::AAX_CParameter](#) ([AAX_CParamID identifier](#), [const AAX_IString &name](#), [T defaultValue](#), [const AAX_ITaperDelegate< T > &taperDelegate](#), [const AAX_IDisplayDelegate< T > &displayDelegate](#), [bool automatable=false](#))

As of Pro Tools 10.2, DAE will check for a matching parameter NAME and not an ID when reading in automation data from a session saved with an AAX plug-ins RTAS/TDM counter part.

As of Pro Tools 11.1, AAE will first try to match ID. If that fails, AAE will fall back to matching by Name.

Module [AAX_DigiTrace_Guide](#)

This feature is available in Pro Tools 12.6 and higher

globalScope> Member [AAX_eConstraintLocationMask_FixedLatencyDomain](#)

This constraint is not currently supported by any AAX host

globalScope> Member [AAX_eCurveType_Dynamics](#)

Pro Tools requests this curve type for [Dynamics](#) plug-ins only

globalScope> Member [AAX_eCurveType_EQ](#)

Pro Tools requests this curve type for [EQ](#) plug-ins only

globalScope> Member [AAX_eCurveType_Reduction](#)

Pro Tools requests this curve type for [Dynamics](#) plug-ins only

globalScope> Member AAX_eDataInPortType_Incremental

Supported in Pro Tools 12.5 and higher; when [AAX_eDataInPortType_Incremental](#) is not supported the port will be treated as [AAX_eDataInPortType_Unbuffered](#)

globalScope> Member AAX_EHostModeBits

Supported in Venue 5.6 and higher

globalScope> Member AAX_eNotificationEvent_ASPreviewState

Supported in Pro Tools 11 and higher

Not supported by Media Composer

globalScope> Member AAX_eNotificationEvent_ASProcessingState

Supported in Pro Tools 11 and higher

Not supported by Media Composer

globalScope> Member AAX_eNotificationEvent_DelayCompensationState

Supported in Pro Tools 12.6 and higher

globalScope> Member AAX_eNotificationEvent_EnteringOfflineMode

Supported in Pro Tools 11 and higher

globalScope> Member AAX_eNotificationEvent_ExitingOfflineMode

Supported in Pro Tools 11 and higher

globalScope> Member AAX_eNotificationEvent_HostModeChanged

Supported in Venue 5.6 and higher

globalScope> Member AAX_eNotificationEvent_MaxViewSizeChanged

Supported in Pro Tools 11.1 and higher

globalScope> Member AAX_eNotificationEvent_PresetOpened

Supported in Pro Tools 11 and higher

globalScope> Member AAX_eNotificationEvent_SessionBeingOpened

Supported in Pro Tools 11 and higher

Not supported by Media Composer

globalScope> Member AAX_eNotificationEvent_SessionPathChanged

Supported in Pro Tools 11.1 and higher

globalScope> Member AAX_eNotificationEvent_SideChainBeingConnected

Supported in Pro Tools 11.1 and higher

globalScope> Member AAX_eNotificationEvent_SideChainBeingDisconnected

Supported in Pro Tools 11.1 and higher

globalScope> Member AAX_eNotificationEvent_SignalLatencyChanged

Supported in Pro Tools 11.1 and higher

globalScope> Member AAX_eNotificationEvent_TrackNameChanged

Supported in Pro Tools 11.2 and higher

Not supported by Media Composer

globalScope> Member AAX_ePlugInStrings_Progress

Not currently supported by Pro Tools

globalScope> Member AAX_eProcessingState_BeginPassGroup

AudioSuite pass group notifications are supported starting in Pro Tools 12.0

globalScope> Member AAX_eProcessingState_EndPassGroup

AudioSuite pass group notifications are supported starting in Pro Tools 12.0

globalScope> Member AAX_eProperty_Constraint_NeverUnload

AAX_eProperty_Constraint_NeverUnload is not currently implemented in DAE or AAE

globalScope > Member AAX_eProperty_DestinationTrack

This property is not supported on Media Composer

globalScope > Member AAX_eProperty_LatencyContribution

Maximum delay compensation limits will vary from host to host. If your plug-in exceeds the delay compensation sample limit for a given AAX host then you should note this limitation in your user documentation. Example limits:

- Pro Tools 9 and higher: 16,383 samples at 44.1/48 kHz, 32,767 samples at 88.2/96 kHz, or 65,534 samples at 176.4/192 kHz
- Media Composer 8.1 and higher: 16,383 samples at 44.1/48 kHz, 32,767 samples at 88.2/96 kHz

globalScope > Member AAX_eProperty_OptionalAnalysis

In Media Composer, optional analysis will also be performed automatically before each channel is rendered. See [MCDEV-2904](#)

globalScope > Member AAX_eProperty_SideChainStemFormat

Currently Pro Tools supports only [AAX_eStemFormat_Mono](#) side chain inputs

[AAX_eProperty_SideChainStemFormat](#) is not currently implemented in DAE or AAE

[AAX_eProperty_SideChainStemFormat](#) is not currently implemented in DAE or AAE

globalScope > Member AAX_eProperty_UsesClientGUI

Currently supported by Pro Tools only

Member AAX_IACFEffectorParameters::CompareActiveChunk (const AAX_SPlugInChunk *iChunkP, AAX_IBoolean *olsEqual) const =0

In Pro Tools, this method will only be called if a prior call to [GetNumberOfChanges\(\)](#) has indicated that the plug-in's state has changed. If the plug-in's current settings are different from the settings in [aChunkP](#) then the plug-in's Compare Light will be illuminated in the plug-in header, allowing users to toggle between the plug-in's custom state and its saved state.

Member AAX_IACFEffectorParameters::GetCurveData (AAC_CTypeID iCurveType, const float *iValues, uint32_t iNumValues, float *oValues) const =0

Versions of S6 software which support the [GetCurveDataDisplayRange\(\)](#) method will not display a plug-in's curve data unless both [GetCurveData\(\)](#) and [GetCurveDataDisplayRange\(\)](#) are supported by the plug-in.

Member AAX_IACFEffectorParameters::GetParameterNameOfLength (AAC_CParamID iParameterID, AAC_IString *oName, int32_t iNameLength) const =0

In most cases, the AAX host will call [GetParameterName\(\)](#) or [GetParameterNameOfLength\(\)](#) to retrieve parameter names for display. However, when Pro Tools is retrieving a plug-in name for display on a control surface the XML data stored in the plug-in's page tables will be used in preference to values retrieved from these methods.

Member AAX_IComponentDescriptor::AddAuxOutputStem (AAC_CFieldIndex inFieldIndex, int32_t inStemFormat, const char inNameUTF8[])=0

There is a hard limit to the number of outputs that Pro Tools supports for a single plug-in instance. This limit is currently set at 256 channels, which includes all of the plug-in's output channels in addition to the sum total of all of its aux output stem channels.

Pro Tools supports only mono and stereo auxiliary output stem formats

Member AAX_IComponentDescriptor::AddClock (AAC_CFieldIndex inFieldIndex)=0

As of Pro Tools 11.1, this field may be used in both Native and DSP plug-ins. The DSP clock data is a 16-bit cycling counter. This field was only available for Native plug-ins in previous Pro Tools versions.

Member AAX_IComponentDescriptor::AddMIDIINode (AAC_CFieldIndex inFieldIndex, AAC_EMIDINodeType inNodeType, const char inNodeName[], uint32_t channelMask)=0

Due to current restrictions MIDI data won't be delivered to DSP algorithms, only to AAX Native.

Member AAX_IController::GetHostName (AAC_IString *outHostNameString) const =0

Pro Tools versions from Pro Tools 11.0 to Pro Tools 12.3.1 will return a generic version string to this call. This issue is resolved beginning in Pro Tools 12.4.

Member AAX_IMIDINode::PostMIDIpacket (AAX_CMidiPacket *packet)=0

Pro Tools supports the following MIDI events from plug-ins:

- NoteOn
- NoteOff
- Pitch bend
- Polyphonic key pressure
- Bank select (controller #0)
- Program change (no bank)
- Channel pressure

Member AAX_ITransport::GetBarBeatPosition (int32_t *Bars, int32_t *Beats, int64_t *DisplayTicks, int64_t SampleLocation) const =0

There is a minor performance cost associated with using this API in Pro Tools. It should not be used excessively without need.

Member AAX_ITransport::GetCurrentLoopPosition (bool *bLooping, int64_t *LoopStartTick, int64_t *LoopEndTick) const =0

This does not indicate anything about the status of the "Loop Record" option. Even when the host is configured to loop playback, looping may not occur if certain conditions are not met (i.e. the length of the selection is too short)

Member AAX_ITransport::GetCurrentTickPosition (int64_t *TickPosition) const =0

The tick resolution here is different than that of the tick displayed in Pro Tools. "Display ticks" (as they are called) are 1/960 of a quarter note.

Member AAX_ITransport::GetCustomTickPosition (int64_t *oTickPosition, int64_t iSampleLocation) const =0

There is a minor performance cost associated with using this API in Pro Tools. It should not be used excessively without need.

Member AAX_IViewContainer::GetModifiers (uint32_t *outModifiers)=0

Although this method allows plug-ins to acquire the current state of the Windows key (normally blocked by Pro Tools), plug-ins should not use key combinations that require this key.

Module AAX_Media_Composer_Guide

Some early versions of Media Composer 8 do not search the system plug-ins directory recursively. If your plug-ins are installed into a sub-directory beneath this main directory then they will not be loaded by the affected versions of Media Composer.

Module AAX_Page_Table_Guide

Pro Tools versions prior to Pro Tools 11.1 use plug-ins' ProControl and ICON page tables (Dynamics, EQ, Channel Strip, Custom Fader, etc.) to map plug-in parameters to EUCON-enabled surfaces, so be sure that your plug-ins also implement these page tables correctly so that users with earlier versions of Pro Tools can have the best possible experience when using your plug-ins.

Module AAX_Pro_Tools_Guide

Pro Tools 11 requires PACE Eden digital signatures for AAX plug-ins.

Pro Tools 10 requires either PACE DSig or Eden digital signatures for AAX plug-ins.

As of Pro Tools 10.2, support has been added to allow binary-level encryption of AAX DSP algorithms. Please note that this is *NOT* a requirement of AAX DSP plug-ins, and serves only as an additional security measure to protect an algorithm's DLL.

In Pro Tools 11 and above, the Avid Audio Engine (AAE) no longer automatically generates clipping indication for plug-ins. This is because the new engine can operate properly well beyond a unity sample value of 1.0. Thus, it is up to the plug-in itself to set and clear its clip indicators as needed.

Supported in Pro Tools 12 and higher.

Supported in Pro Tools 12 and higher.

Supported in Pro Tools 12 and higher.

Supported in Pro Tools 12.6 and higher.

Supported in Pro Tools 12.8.2 and higher.

Module AAX_TI_Guide

32 and 64-sample quantum is available in Pro Tools 10.2 and higher

Beginning in Pro Tools 11, AAX DSP algorithms also support optional temporary data spaces that can be described in the Describe module and are shared among all instances on a DSP. This is an alternative to declaring large data blocks on the stack for better memory management and to prevent stack overflows. Please refer to [AAX_IComponentDescriptor::AddTemporaryData\(\)](#) for usage instructions.

Beginning with Pro Tools 10.2, the TI shell supports a "processor affinity" property, which indicates that a DSP ProcessProc should be preferentially loaded onto the same DSP as other instances from the same DLL binary. This is a requirement for some designs that must share global data between different processing configurations.

Note that this property should only be used when absolutely required, as it will constrain the DSP manager and reduce overall DSP plug-in instance counts on the system.

Module AdditionalFeatures_CurveDisplays

For S6 control surface displays, see [PT-226228](#) and [PT-226227](#) in the Known Issues page for more information about the requirements listed in this section.

Module advancedTopics_relatedTypes

Pro Tools versions prior to Pro Tools 12.3 do not allow explicit type conversion between types with different product ID values. Beginning in Pro Tools 12.3 both the product ID and the plug-in ID may differ between explicitly related types.

- Pro Tools versions before Pro Tools 12.3 treat deprecated and related type associations identically and do not support type deprecation features
- Media Composer does not support type deprecation features
- VENUE does not support type deprecation features

Module CommonInterface_Algorithm

As of Pro Tools 10.2.1 an algorithm's initialization callback routine will have up to 5 seconds to execute.

Module CommonInterface_FormatSpecification

The plug-in's binary filename must be the same as the outer .aaxplugin bundle name

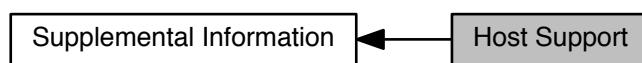
- On Windows, the plug-in binary (DLL) must use the ".aaxplugin" suffix; i.e. the DLL must use exactly the same name as the outer .aaxplugin folder. On OS X, the plug-in binary does not require a specific suffix.
- On Windows, the plug-in's binary filename (and therefore also the outer .aaxplugin file name) must not contain any spaces. There is a bug in AAE that will prevent binaries with spaces from being loaded properly. This is logged as PTSW-189928.

* _ACFGetSDKVersion is required for 64-bit AAX plug-ins only

Module ExamplePlugins

The DemoDelay_DynamicLatencyComp example is compatible with Pro Tools 11.1 and higher.

Collaboration diagram for Host Support:



12.53 Known Issues

A list of known bugs affecting AAX plug-ins.

12.53.1 Contents

- [Known Issues in the AAX SDK](#)
- [Known Issues in Pro Tools](#)
- [Known Issues in Media Composer](#)
- [Known Issues in Control Surfaces](#)
- [Known Issues in AAX Tools](#)

12.53.2 Known Issues in the AAX SDK

12.53.2.1 AAXSDK-561

An AAX Hybrid DSP plug-in with 7.1.2 input and output stem formats and a 16-sample processing buffer size will throw an AAE -14382 error upon instantiation at 192kHz

Resolution: This bug is unresolved

12.53.2.2 AAXSDK-599

In the Win32 GUI example plug-in, the mouse cursor disappears when text is entered in text box and only re-appears when the mouse is moved out of the plug-in window bounds

Resolution: This bug is unresolved

12.53.2.3 AAXSDK-533

AAXLibrary compiles with warnings in Visual Studio 2015

Resolution: This bug is fixed as of AAX SDK 2.3.0

12.53.2.4 AAXSDK-514

Using collection-level properties leads to a leaked ACF object

Resolution: This bug is fixed as of AAX SDK 2.3.0

12.53.2.5 AAXSDK-321

Demo Delay (mono) DSP / Demo Gain (Cocoa UI) (mono) DSP can't be instantiated

Resolution: This bug is fixed as of AAX SDK 2.2.0

12.53.2.6 AAXSDK-271

DemoMIDI_Sampler: No audio on right channel (multi-mono)

Resolution: This bug is fixed as of AAX SDK 2.2.0

Discussion: DemoMIDI_Sampler and DemoMIDI_Synth are now restricted to not use multi-mono, via the [AAX_e↔Property_Constraint_MultiMonoSupport](#) property.

In Pro Tools, when a track's MIDI destination is set to "none" and a new plug-in that includes a MIDI input node (e.g. any Instrument plug-in) is instantiated, the track's MIDI destination is set to the first newly created MIDI input node.

When the track in question is greater than mono, and the Instrument plug-in is multi-mono, the default behavior is for the track's MIDI destination to be set to the first of the newly created input nodes, not to all of them simultaneously. As a result, the track's MIDI destination is set to the MIDI input node of the first (left) multi-mono instance of the plug-in.

To route MIDI to all channels of a multi-mono plug-in in Pro Tools, ctrl-select the MIDI destination and choose the additional MIDI nodes.

12.53.2.7 AAXSDK-186

C99Compatibility constructs are incorrect when used with VS2012

Resolution: This bug is fixed as of AAX SDK 2.1.1

12.53.2.8 AAXSDK-162

Misleading warning message when attempting hardware debugging using a non-local TIShell.out

The "Load ProTools Plug-in Symbols" script gives the following warning: D:/Code_7/dev.ws↔backup-win7-concert/AAX/Internal/SystemSoftware/TIShell/CCS_Project/TI↔Shell/../../../../../../../../WinBag/x64/Release/bin/TIShell.out does not exist! Please question everything.

This error message includes a hard-coded path that is not relevant to the running system. This error is benign but it can be confusing.

Resolution: This bug is unresolved

12.53.2.9 AAXSDK-16

AAX SDK: Win32 example plug-in GUI does not appear in Windows 8

Resolution: This bug is fixed as of AAX SDK 2.2.0

12.53.2.10 AAXSDK-14

AAX DemoGain_VST: Text box entry is not acknowledged upon click outside of window

Resolution: This bug is unresolved

12.53.2.11 AAXSDK-13 / AAX-579 / PTSW-158381

AAX SDK Win32 example plug-in does not "snap to default" on option-click

Resolution: This bug is fixed as of AAX SDK 2.2.0

12.53.2.12 AAXSDK-11 / AAX-581 / PTSW-158348

AAX SDK VSTGUI example plug-in does not respond to 'alt' or 'win' modifier keys (Windows)

Resolution: This bug is fixed as of AAX SDK 2.2.0

12.53.2.13 AAXSDK-10 / AAX-580 / PTSW-154083

AAX DemoGain_VST and DemoGain_Cocoa require initial click on GUI to take focus before editing (OS X)

Resolution: This bug is not yet resolved. For OS X, one workaround is to modify VSTGUI's cocoasupport.mm file in order to add a handler for the acceptsFirstMouse selector:

```
static BOOL VSTGUI_NSView_acceptsFirstMouse(
    id self,
    SEL _cmd)
{
    return YES;
}

// In VSTGUI_NSView_isOpaque()
res = class_addMethod(
    viewClass,
    @selector(acceptsFirstMouse:),
    IMP (VSTGUI_NSView_acceptsFirstMouse),
    "B@:@:");

```

Special thanks to Nick Protokowicz for suggesting this workaround.

12.53.2.14 AAXSDK-6 / AAX-646

AAX SDK: Incorrect output from scatter/gather DMA example plug-in when increasing playback buffer size while audio is present (Native decks)

Resolution: This bug is unresolved

Workaround: The workaround for this issue is to not run audio through this plug-in while increasing the playback buffer size.

12.53.2.15 AAXSDK-5

[AAX_CChunkDataParser::LoadChunk](#) doesn't handle unknown chunk items well

Resolution: This bug will not be fixed

Discussion: [AAX_CChunkDataParser](#) does not store the size of each chunk item in the data stream. Therefore there is no way to determine the correct size for each data element when reading a chunk that was generated by this parser.

Workaround: If you know the correct size of each data element in a chunk when it is read by the plug-in, you can override the [AAX_CChunkDataParser](#) methods to ensure that each data element is correctly sized.

12.53.2.16 AAXSDK-2 / AAX-648

AAX SDK: Output from DMA example plug-in is one buffer early

Resolution: This bug is unresolved

12.53.2.17 AAX-582 / PTSW-157726

AAX SDK example plug-ins' controls do not write automation properly when in 'touch' mode (frequently revert to default value while writing)

Resolution: This bug is fixed as of the 1.0.4 SDK

12.53.2.18 AAX-585 / PTSW-157451

AAX DemoGain GUI example plug-ins do not correctly handle alt/opt-click for resetting controls to their default state

Resolution: This bug is partially resolved as of AAX SDK 1.0.4. See also PTSW-158348 and PTSW-158381.

12.53.2.19 AAX-578 / PTSW-158310

AAX SDK JUCE example plug-in does not "snap to default" on option-click

Resolution: This bug is fixed as of AAX SDK 1.0.4

12.53.3 Known Issues in Pro Tools**12.53.3.1 PT-235831**

The plug-in frame overlaps the plug-in window header if the plug-in GUI is taller than the screen height less the plug-in window header height.

Resolution: This bug is unresolved

Workaround: Avoid resizing the plug-in GUI to a height greater than the display height.

12.53.3.2 PT-235333

Dynamic plug-in processing incorrectly shuts off processing in mixed multi-mono/multichannel chains that should force processing

This bug can occur if a multi-mono plug-in precedes a multichannel plug-in which sets [AAX_eProperty_ConstraintAlwaysProcess](#)

Resolution: This bug is unresolved

12.53.3.3 PT-234681

AAX plug-in parameter handling may cause audio glitches on Windows for plug-ins with very long [GenerateCoefficients\(\)](#) execution time

Resolution: This bug is unresolved

12.53.3.4 PT-233726

Unprintable characters in four-char parameter IDs may result in -9105 errors

Resolution: This bug is fixed as of Pro Tools 2018.1

12.53.3.5 PT-233176

AAX digital signature check fails on pre-Sierra systems for plug-ins signed on Sierra

Resolution: This bug has been reported to Avid but is not yet confirmed. Contact PACE Anti-Piracy, Inc. if you encounter this behavior.

12.53.3.6 PT-232678 / PT-236755

Plug-in aux outputs are silent for upmix plug-ins when using AAX Native plug-ins with the HDX playback engine

Resolution: This bug is fixed as of Pro Tools 2018.1

12.53.3.7 PT-232403

ProTools shows error if the plug-in's multi-chunk preset file contain incorrect chunk listed in the end

For example, if a new chunk type has been added to a later version of a plug-in and a preset from that version is opened in an earlier version of the plug-in.

Resolution: This bug is fixed as of Pro Tools 12.8.2

12.53.3.8 PT-232159

AAX Hybrid plug-ins with more than 16 total input channels (direct input and hybrid input) raise AAE -14382 error upon instantiation at 192 kHz sample rate

Resolution: This bug is unresolved

12.53.3.9 PT-230327

The AAX related types feature is broken ([AAX_IACFPropertyMap_V3](#) inheritance is incorrect)

Resolution: This bug was introduced in Pro Tools 12.8 and is fixed as of Pro Tools 12.8.1

12.53.3.10 PT-230290

The plug-in preset menu takes a long time to build for plug-ins with a very large number of preset .tfx files

Resolution: This bug is unresolved

12.53.3.11 PT-230288

The plug-in preset menu contains empty folders for other Effects in the same .aaxplugin bundle

Resolution: This bug is fixed as of Pro Tools 12.8.2

12.53.3.12 PT-229026

Pro Tools may crash after plug-in parameter tweaks on Windows

Resolution: This rare crash may be fixed in Pro Tools 2018.1

12.53.3.13 PT-227655

[AAX_ITransport::GetTimelineSelectionStartPosition\(\)](#) provides incorrect values for real-time plug-ins - value depends on transport time display selection in Pro Tools

Resolution: This bug is fixed as of Pro Tools 12.8

12.53.3.14 PT-227173

Incorrect timecode is sent to plug-ins when delay is present before the plug-in

Resolution: This bug is unresolved

12.53.3.15 PT-227069

Pro Tools 12.7 and later developer builds crash if a debugger is attached during the first session open (Mac only)

Resolution: This bug is unresolved

Workaround: Attach a debugger after opening a session in Pro Tools. After the first session open, subsequent session open actions will not result in a crash.

12.53.3.16 PT-226559

Transport location provided to plug-ins is incorrect during half-speed playback

Resolution: This bug is unresolved

12.53.3.17 PT-225763

Incorrect AudioSuite processing modes are available for multichannel random access plug-ins

Resolution: This bug is unresolved

12.53.3.18 PT-223581

Pro Tools removes plug-ins from the insert menu if unsupported stem formats are detected

Resolution: This bug is fixed as of Pro Tools 12.8

This bug is also now fixed in earlier versions of Pro Tools via a workaround which is now built into the AAX SDK during `Describe`. See [AAX_VPropertyMap::AddProperty\(\)](#)

12.53.3.19 PT-218545

There is no way for AAX plug-ins to opt out of the default settings chunk sequence during plug-in instantiation

Resolution: This requested enhancement is not yet implemented

12.53.3.20 PT-210904 / VSW-14216

HDX errors can occur due to over-allocation of certain plug-ins

This bug applies specifically to AAX DSP plug-ins which register the same DLL and algorithm entry point name for multiple modules with different [AAX_eProperty_TI_MaxInstancesPerChip](#) requirements.

In VENUE, this bug causes a 'No Information Available' error dialog on a single DSP chip

Resolution: This bug is fixed as of Pro Tools 12.5 and VENUE 5.1

12.53.3.21 PT-206995

AOS is not cleaned up in [AAX_IComponentDescriptor::Clear](#)

Resolution: This bug will not be fixed

12.53.3.22 PT-206541

AAX automation playback is late and non-deterministic

Resolution: This bug will not be fixed

12.53.3.23 PT-206449

Pro Tools developer builds: Asserts result in -300xx errors rather than an explanatory dialog

Resolution: This bug is fixed as of Pro Tools 12.3

12.53.3.24 PT-206161

HDX: AAX packets are not delivered to ports 16 or 24 (zero-indexed) when [PostPacket\(\)](#) is called outside of [GenerateCoefficients\(\)](#)

Workaround: Make all calls to [AAX_IController::PostPacket\(\)](#) within the scope of [AAX_IEffectParameters::GenerateCoefficients\(\)](#)

Resolution: This bug will not be fixed

12.53.3.25 PT-205610

AAX Hybrid: transport location and clock methods do not provide correct values when called from the Hybrid render callback

Resolution: This bug is fixed as of Pro Tools 12.4

12.53.3.26 PT-203420

`TestGetCurveData` DigiOption results in incorrect plug-in view offset for plug-ins with MIDI

Resolution: This bug will not be fixed

Workaround: Temporarily disable MIDI in your plug-in while developing or debugging the plug-in's curve data, then re-enable MIDI once you have finished using the `TestGetCurveData` DigiOption.

12.53.3.27 PT-202345

Pro Tools may incorrectly identify plug-ins when a single plug-in uses identical plug-in IDs across different Effects

Resolution: This bug is fixed as of Pro Tools 12.2

12.53.3.28 PTSW-200437 / PTSW-197598

Plug-Ins that use the "Related Types" feature cannot relate to plug-ins with a different ProductID

Resolution: This bug is fixed as of Pro Tools 11.3.2 and Pro Tools 10.3.11

12.53.3.29 PTSW-197651 / PT-218405

In some cases AAX plug-ins do not show the correct Control Name Variation and just read out the automation name

Resolution: This bug will not be fixed

12.53.3.30 PTSW-197601 / PT-218459

Control surfaces can send illegal parameter values to plug-ins

Resolution: This bug will not be fixed

12.53.3.31 PTSW-197593 / PT-218480

HDX: A chip may be full with just a small percent of the System Usage meter filled

Resolution: This bug will not be fixed

12.53.3.32 PTSW-197540

AudioSuite: Analyze mode is not working properly when processing method is set to "clip-by-clip"

Resolution: This bug is fixed as of Pro Tools 11.3.1

12.53.3.33 PTSW-197472

Dynamic Plug-In Processing is unnecessarily disabled for plug-ins in the "Other" category

Resolution: This bug is fixed as of Pro Tools 12

12.53.3.34 PTSW-197471

AudioSuite only analyzes the first clip in "clip list" mode

Resolution: This bug is fixed as of Pro Tools 12

12.53.3.35 PTSW-197468 / PT-218460

Pro Tools may incorrectly change a plug-in instance when another instance is edited

Resolution: This bug is fixed in Pro Tools 2018.1

Discussion: This issue only happens when the two plug-in types use the same Manufacturer and Plug-In IDs and use Product IDs with unprintable chars when interpreted as four-char values.

We strongly recommend that you select Product IDs in the printable ASCII four-char range.

12.53.3.36 PTSW-197431 / PT-218414

Pro Tools 10: Plug-in Side-chain input is silent when the plug-in supports Aux Outputs

Resolution: This bug will not be fixed

12.53.3.37 PTSW-197075

Plug-in preset files for some plug-ins are not cross-compatible between Mac and Windows

Resolution: This bug is fixed as of Pro Tools 11.2 and Pro Tools 10.3.10

12.53.3.38 PTSW-196772 / PT-218423

A [View Size Changed](#) notification is not sent when connecting/disconnecting a display

Resolution: This bug will not be fixed

12.53.3.39 PTSW-196604

Cannot adjust plug-in parameters with large numbers of steps using control surface (Artist Series)

Resolution: This bug is fixed as of Pro Tools 12.4

12.53.3.40 PTSW-196428 / PT-218488

AudioSuite preview allows processing with incorrect number of channels

Resolution: This bug will not be fixed

12.53.3.41 PTSW-195316 / PT-218485

EQ/Dyn graphs on EUCON surfaces are not always updated for plug-ins with many EQ/Dyn parameters

Resolution: This bug will not be fixed

Discussion: Currently, EUCON surfaces only update a plug-in's EQ/Dyn curve plots when an update occurs to one of the parameters which is mapped to the plug-in's "center section" EQ/Dyn page tables. Other parameters will not trigger an update to the plug-in's EQ/Dyn curve plot.

12.53.3.42 PTSW-195257

Calling [AAX_ICollection::AddEffect\(\)](#) multiple times using the same `iEffectID` only returns an error if called with the same [effect descriptor](#), but this is always illegal

Resolution: This bug is fixed as of Pro Tools 12

Discussion: Calling [AAX_ICollection::AddEffect\(\)](#) using the same ID will now return an error in all cases

12.53.3.43 PTSW-195256 / PT-218429

Plug-in is not notified of preset load when loading factory default presets

Resolution: This bug will not be fixed

12.53.3.44 PTSW-195209 / PT-218474

[AAX_IViewContainer::HandleParameterMouseUp\(\)](#) returns [AAX_ERROR_UNIMPLEMENTED](#) when using control-command-option-click on a plug-in GUI control

Resolution: This bug will not be fixed

Discussion: See the discussion of this bug in the [AAX_IViewContainer](#) documentation.

12.53.3.45 PTSW-195113

Automation problems with plug-in parameter names > 31 characters

Resolution: This bug is fixed as of Pro Tools 11.2.1

Discussion: This bug was introduced in Pro Tools 11.1

12.53.3.46 PTSW-194698 / PT-218478

Very hard to edit plug-in parameters with many steps using a control surface rotary encoder

Resolution: This bug will not be fixed

12.53.3.47 PTSW-194231 / PT-218434

When the output on a track is set to "no output" then no audio is sent to Auxiliary Outputs of the plug-ins on the track

Resolution: This bug is unresolved

Workaround: Users can work around this issue by ensuring that a track output is always assigned for tracks with plug-ins that generate Auxiliary Output channels. For example, the user may pull down the track's output gain fader, enable MUTE, or select an unused output channel or bus.

12.53.3.48 PTSW-193646

AudioSuite plug-ins are not able to partially re-name clips

Resolution: This bug is fixed as of Pro Tools 12

12.53.3.49 PTSW-193400

AOS plug-ins become active when moving an inactive plug-in to another insert

Resolution: This bug is fixed as of Pro Tools 11.2

12.53.3.50 PTSW-193345

AudioSuite processing notifications are not sent at the start and end of a processing event

Resolution: This bug is fixed as of Pro Tools 12

Discussion: Two new notifications were added to provide this behavior. The existing AudioSuite notifications retain their behavior: they are sent before and after each processing pass, i.e. at the beginning and end of each audio channel that is processed, even if the current selection includes multiple channels. For more information, see [AAX_EProcessingState](#)

12.53.3.51 PTSW-193339

Pro Tools does not update plug-in settings when a new setting's name matches an old setting and the modification date is later

Resolution: This bug will not be fixed

Workaround: To update the settings that are bundled with a plug-in, the plug-in's installer should search for and remove any deprecated settings files on the system.

12.53.3.52 PTSW-193051

Using Aux Output Stems on DSP plug-ins causes them to crash

Resolution: This bug is fixed as of Pro Tools 11.2

Discussion: This bug was introduced in Pro Tools 11.1

12.53.3.53 PTSW-192863 / PT-218498

Plug-in side chain input is not properly delay compensated: aligned with output instead of input, no individual tap per insert

Resolution: This bug is unresolved

Discussion: Side chain delay compensation on native playback engines is not currently supported

12.53.3.54 PTSW-192755

Key focus is not returned to a plug-in after it launches a dialog

Resolution: This bug will not be fixed (design limitation)

12.53.3.55 PTSW-192720 / PT-218467

External source (SideChain) key input is not reported to DSP Dynamics plug-ins after HDX re-shuffle, hence no Gain Reduction occurs.

Resolution: This bug is unresolved

12.53.3.56 PTSW-192635

Implement Manufacturer ID byteswap

Resolution: This bug is fixed as of Pro Tools 11.2 and Pro Tools 10.3.10

12.53.3.57 / PT-218490

Plug-in settings chunks with incorrect fSize result in junk data

Resolution: This bug will not be fixed

12.53.3.58 PTSW-192251 / PT-218394

EUCON surface cells sometimes behave inconsistently when discrete plug-in parameters are mapped to rotary encoders

Resolution: This bug will not be fixed

12.53.3.59 PTSW-192086 / PT-218465

AudioSuite [AAX](#): Pro Tools performs multiple unnecessary render passes when rendering in multi-input mode

Resolution: This bug will not be fixed

12.53.3.60 PTSW-191875

Pro Tools uses a hard-coded version string when publishing its version to AAX plug-ins ([AAX_IController::Get←HostName](#))

Resolution: This bug is fixed as of Pro Tools 12.4

12.53.3.61 PTSW-191446 / PT-218600

Global symbols due to statically linked boost libs in Pro Tools components may conflict with plug-in components that use boost

Resolution: This bug is unresolved

12.53.3.62 PTSW-191317 / PT-218425

The Pro Tools meter decay setting is not applied to plug-in meters

Resolution: This bug will not be fixed

12.53.3.63 PTSW-191139

Plug-ins do not receive parameter touch state when automation-enabled in Pro Tools 11.1

Resolution: This bug is fixed as of Pro Tools 11.1.2

12.53.3.64 PTSW-190722

Some plug-in state changes do not trigger the Pro Tools session "dirty" flag

Resolution: This bug is fixed as of Pro Tool 11.1.2 and and Pro Tools 10.3.9

12.53.3.65 PTSW-190719

Unexpected behavior for plug-in auxiliary output channels > 128

Resolution: This bug will be fixed as of Pro Tools 11.1.3

12.53.3.66 PTSW-190340

In some cases AAX plug-ins do not show the correct Control Name Variation on control surfaces

This bug can occur when a page table references parameters by ID and some parameters' ID strings are exactly as long as the control surface's display. In this scenario, the control surface will display the parameter's ID string rather than a Control Name Variations (abbreviation) string of equivalent length.

Resolution: This bug is fixed as of Pro Tools 12

12.53.3.67 PTSW-189928 / PT-218456

Failure to load AAX plug-ins with spaces in DLL filename

Resolution: This bug will not be fixed

12.53.3.68 PTSW-189738 / PT-218494

AAX: [AAX_IController::PostPacket\(\)](#) doesn't return any error if you attempt to post to a private data field

Resolution: This bug will not be fixed

12.53.3.69 PTSW-189725 / PT-218397

Auto-generated AudioSuite plug-in GUIs are non-functional for the first plug-in loaded into the window

This bug applies to AudioSuite plug-ins which use the [AAX_eProperty_UsesClientGUI](#) property. This bug is present in all Pro Tools versions which support AAX.

Workaround: This bug only applies when an AudioSuite window is first created. To resolve the issue, toggle an open AudioSuite window between different plug-ins. After toggling to another plug-in and back to the original plug-in, the auto-generated GUI will again be functional.

Resolution: This bug will not be fixed

12.53.3.70 PTSW-189439 / PT-218427

Attempts to set signal latency by non-linear AudioSuite plug-ins should fail, but do not

Resolution: This bug will not be fixed

12.53.3.71 PTSW-189279

An AudioSuite plug-in ID may be incorrectly used as a related type, preventing type-swapping

Resolution: This bug is fixed as of Pro Tools 11.2 and Pro Tools 10.3.10.

12.53.3.72 PTSW-188836 / PT-218428

[AAX_CMidiPacket::mIsImmediate](#) field is not getting set for real-time MIDI messages

Resolution: This bug will not be fixed

12.53.3.73 PTSW-188830

AudioSuite: [PreRender\(\)](#) is not called before each preview pass

Resolution: This bug is fixed as of Pro Tools 11.1

12.53.3.74 PTSW-188653 / PT-218451

Plug-ins are not unloaded and cannot be swapped when [EnablePlugInHotSwap](#) option is enabled

Resolution: This bug will not be fixed

The workaround for this issue is to re-launch Pro Tools after installing a new build of the plug-in

12.53.3.75 PTSW-188310 / PT-218420

Pro Tools 11 developer builds hang when LLDB is detached

Resolution: This bug will not be fixed

12.53.3.76 PTSW-188309

Pro Tools 11 developer builds hang when LLDB is attached at the [PauseDuringLaunchToAttachDebugger](#) dialog

Resolution: This bug is fixed as of Pro Tools 11.2

12.53.3.77 PTSW-188161

It is not possible to launch some Pro Tools 11 and later development builds from a debugger

Resolution: This bug is unresolved. It applies to all Pro Tools 11 and higher development builds on Windows and to some development builds on Mac.

12.53.3.78 Workaround

Use the [PauseDuringLaunchToAttachDebugger](#) [DigiOption](#) to attach the debugger at a safe point in the Pro Tools launch process

12.53.3.79 PTSW-187670

Plug-in preset menu takes a long time to load with many presets

Resolution: This bug is fixed as of Pro Tools 11.1

12.53.3.80 PTSW-187220 / PT-218584

[AAX_ePrivateDataOptions_KeepOnReset](#) is not implemented

Resolution: This bug is unresolved

12.53.3.81 PTSW-187216 / PT-218491

Pro Tools has a problem with [AAX_IController::PostPacket\(\)](#) being called during [AAX_IEffectParameters::Timer←Wakeup\(\)](#)

Resolution: This bug will not be fixed

Workaround: This bug occurs only with unbuffered plug-ins' ports for coefficients, so the workaround for this issue is to use buffered ports instead.

12.53.3.82 PTSW-187159

Plug-in parameters can get stuck in touched state; touch/release tokens do not always match

One race condition that could result in this bug behavior has been addressed in Pro Tools 11.1.0. However, the bug can still occur when using EUCON control surfaces due to a mismatch in touch/release tokens sent from those surfaces.

Resolution: This bug is resolved as of Pro Tools 11.1.3. It is unresolved in Pro Tools 10

12.53.3.83 PTSW-187066 / PT-218391

[AAX_ITransport::GetCurrentNativeSampleLocation\(\)](#) returns invalid value on the start of playback

Resolution: This bug will not be fixed

12.53.3.84 PTSW-186864

Automatable parameter values may change between [Set](#) and [Update](#)

Resolution: This bug is unresolved

12.53.3.85 PTSW-186725

Related types do not work when used with [AAX_eProperty_SampleRate](#)

Resolution: This bug is fixed as of Pro Tools 11.1

12.53.3.86 PTSW-186627

AAX plug-ins whose context field IDs are not defined in Describe cause a crash in Pro Tools

Resolution: This bug is closed (unable to reproduce)

12.53.3.87 PTSW-186253

AudioSuite GUI work causes audio playback glitches and stutters

Resolution: This bug is fixed as of Pro Tools 11.2.

12.53.3.88 PTSW-186189

If an AAX plug-in does not declare all the fields in its context block, undefined behavior may occur (possibly a crash)

Resolution: This bug is fixed as of Pro Tools 10.3.8 and Pro Tools 11.0.2

12.53.3.89 PTSW-186182

On Windows, VSTGUIv4 plug-in GUIs do not receive key events (PT11 only)

Resolution: This bug is addressed with a patch to the AAX SDK's VSTGUI extension implementation as of AAX SDK 2.1.0.

12.53.3.90 PTSW-185868 / PT-218439

AAX: Calls to [SetValue\(\)](#) early in plug-in life may not propagate to [UpdateParameterNormalizedValue\(\)](#)

Resolution: This bug will not be fixed

The workaround for this issue is to call [SetValue\(\)](#) redundantly until the desired value is updated.

12.53.3.91 PTSW-185867 / PT-218470

Session tempo should be available during [EffectInit\(\)](#)

Resolution: This bug will not be fixed

Workaround: The workaround for this issue is to poll the transport interface in [TimerWakeUp\(\)](#) or otherwise call it after [EffectInit\(\)](#) completes.

12.53.3.92 PTSW-185866

Pro Tools does not respond to [SetParameterNormalizedValue\(\)](#) while offline bouncing

Resolution: This bug is fixed as of Pro Tools 11.1. However, note that we do not recommend implementing linked parameters using direct calls to [SetParameterNormalizedValue\(\)](#). For an explanation of the correct approach to parameter linking, see [Linked parameters](#), with examples provided in the SDK example plug-ins.

12.53.3.93 PTSW-185825 / PT-218464

Undo key events do not reach plug-ins (Windows)

Resolution: This bug will not be fixed

12.53.3.94 PTSW-185537

Use of DigiTrace results in [eTISysSwapScriptTimeout](#)

Resolution: This bug fixed as of Pro Tools 11.1

12.53.3.95 PTSW-185484

[AAX_TRACE_RELEASE](#) crashes at highest optimization setting in AAX DSP plug-ins

Resolution: This bug is unresolved

12.53.3.96 PTSW-185483

DigiTrace: Only one parameter can be sent per trace on HDX

Resolution: This bug is fixed as of Pro Tools 11.1

12.53.3.97 PTSW-185462

AudioSuite: Error 1224 on AudioSuite render when significantly changing the length of a clip (Windows 8)

Resolution: This bug is fixed as of Pro Tools 11.1

12.53.3.98 PTSW-185343

[AAX_ITransport::GetTimeCodeInfo](#) returns invalid values for AAX Instruments

Resolution: This bug is fixed as of Pro Tools 10.3.7 and Pro Tools 11.0.2

12.53.3.99 PTSW-185341

Related types come up as inactive when going from HDX > Native

Resolution: This bug is fixed as of Pro Tools 11.1

12.53.3.100 PTSW-184777 / PT-218483

AAX plug-in meters are not cleared during silence

This bug is new to Pro Tools 11. It does not occur in Pro Tools 10.

Resolution: This bug will not be fixed

12.53.3.101 PTSW-184770

AAX Hybrid plug-ins cannot be opened as AudioSuite (AAE -7103 error)

Resolution: This bug is fixed as of Pro Tools 11.1

12.53.3.102 PTSW-184682

Incorrect audio buffer length provided when a native plug-in (erroneously) registers [AAX_eProperty_AudioBufferLength](#)

Resolution: Since this is an unsupported plug-in configuration this bug will not be fixed

12.53.3.103 / PT-218627

Re-add support for AudioSuite "progress" dialog re-naming (was supported in Pro Tools 9 and earlier)

Resolution: This bug is unresolved

12.53.3.104 PTSW-184619 / PT-218473 / AAX-600

AAX MIDI plug-ins' MIDI channels are not uniquely labeled

Resolution: This bug will not be fixed

12.53.3.105 PTSW-184541

Native engine strides by 2048 samples at 96kHz (expect <= 1024)

Resolution: This bug fixed as of Pro Tools 11.0.1

12.53.3.106 PTSW-183902 / PT-218479

[AAX_IHostProcessorDelegate::GetAudio\(\)](#) responds to invalid iLocation as if everything succeeded

Resolution: This bug will not be fixed

12.53.3.107 PTSW-183848 / PT-218390

[AAX_IHostProcessorDelegate::GetAudio\(\)](#) ignores input audio buffer parameter

Resolution: This bug will not be fixed

The workaround for this issue is to make sure that HostProcessor plug-ins only request valid audio - do the boundary-condition checking inside the plug-in.

12.53.3.108 PTSW-183841

Plug-ins defining [AAX_eProperty_RequestsAllTrackData](#) quit when processing a timeline region with no audio

Resolution: This bug is fixed as of Pro Tools 11.0.2

12.53.3.109 PTSW-183731

Failures returned by 3P AAX-AS Pls in Pre-Analyze/Render are not used by the host

Resolution: This bug is fixed as of Pro Tools 11.1

12.53.3.110 PTSW-183708

AudioSuite: plug-in parameters are not changed upon 1st click after you click Bypass. [Win]

Resolution: This was found to be an issue in certain plug-ins with JUCE-based GUI implementations. In JUCE, the real-time variants of the modifiers key getter method can cause seemingly unrelated problems with the responsiveness of the GUI. In this instance, the symptom was that plug-in parameters would not be changed on the first click inside the GUI window.

The workaround for this issue, and for other unusual GUI behavior in these plug-ins, is to always use `juce::ModifierKeys::getCurrentModifiers();` do not use `juce::ModifierKeys::getCurrentRealtime();`.

12.53.3.111 PTSW-168222

Sample rate specific plug-ins cause Pro Tools to throw a misleading error message when opened in non-supported sample rate sessions

Resolution: This bug is fixed as of Pro Tools 11.1

12.53.3.112 PTSW-165992

Make automation link by Parameter ID instead of Parameter Name. Fall-back to Parameter Name if no match

Resolution: This behavior is supported starting in Pro Tools 11.1

12.53.3.113 PTSW-163739

AudioSuite works incorrectly in Clip List mode.

Resolution: This bug is fixed as of Pro Tools 12

12.53.3.114 PTSW-161674

Stereo instrument plug-ins: "MIDI Node" field in plug-in window header disappears when insert is dragged to a new slot

Resolution: This bug is unresolved in Pro Tools and will not be fixed for the foreseeable future.

12.53.3.115 PTSW-160778

After making a Preview pass, AudioSuite plug-ins no longer make calls to InitOutputBounds()

Resolution: This bug is fixed as of Pro Tools 10.2.1

12.53.3.116 PTSW-160620

AAX plug-ins receive meaningless Clock data on Native decks, and less-than-ideal data on DSP decks

Resolution: This bug is fixed as of Pro Tools 10.2

12.53.3.117 PTSW-159702

AAX VI Issue - All AAX VIs do not have MIDI Nodes

Resolution: This bug is fixed as of Pro Tools 10.2

12.53.3.118 PTSW-159700

AAX VI Issue - Instrument Tracks do not automatically map to the AAX VI that is instantiated on them

Resolution: This bug is fixed as of Pro Tools 10.2

12.53.3.119 PTSW-159524

Incorrect error message when power is not connected to HDX card (EDIT: occurs with pre-A1 HDX prototypes only)

Resolution: This bug will not be fixed

12.53.3.120 PTSW-158119

Some plug-ins' DSP Instance counts are much lower in Pro Tools 10.2 than in Pro Tools 10.1

Resolution: This issue affects plug-ins that employ more than one buffered data port and that support many instances per DSP chip on HDX. As of Pro Tools 10.2, there is a limit of 164 buffered data ports per DSP (this is equal to the total I/O limit per DSP.)

To work around this issue, use as few data ports in your plug-in's algorithm context as possible. Note that DMA transfers on HDX occur in 128-byte chunks, so packet sizes below 128 bytes do not increase transfer efficiency on HDX.

See this forum post for more information: <https://developer.digidesign.com/index.php?L1=5&L2=13&L3=56&LC=/viewtopic.php?f=93&t=228>

12.53.3.121 PTSW-157745

Plug-ins write automation with pairs of updates, causing undesired "stepping" in recorded automation

Resolution: This bug is fixed as of Pro Tools 10.2 and 10.1.1

12.53.3.122 PTSW-157518

Poor plug-in performance with multiple processors selected; plug-ins are not consistently assigned to the same worker/thread by DAE, leading to cache thrashing.

Resolution: This bug is fixed as of the audio engine changes in Pro Tools 11

12.53.3.123 PTSW-157012

AAX DSP plug-ins with same DLL name are not properly labeled in the System Usage window

Resolution: This bug is fixed as of Pro Tools 10.2

12.53.3.124 PTSW-156310

Mouse cursor does not reliably update when positioned over plug-ins. Instead the mouse cursor shows the current Edit Tool.

Resolution: This bug is fixed as of Pro Tools 10.2

12.53.3.125 PTSW-156286

GUI elements fill window in some 3P AAX plug-ins GUIs on Windows

Resolution: This bug is fixed as of Pro Tools 10.2

12.53.3.126 PTSW-156216

`pluginGestalt_SupportsControlChangesInThread` is not properly implemented for AAX plug-ins

Resolution: Parameter updates are handled by a non-main thread for all AAX plug-ins as of Pro Tools 10.1

12.53.3.127 PTSW-156195

Silent failure when plug-ins attempt to register components with different platform support

Resolution: This bug is not yet resolved. This is an expected constraint, but the silent failure is unexpected

12.53.3.128 PTSW-156035

`GetCurrentTDMSSampleLocation()` returns the wrong value.

Resolution: This bug is fixed as of Pro Tools 10.2

12.53.3.129 PTSW-155300 / PT-218458

When an AudioSuite plug-in modifies the output audio length, the audio is not positioned at the correct location

This bug is due to AudioSuite handles processing. A plug-in that modifies the output audio length may move audio from the handle region into the visible clip region, which is unexpected behavior from the user's perspective.

Resolution: This bug will not be fixed

The workaround is for plug-ins that experience this issue to disable AudioSuite handles, thereby only processing the audio that the user sees on the timeline.

12.53.3.130 PTSW-155177

eFicGestalt_GetASPreHandleLength and eFicGestalt_GetASPostHandleLength return the wrong handle length values upon a call to AnalyzeAudio with 'WHOLE FILE' mode selected

Resolution: This bug is fixed as of Pro Tools 10.2

12.53.3.131 PTSW-154361

Highlight info sent to plug-ins before GUI is created.

Resolution: This bug is fixed as of Pro Tools 10.0

12.53.3.132 PTSW-153140

Crash on Pro Tools quit when plug-in GUI is open (OSX)

Resolution: This bug is unresolved in Pro Tools and will not be fixed for the foreseeable future. Plug-in workarounds are demonstrated in the DemoGain_GUIExtensions example plug-ins:

- a) Separating all Obj-C elements into a separate bundle that is loaded manually by the main plug-in bundle (see DemoGain_Cocoa)
- b) Applying an NSAutoreleasePool to the AAX GUI object destructors (see DemoGain_VST and DemoGain_JUCE)

See this forum post for more information: <https://developer.digidesign.com/index.php?L1=5&L2=13&L3=56&LC=/viewtopic.php?f=93&t=228>

12.53.3.133 PTSW-150047

AAX MIDI plug-ins do not get correct MIDI routing on Instrument tracks

Resolution: This bug is fixed as of Pro Tools 10.2

12.53.3.134 PTSW-149880

Configurations with duplicate PluginID properties are silently hidden with no error

Resolution: As of Pro Tools 10.2, duplicate PluginID properties will trigger the following DigiTrace log:

DTF_AAXHOST DTP_NORMAL

"AAXH ERROR: Attempted to add new configuration with duplicate ID: %x" existingID

12.53.3.135 PTSW-149819

MIDI packet alignment is not identical between DAE and AAX

This is a known bug in Pro Tools 10.0. This bug results in corrupted MIDI stream data to AAX plug-ins.

Resolution: This bug is fixed as of Pro Tools 10.0.1

12.53.3.136 PTSW-135536 / PT-218412

Erroneous transport location information provided to plug-ins after playback (new to PT9)

Resolution: This bug will not be fixed

12.53.3.137 PTSW-3020 / PT-218463

Groups do not follow changes to "Inserts" Globals group settings

Resolution: This bug will not be fixed

The workaround for this issue is to modify the group's settings to de-select "follow globals", then re-modify the group's settings to select "follow globals". This will apply the current Globals settings as well as any future changes to the Globals without need for additional workarounds.

12.53.3.138 RELENG-1484

Pro Tools developer build: Crash on launch when not "run as administrator" (Windows)

Resolution: This bug is unresolved

12.53.3.139 AAX-686

Re-add support for AudioSuite "progress" dialog re-naming (was supported in PT 9 and earlier) (see PTSW-159768)

Resolution: This bug is unresolved

12.53.3.140 AAX-583 / PTSW-157743

AAX SDK Win32 GUI example plug-in does not draw correctly

Resolution: Duplicate of PTSW-156286 (see above.) Resolved as of Pro Tools 10.2

12.53.4 Known Issues in Media Composer

12.53.4.1 MCDEV-2904

Optional analysis is not applied to every channel in a multi-channel selection

Resolution: This bug is fixed as of Media Composer 8.4

Discussion: When an optional analysis pass is triggered in Media Composer, only the channel that is currently represented in the AudioSuite Dialog will be analyzed. Other channels in a multi-channel selection will not be analyzed.

This issue is fixed in Media Composer 8.4; now the following behavior will occur for AudioSuite plug-ins:

- If a plug-in defines only [AAX_eProperty_RequiresAnalysis](#) then an Analyze pass will be performed before Render/Preview and the "Analyze" button will be disabled
- If a plug-in defines only [AAX_eProperty_OptionalAnalysis](#) then an Analyze pass will be performed before Render/Preview as well as when the "Analyze" button is clicked, and the "Analyze" button will be enabled
- If a plug-in defines both [AAX_eProperty_RequiresAnalysis](#) and [AAX_eProperty_OptionalAnalysis](#) then an Analyze pass will be performed before Render/Preview as well as when the "Analyze" button is clicked, and the "Analyze" button will be enabled

12.53.5 Known Issues in Control Surfaces

12.53.5.1 PT-226228

On EUCON control surfaces, Dynamics curves are not displayed if a plug-in does not provide a custom curve display range

Workaround: In order for a plug-in's Dynamics curve to be displayed, the plug-in must implement [AAX_IEffect::Parameters::GetCurveDataDisplayRange\(\)](#) for whichever Dynamics [curve types](#) it supports

12.53.5.2 PT-226227

EUCON control surfaces do not support custom EQ curve display ranges

Resolution: This feature is not yet implemented

12.53.5.3 GWSW-6694

S6: Plug-in parameter order is inverted when using ProControl page tables

Resolution: Resolved as of S6 Software 1.3

12.53.6 Known Issues in AAX Tools

For a list of known issues in AAX Tools such as [DigiShell](#) or the [AAX Plug-In Page Table Editor](#), see the dedicated ReadMe file that is distributed with each tool. Collaboration diagram for Known Issues:



12.54 Change Log

Changes between AAX SDK versions.

12.54.1 Change Log

12.54.1.1 AAX SDK 2.3.1

12.54.1.1.1 AAX Library

- Enhanced support for [AAX_CheckedResult](#) - added [AAX_CAPTURE](#), [AAX_CAPTURE_MULT](#), and [AAX_AggregateResult](#) to assist with common Describe error handling scenarios
- Updated [AAX_CMonolithicParameters::StaticDescribe\(\)](#) to use [AAX_CheckedResult](#) for error checking
- Added [AAX_CStatelessParameter](#) for "momentary" parameters which do not require state, such as tap tempo buttons which can be mapped to a control surface
- Improved tolerance for unknown parameters when building or parsing plug-in settings chunk data
- Added [AAX_DEBUGASSERT](#), [AAX_STACKTRACE](#), and [AAX_TRACEORSTACKTRACE](#) to the library of tracing and assertion macros in [AAX_ASSERT.h](#)
- Fixed warnings which would prevent compilation in Visual Studio 2015 and Visual Studio 2017 when Treat Warnings As Errors is enabled
- Removed Visual Studio 2008, Visual Studio 2010, and Xcode 3 projects

12.54.1.1.2 Definitions

- Removed guard preventing [AAX_CPP11_SUPPORT](#) from being set for PACE Fusion compiler builds
- Deprecated [AAX_EHostMode](#) - replaced by [AAX_EHostModeBits](#)

12.54.1.1.3 Documentation

- Documentation added to [EQ and Dynamics Curve Displays](#) for the EQ Curves feature in Pro Tools 2018.1
- Added [Checking Results](#) and [Describe Validation](#) sections to [Description callback](#)
- Added [Building your plug-in installer](#) section to [Distributing Your AAX Plug-In](#), including information about bundling Track Presets with the plug-in installer

12.54.1.1.4 Example plug-ins

- Updated all example plug-ins' Describe routines to use [AAX_CheckedResult](#) for error checking
- Updated some example plug-ins' parameter registration code in [EffectInit\(\)](#) with a safer parameter creation and release style using std::unique_ptr
- Updated the [RectiFi](#) example plug-in to match the current shipping version of Avid's Recti-Fi plug-in
- [DemoGain_UpMixer](#) now converts arbitrarily between all stem formats, both wider and narrower
- Removed Visual Studio 2008, Visual Studio 2010, and Xcode 3 projects
- Common Xcode settings updated with "macosx10.11" base SDK and "10.9" deployment target
- Added [AAX](#) DSP for higher stem formats in [DemoGain_Multichannel](#) and [DemoGain_UpMixer](#)

12.54.1.1.5 Extensions

- Updated the VSTGUI extension and example plug-in to use VSTGUI 4.3

12.54.1.1.6 Interface

- New interfaces:
 - [AAX_IACFPageTable_V2](#), with methods accessed through [AAX_IPageTable](#)
 - [AAX_IACFHostServices_V3](#), with methods typically accessed through the macros in [AAX_Assert.h](#)

See [Host Support](#) for host support information

12.54.1.1.7 Utilities

- Added reference count tracing logic to `AAX_CACFUnknown.cpp`, which can be toggled on using the `AAX_DEBUG_ACF_REFCOUNT` macro
- Added some convenience functions to [AAX_PageTableUtilities.h](#)
- Added `getLowestSampleRateInMask()` and `getMaskForSampleRate()` convenience functions

12.54.1.2 AAX SDK 2.3.0

12.54.1.2.1 AAX Library

- Added [AAX_Exception.h](#) with the [AAX::Exception](#) namespace for AAX-specific exception objects and the [AAX_CheckedResult](#) class which can be used for throwing AAX exceptions when an error is encountered.
- Added a try/catch block in the library implementation of [AAXRegisterPlugin](#) such that exceptions may safely be thrown during `Describe`
- [AAX_ICollection](#) now provides convenience methods to access an [AAX_IDescriptionHost](#) and [IACFDefinition](#), if these interfaces are supported by the host during `Describe`
- [AAX_IComponentDescriptor](#) now provides the generic `AddProcessProc()` method for specifying multiple ProcessProcs at once using a property map
- [AAX_IController](#) now provides methods for copying page table data from other effect variants or from arbitrary page table files on disk
- [AAX_IPropertyMap](#) now supports pointer-sized properties
- [AAX_IPropertyMap](#) objects can now be generated from other property map objects without requiring access to a component factory interface

12.54.1.2.2 Definitions

- Added a new stem format definition for the [7.0.2](#) format
- Removed the previous FuMa Ambisonics formats and added definitions for [second-order](#), and [third-order](#) ACN Ambisonics stems
- Added new notification types:
 - [AAX_eNotificationEvent_ParameterMappingChanged](#) (plug-in to host)
 - [AAX_eNotificationEvent_HostModeChanged](#) (host to plug-in)
- C++11 keyword compatibility macros added to [AAX.h](#)
- Removed the `AAX_AlignedDouble` definition, which was unused

12.54.1.2.3 Documentation

- New documentation:
 - [Distributing Your AAX Plug-In](#)
 - [EQ and Dynamics Curve Displays](#)
 - [Adding signposts to the DigiTrace log at run-time](#)
 - [Plug-in preset data comparison](#) for Media Composer
 - [Interactive mode for DTT](#)
 - Descriptions of [Pro Tools | Control app](#) and [Pro Tools | Dock](#) in the [Page Table Guide](#)
- There is a new process for [requesting the digital signing toolkit](#) for digitally signing AAX plug-ins
- Added a PDF print-out of this Doxygen documentation to assist with text-based searches
- Updated the [Contacting Avid](#) section of the main page to clarify the various processes for communicating with Avid
- Updated [AAX_Errors.h](#) with a list of current internal AAX host error values, which are useful for reference when troubleshooting host errors.
- Updated the [TI Guide](#) with information about using the latest version of Code Composer Studio with this AAX SDK

12.54.1.2.4 Example plug-ins

- Base Mac OS SDK setting in the common .xccconfig files is now macosx10.9
- DemoGain_Multichannel now includes an example of gain reduction metering
- DemoGain_Multichannel now supports [7.0.2](#) and [First-order](#), [second-order](#), and [third-order](#) Ambisonics stem formats
- DemoGain_UpMixer example plug-in added to demonstrate a width-changing effect
- The DemoMIDI_NoteOn example plug-in algorithm now supports note hold

12.54.1.2.5 Interface

- New interfaces:
 - [AAX_IACFComponentDescriptor_V3](#), with methods accessed through [AAX_IComponentDescriptor](#)
 - [AAX_IACFDescriptionHost](#), with methods accessed through [AAX_IDescriptionHost](#)
 - [AAX_IACFEffectorParameters_V4](#), with methods accessed through [AAX_IEffectParameters](#)
 - [AAX_IACFFeatureInfo](#), with methods accessed through [AAX_IFeatureInfo](#)
 - [AAX_IACFPageTable](#), with methods accessed through [AAX_IPageTable](#)
 - [AAX_IACFPageTableController](#), with methods accessed through [AAX_IController](#)
 - [AAX_IACFPropertyMap_V3](#), with methods accessed through [AAX_IPropertyMap](#)

See [Host Support](#) for host support information

- Added the concept of a host "feature" which can be queried during Describe execution using [AAX_I<DescriptionHost](#) and [AAX_IFeatureInfo](#)

12.54.1.2.6 Resolved bugs

- Resolved [AAXSDK-533](#): AAXLibrary compiles with warnings in VS2015 / VS2017
- Resolved [AAXSDK-514](#): Using collection-level properties leads to a leaked ACF object
- Fixed bugs with taper delegates when the minimum and maximum values are equal
- Some unnecessary headers removed or converted to forward declarations

12.54.1.3 AAX SDK 2.2.2

12.54.1.3.1 AAX Library

- Added new methods to [AAX_IParameter](#) for easier conversion between logical and normalized parameter values
- Re-named [AAX_CParameterManager::ControlIndexFromID\(\)](#) to [AAX_CParameterManager::GetParameterIndex\(\)](#)
- Added AAX Library project for Visual Studio 2013
- Added warning exclusion for C4738 to 32-bit Release configuration of the AAX Library project on Windows to fix a treat-warnings-as-errors build failure that can occur in this configuration when linking statically to the MSVC run-time libraries

12.54.1.3.2 Definitions

- Added new stem format selectors for the following stem formats:
 - The [7.1.2](#) speaker configuration
 - [First-order](#), [second-order](#), and [third-order](#) Ambisonics
- Added a new notification type for information regarding the host's delay compensation state: [AAX_eNotificationEvent_DelayCompensationState](#)
- Added a new [input data port type](#) property for ports which request [incrementally-buffered](#) packet delivery
- Added a property to allow different AAX DSP plug-in types to share the same DSP chip even if [AAX_eProperty_TI_MaxInstancesPerChip](#) is declared: [AAX_eProperty_TI_ForceAllowChipSharing](#)

12.54.1.3.3 Documentation

- Added specific details about display hardware to the [VENUE Guide](#)

12.54.1.3.4 Example plug-ins

- Added the [DemoGain_Multichannel](#) example plug-in
- Updated page tables of all example plug-ins
- Example plug-in Xcode projects now use C++11 and libc++ by default
- Updated [DemoDelay_Hybrid](#) to fix problems with instantiation in [DSH](#) and other test hosts
- Removed multi-mono support from [DemoMIDI_Synth](#) to provide a better example of a standard VI configuration
- Updated [Recti-Fi](#) example plug-in IDs so that they will not collide with the shipping version of Recti-Fi

12.54.1.3.5 Extensions

- Updated [AAX_JuceContentView::mouseMove\(\)](#) for compatibility with Juce version 4 and higher
- Updated [AAX_CEffectGUI_VST](#) for compatibility with 32-bit plug-ins when used with VSTGUI 4.2

12.54.1.3.6 Interface

- ACF interface files updated to a more recent version of the ACF SDK

12.54.1.3.7 Resolved bugs

- [AAX_CEffectParameters::UpdateParameterNormalizedValue\(\)](#) now increments the effect change counter only when the parameter's value actually changes

12.54.1.3.8 Utilities

- New utility functions: `AAX::AsStringStemFormat()`, `AAX::AsStringStemChannel()`
- Added `AAX_SCOPE_COMPUTE_DENORMALS()` for forcing denormal float values to be calculated within a scope, rather than being treated as zero (currently implemented for Mac only)

12.54.1.4 AAX SDK 2.2.1

12.54.1.4.1 Interface

- New interfaces:
 - `AAX_IACFController_V3`

12.54.1.4.2 Documentation

- Added the [VENUE Guide](#) page
- Updated the [Page Table Guide](#)
 - Updated VENUE information: Added information about `VENUE | S6L` and `VENUE | S3L-X` and removed information about VENUE systems which do not support AAX plug-ins
 - Added information for S6, including details about the '`Av46`' page table type and a new section on [Center Section Parameter Mapping in S6 Expand Mode](#)
 - Added information about quickly [Testing new XML resources](#)
- Updated the documentation for [Plug-in type conversion](#), including a new section describing [Type deprecation](#)
- Fixed image display problems on the [DSH Guide](#) page
- Added pre-built HDX DLL files to the SDK for all example plug-ins which support AAX DSP

Note

The example plug-ins' Visual Studio projects now include a `PostBuildEvent` command which will copy the plug-in's HDX DLL from the project's `TI/bin/Release` folder to the built `.aaxplugin`'s `Resources` folder.

- Additional minor example plug-in fixes
 - Removed unnecessary build phases and framework dependencies from the plug-ins' Xcode projects
 - Removed "%AAX" from the example plug-ins' display names
 - Changed the guard for AAX DSP cycle count declarations to check for the definition of the `AAX_← TI_BINARY_IN_DEVELOPMENT` preprocessor symbol before adding cycle counts to the plug-in's description
 - Added "example" to the names of all example plug-ins

12.54.1.4.3 AAX Library

- Extended `AAX_CParameter::GetValueAsString()` and `AAX_CParameter::SetValueWithString()` with support for all value types
- Fixed the specialization of `AAX_CPacket::GetPtr()` for `void*` so that it is called when the `void*` version of the function template is requested

12.54.1.4.4 Definitions

- Added `AAX_ePlugInStrings_ClipNameSuffix`
- Added a definition of the `TI_VERSION` preprocessor macro for the TI DSP compiler in `AAX.h`

12.54.1.5 AAX SDK 2.2.0

12.54.1.5.1 Interface

- New interfaces:
 - [AAX_IACFEffectParameters_V3](#)
 - [AAX_IACFHostProcessor_V2](#)
 - [AAX_IACFHostProcessorDelegate_V3](#)
 - [AAX_IACFHostServices_V2](#)
 - [AAX_IACFViewContainer_V2](#)

12.54.1.5.2 Directory changes

- Moved common processing classes for the SDK example plug-ins to ExamplePlugins/Common/Processing Classes
- Moved MIDI logging utilities to the Extensions folder
- Moved [AAX_CMonolithicParameters](#) to the Extensions folder and removed it from the AAX Library

12.54.1.5.3 Extensions

- Changed VST project to use the newest version of VSTGUI sources - VSTGUI 4.2
- Created Visual Studio 2012 projects for GUI Extensions
- Fixed [AAX_CMonolithicParameters](#) so that it correctly supports [AAX_eConstraintLocationMask_DataModel](#)

Note

This value is **required** for all plug-ins that share memory between their data model and algorithm callback

- Updated [AAX_CMonolithicParameters](#) to include parameter value synchronization
- Updated [AAX_CMonolithicParameters](#) to support Hybrid and include a state counter field

12.54.1.5.4 Definitions

- Changed name of [AAX_eProperty_StoreXMLPageTablesByType](#) to [AAX_eProperty_StoreXMLPageTablesByEffect](#) to best reflect the actual behavior of this property
- Replaced [AAX_EPlugInCategory_Effect](#) category (erroneously removed in AAX SDK 2.1)

12.54.1.5.5 Utilities

- Added utilities for atomic operations and a thread-safe FIFO queue class: [AAX_CAtomicQueue](#)
- Added AAX stacktrace logging support to make plug-in debugging easier: see [AAX_STACKTRACE](#) and [AAX_TRACEORSTACKTRACE](#)
- Added a utility for locating the .aaxplugin bundle to provide an ability to access resources in the bundle

12.54.1.5.6 AAX Library

- Created an AAX Library project for Visual Studio 2012
- Created a libc++ target in the AAX Library Xcode project
- Resolved "incompatible ms_struct" warning in Xcode 6; removed [AAX_ALIGN_FILE_ALG](#) from inappropriate locations such as virtual classes that do not cross library boundaries

- Added [AAX_IParameterValue](#), an abstract value class for parameter data, and refactored [AAX_CParameter](#) to use this interface
- Re-named [AAX_CIInstrumentParameters](#) to [AAX_CMonolithicParameters](#) (see the [Extensions](#) section for more information)
- Added an [AAX_CStateDisplayDelegate](#) constructor taking `std::vector<AAX_IString*>`
- Added an [AAX_CParameter](#) constructor taking [AAX_IString](#) as an identifier
- Added hex conversion methods to [AAX_CString](#)
- Fixed chunk size error handling in [AAX_CChunkDataParser](#)

12.54.1.5.7 Example plug-ins

- Added [DemoMIDI_Synth](#) and [DemoMIDI_Synth_AuxOutput](#) plug-ins
- Created Visual Studio 2012 projects for all example plug-ins
- Added EUCON page tables for all example plug-ins
- Various fixes for modifier-click event handling in example plug-ins
- Updated the example plug-in projects so that all built plug-in bundle filenames include "_Example"
- Corrected input/output property usage in HostProcessor example plug-ins
- Fixed multi-channel processing in [DemoDelay_HostProcessor](#)
- Fixed a bug with dynamic processing in [DemoMIDI_NoteOn](#) example plug-in
- Fixed [DemoGain_GUIExtensions](#) Win32 example plug-in GUI so that it is correctly displayed in Windows 8

12.54.1.5.8 Documentation

- Added [Media Composer Guide](#)
- Updated [Host Support](#) documentation for latest AAX host versions
- Updated the [Page Table Guide](#)
 - Updated [EUCON Page Tables](#) documentation
 - Updated [Avid Center Section Page Tables](#) documentation with tables mapping the EQ, Comp/Lim, and Exp/Gate table indices to their respective functions
- Updated [MIDI node](#) documentation
- Added new documentation pages for [Parameter update timing](#) and [Parameter automation](#)
- Improved [Presets and settings management](#) documentation
- Documented the [plug-in caching](#) behavior in Pro Tools
- Added documentation for optimizing an AAX DSP plug-in by using a hard-coded buffer size in the algorithm callback/ See the [Refactoring conditionals and branches](#) section of the [TI Guide](#)

12.54.1.6 AAX SDK 2.1.1

12.54.1.6.1 Definitions

- Explicitly removed support for the SDK's C99Compatibility headers in Microsoft Visual C++ 10.0 and later

12.54.1.6.2 DSP

- Added support and documentation for compiling AAX DSP plug-ins using Code Composer Studio 5
- Updated all example plug-in projects for use with Code Composer Studio 5

12.54.1.6.3 Documentation

- Extended the [parameter update documentation pages](#) with sequence diagrams and further information about linked parameter behavior
- Added guides for [DigiTrace](#) and [DSH](#)
- Added a reference list of [AAX interfaces](#)

12.54.1.7 AAX SDK 2.1.0

12.54.1.7.1 Interface

- New method added to [AAx_IACFTransport_V2](#) : `IsMetronomeEnabled()`

12.54.1.7.2 AAX Library

- New methods in [AAx_CString](#) for direct copy from, assignment to, and comparison with `std::string`
- Fixed many implicit sign conversions
- Added `const` qualification to some [AAx_C...](#) methods
- Updated [AAx_IParameter::GetValueAsString\(\)](#) to take a pointer-to [AAx_IString](#) (was lvalue ref)
- Fixed a bug in [AAx_CEffectParameters::GetParameterNameOfLength\(\)](#); the method now correctly truncates a parameter name if the requested length is shorter than the shortest available abbreviated name
- Treat Warnings As Errors enabled in AAX Library projects
- clang pragmas added to avoid warnings for non-virtual destructors in ACF interface classes (cf. Microsoft COM)
- Xcode 3 project added for the AAX Library

12.54.1.7.3 Definitions

- Alignment of [AAx_CMidiPacket](#) and [AAx_CMidiStream](#) on 32-bit OS X is now explicitly set using `#pragma options align=power` to maintain backwards-compatibility with earlier versions of Pro Tools
- New property added: [AAx_eProperty_RequiresChunkCallsOnMainThread](#)
- New property added: [AAx_eProperty_Constraint_AlwaysProcess](#)
- Converted [AAx_eProperty_Related_Plugin_List](#) (property #22) to dedicated [DSP](#) and [Native](#) versions
- Re-named [AAx_eProperty_AudioBufferLength](#) to [AAx_eProperty_DSP_AudioBufferLength](#)
- Added new [AAx_ECurveType](#) selector: [AAx_eCurveType_Reduction](#)
- Added various new selectors to [AAx_ENotificationEvent](#)
- Updated [AAx_STEM_FORMAT](#) macros to allow negative index values
- Added new error codes to [AAx_EError](#)

12.54.1.7.4 Utilities

- New utility functions: `AAX::IsAvidNotification()`, `AAX_IsASCII()`, `AAX_AsStringFourChar()`
- `AAX_ASSERT` and `AAX_TRACE` now require a trailing semicolon
- Re-named `LIMIT` to `AAX_LIMIT`
- Removed unused extended-80 conversion utilities

12.54.1.7.5 Extensions

- Resolved issue in which VSTGUI v4 key events were not received on Windows
- Xcode 3 projects added for the Juce and VSTGUI extension libraries

12.54.1.7.6 Documentation

- .pdf documentation moved to Doxygen
- Added several new sample plug-ins
- Expanded documentation for Host Processor and AAX Hybrid

12.54.1.8 AAX SDK 2.0.1

12.54.1.9 AAX SDK 2.0.0

12.54.1.9.1 AAX Library

- Added support for the AAX Hybrid processing architecture
- Added methods for better access to global MIDI data from `AAX_IEffectParameters`
- Extended the `AAX_ITransport` interface with several new methods
- Host Processor plug-ins can now trigger an analysis pass programmatically

12.54.1.9.2 Definitions

- Added new selectors to `AAX_ENotificationEvent` for state information during AudioSuite, bounce, and restore events
- AudioSuite reverb and delay plug-ins may opt out of the "Reverse" processing mode

12.54.1.9.3 Algorithm

- Support for temporary algorithm data blocks

12.54.1.10 AAX SDK 1.5.0

12.54.1.10.1 AAX Library

- Plug-ins now receive a different notification when receiving chunks from session and preset loads
- Aux output stems now support up to 256 output channels
- Added alpha versions of V2 interfaces
- Added projects for Visual Studio 2005 and 2008

12.54.1.11 AAX SDK 1.0.6

12.54.1.11.1 Documentation

- 64-bit targets enabled for the AAX Library and sample plug-ins

12.54.1.11.2 AAX Library

- Changed scope of some methods in [AAX_CEffectParameters](#) and [AAX_CEffectGUI](#)
- New 8 byte structure alignment added to [AAX.h](#)
- Changed the scope of some chunk parser items
- Clock context field is set to be synchronized across multiple plug-in instances
- Support for multiple input MIDI nodes
- Support for multiple named Aux Outputs ([AAX_CInstrumentParameters](#))
- Instrument parameters no longer uses host generated GUI by default

12.54.1.11.3 DSP

- Algorithm initialization routine now has 5 seconds to execute

12.54.1.12 AAX SDK 1.0.5

12.54.1.12.1 Directory Changes

- Removed 3 files in /ExamplePlugIns/Common
- Added [AAX_UtilsNative.h](#) and [AAX_Version.h](#)
- Moved AAXLog(), AAXLogf(), and isParameterIDEqual() to [AAX_UtilsNative.h](#)

12.54.1.12.2 Documentation

- Fixed instance tracking bugs in DemoGain_BackGround
- Added a time-stamp parameter to DemoMIDI_NoteOn
- Added MIDI-through to DemoMIDI_NoteOn
- Added DemoGain_DMA sample plug-in

12.54.1.12.3 AAX Library

- Changed scope of some methods in [AAX_CEffectParameters](#)
- Set default number of steps in [AAX_CParameter.h](#) to non-zero
- Renamed enum AAX_EConstraintLocation to AAX_EConstraintLocationMask

12.54.1.12.4 DSP

- Larger buffer size allowed on TI
- Support for DLL chip affinity in Pro Tools 10.2 and higher
- New AAX_INT_LO and AAX_INT_HI utilities defined

12.54.1.13 AAX SDK 1.0.4

12.54.1.13.1 Describe

- Multi-mono support constraint property added
 - Will be supported in DAE versions 10.2 and higher

12.54.1.13.2 AAX Library

- AAX_CInstrumentParameters class added as helper for monolithic instruments
- AAX_CTimestamp type changed to signed 64-bit integer
- Maximum string length support added to binary display delegate

12.54.1.13.3 Documentation

- Resolved several DemoGain_GUIExtensions example plug-in bugs and improved parity with expected Pro Tools plug-in GUI features
- Added DemoMIDI_Sampler example plug-in
- Added /TI/SignalProcessing directory with example signal processing utilities
- Added new "AAX for Pro Tools" document (still in progress)

12.54.1.14 AAX SDK 1.0.3

12.54.1.14.1 Describe

- Added "deprecated type" feature for swapping in new Effect types
- Removed AAX_eProperty_TI_UncachedCycleCount
- Removed AAX_eProperty_UseSmallPreviewBuffer, as this property is now mandatory

12.54.1.14.2 Algorithm

- Established 1024 as the maximum expected audio buffer length for any AAX plug-in
- Created new instance initialization action flag for instance reset events

12.54.1.14.3 AAX Library

- Fixed reference-counting bug in [AAXRegisterPlugin\(\)](#)

12.54.1.14.4 DSP

- Extra software pipeline information added to CCS asm output by default
- External memory support added to default CommonPlugIn_Linkercmd.cmd file
 - ExtendedPlugIn_Linkercmd.cmd is now deprecated

12.54.1.14.5 Utilities

- DigiTrace facility for AAX_Assert changed from DTF_TIPUGINS to DTF_AAXPLUGINS
- Added example DTT script for signal cancellation testing to Development builds
- Added DSP information tooltip feature to plug-in window header in Pro Tools

12.54.1.14.6 Documentation

- Win32 GUI example plug-in added to the SDK
- Basic coefficient smoothing example plug-in added to SDK
- Side Chain and Auxiliary Output Stem information page added to Doxygen
- Resolved SetControlHighlightInfo() naming inconsistency in sample plug-ins
- Expanded GUI information in AAX Manual

12.54.1.15 AAX SDK 1.0.2

12.54.1.15.1 AAX Library

- Moved AAX Library source to /Libs directory
- Added complete library source code and project files
- Removed pre-compiled AAX library binaries

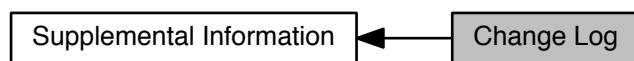
12.54.1.15.2 Documentation

- Added correct mouse event handling logic to DemoGain_GUIExtensions plug-ins
- Added meters to DemoGain_Cocoa
- New TI optimization case studies added to the TI Guide document

12.54.1.15.3 Resolved bugs

- PTSW-149745
 - Loading code into external DSP memory is functional as of TI Shell build 10.1x828

Collaboration diagram for Change Log:



12.55 Example Plug-Ins

Descriptions of the SDK's example plug-ins.

12.55.1 SDK Example plug-ins

This SDK includes the following example plug-ins. These plug-ins are designed to demonstrate good AAX plug-in design with varying levels of complexity.

In general, the SDK includes one basic version of each example plug-in, as well as multiple variations on this basic version. Each of these variations demonstrates a particular feature or design approach. To see the specific changes that were made to implement a feature, compare the example plug-in variant that demonstrates the feature to the basic version of the plug-in.

Aside from the GUI Extension examples, which are designed to work with third-party GUI frameworks, each sample plug-in should successfully compile "out of the box". However, you may receive compilation errors during the plug-ins' post-build copy step due to the fact that compiled TI DLLs are not included with this SDK.

12.55.1.1 Basic examples

These plug-ins provide complete working examples of AAX plug-ins without a lot of extra features. Use these plug-ins as a starting point for understanding [AAX](#).

12.55.1.1.1 DemoGain

DemoGain is the simplest example plug-in, incorporating a mono algorithm with gain and bypass parameters.

12.55.1.1.2 DemoDist

DemoDist demonstrates some more sophisticated techniques such as coefficient calculation and packaging, private data allocation, and multiple stem format support. DemoDist also demonstrates some basic optimization strategies for improving real-time algorithmic performance.

12.55.1.1.3 DemoDelay

DemoDelay implements a basic delay algorithm. The variants of this example demonstrate a variety of alternative processing features provided by [AAX](#).

12.55.1.1.4 DemoMIDI_NoteOn

DemoMIDI_NoteOn demonstrates basic MIDI input functionality. The example will create a step function with every Note On and Note Off message it receives. It also shows how to handle MIDI packages in the Data Model by overriding the [AAEFFECTPARAMETERS::UPDATEMIDINODES\(\)](#) method.

12.55.1.1.5 RectiFi

This is a fully ported version of the Recti-Fi plug-in from Avid's D-Fi suite. For more information about Recti-Fi, see <http://www.avid.com/plugins/d-fi>

Note

The SDK's Recti-Fi example plug-in is currently out of date and does not accurately represent Avid's shipping Recti-Fi plug-in.

12.55.1.2 Feature examples

Each of these plug-ins is a slight variation on one of the [Basic examples](#). Each feature example plug-in demonstrates a specific feature or a possible alternative design approach for the plug-in. Compare these plug-ins with the

corresponding basic example plug-in when you want to understand how a feature or design should be applied to your own AAX plug-ins.

12.55.1.2.1 DemoGain_GUIExtensions

These examples demonstrate the use of various native and third-party GUI frameworks with [AAX](#). The examples that use third-party frameworks are configured to link to static libraries that combine the SDK's **GUI Extensions** (located in /Extensions/GUI) and the applicable third-party GUI framework. These libraries are not included in the SDK, and you will need to install the applicable framework SDK before it will be possible to compile these example plug-ins.

12.55.1.2.2 DemoGain_LinkedParameters

This example demonstrates parameter linking. The plug-in is a stereo version of DemoGain, with options to link the left and right channels in two different modes.

12.55.1.2.3 DemoGain_Smoothed

This example demonstrates efficient algorithmic coefficient smoothing using a slight variation on the basic DemoGain plug-in algorithm.

12.55.1.2.4 DemoGain_Background

This example demonstrates a background routine for algorithm processing. This example also uses the [AAX direct data interface](#) for communicating algorithmic delay to the plug-in's controller.

12.55.1.2.5 DemoGain_DMA

This example includes two Effects that demonstrate use of the Scatter/Gather and Burst DMA facilities in [AAX](#).

12.55.1.2.6 DemoGain_Multichannel

This example demonstrates a multichannel plug-in configuration supporting all available point source stem formats. This plug-in also includes a simple example of gain-reduction metering, which can be used to test host features which use this data such as the [gain reduction meters](#) in Pro Tools.

12.55.1.2.7 DemoGain_UpMixer

This example demonstrates conversion between different stem formats

12.55.1.2.8 DemoGain_ParamValueInfo

This example demonstrates an implementation of the [GetParameterValueInfo\(\)](#) method, which is used to properly display certain parameter details on attached control surfaces. See [Avid Center Section Page Tables](#) in the [Page Table Guide](#).

12.55.1.2.9 DemoDist_GenCoef

This example demonstrates an alternative approach to parameter update handling. It bypasses the packet dispatcher helper class and directly overrides [UpdateParameterNormalizedValue\(\)](#) and [GenerateCoefficients\(\)](#). This approach may be appropriate for plug-ins that involve complex mapping between parameter updates, coefficient generation algorithms, and coefficient data packets.

12.55.1.2.10 DemoDelay_HostProcessor

This example includes two Effects that demonstrate the optional [Offline processing interface](#) for advanced offline processing features. One Effect implements a simple offline delay line, while the other Effect implements the same delay line but compensates for its delay when rendering to the timeline. This demonstrates how to manually compensate for inherent algorithmic delay in an offline processor.

Note

The output of offline plug-ins that do not use the [Offline processing interface](#) will be automatically adjusted by the host host to account for any declared latency. The manual compensation technique demonstrated by DemoDelay_HostProcessor is **only** necessary in plug-ins that implement the [Offline processing interface](#), e.g. plug-ins that require nonlinear offline processing features.

12.55.1.2.11 DemoDelay_Hybrid

This example demonstrates the optional [Hybrid Processing architecture](#) architecture for AAX plug-ins. This plug-in implements a short delay line that is rendered in the high-latency hybrid context. It can be built and run for either AAX Native or AAX DSP.

12.55.1.2.12 DemoDelay_DynamicLatencyComp

This example demonstrates how to properly handle algorithmic latency changes at run-time. It uses a delay line to emulate a latency-inducing algorithm with varying latency based on the delay parameter setting. When the plug-in's latency compensation feature is enabled it declares this latency to the host.

Host Compatibility Notes The DemoDelay_DynamicLatencyComp example is compatible with Pro Tools 11.1 and higher.

12.55.1.2.13 DemoMIDI_Synth

A basic synthesizer plug-in demonstrating use of an external object to manage the plug-in's state. AAX Native plug-ins that are designed to work with a cross-format framework may use a similar design. This plug-in uses [AAX_CMonolithicParameters](#) and therefore is AAX Native only.

12.55.1.2.14 DemoMIDI_Synth_AuxOutput

A variation on [DemoMIDI_Synth](#) demonstrating the [Auxiliary Output Stems](#) feature. This instrument plug-in supports four independently-routable synthesizer objects.

12.55.1.2.15 DemoMIDI_Sampler

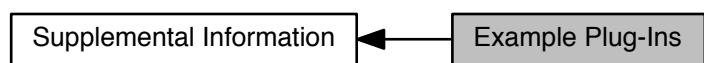
This simple "drum machine" style sampler plug-in demonstrates sample-accurate global and local MIDI input and the MIDI Transport interface. This plug-in uses [AAX_CMonolithicParameters](#) and therefore is AAX Native only.

12.55.1.3 deprecated_examples

12.55.1.3.1 DemoGain_Delay

Deprecated The DemoGain_Delay example is deprecated. See DemoDelay_HostProcessor

Collaboration diagram for Example Plug-Ins:



12.56 VENUE Guide

Details about using AAX plug-ins in VENUE live sound systems.

12.56.1 Contents

- [About this document](#)
- [Overview of VENUE](#)
- [VENUE systems](#)
- [Host environment](#)
- [AAX feature support and compatibility](#)
- [VENUE Plug-in installer specification](#)
- [Additional plug-in guidelines](#)
- [System details](#)
- [Additional Information](#)

12.56.2 About this document

This guide discusses specific details related to creating AAX plug-ins which are compatible with Avid VENUE systems.

This guide includes a general overview of the new VENUE architecture as it pertains to plug-ins, a set of guidelines for developing compatible plug-ins, and details for creating full-featured plug-in installers for VENUE.

Note

Any reference in this document to "VENUE" refers specifically to VENUE | S6L, and VENUE | S3L systems. Older VENUE systems such as VENUE Profile, D-Show, and SC48 are not compatible with AAX plug-ins and are not considered in this document.

12.56.3 Overview of VENUE

VENUE is Avid's product line aimed at live sound users. VENUE systems are modular, with audio engine, control surface, console, I/O, and external GUI units.

VENUE offers plug-in racks to utilize the power of AAX DSP plug-ins. As virtual outboard racks inside the VENUE system, the plug-in racks allow users to take their AAX DSP plug-ins out of the studio and into a live performance.

Figure 1: The main VENUE software interface

Figure 2: VENUE plug-in rack

Using the VENUE GUI, an operator is able to see thumbnails for each of the plug-ins in a plug-in rack. An operator can choose to zoom in on a plug-in from this rack view and, from there, graphically control the plug-in with the mouse, keyboard, or touch-screen. Only one plug-in interface can be displayed at a time in this mode.

Figure 3: Plug-in zoom view

12.56.4 VENUE systems

This section will provide a brief overview of Avid's AAX-compatible VENUE systems. For more information about the features, functionality, and use of these systems see the VENUE user documentation.

12.56.4.1 VENUE | S6L

VENUE | S6L is a modular system designed to take on the world's most demanding tours and events with ease. Offering unprecedented processing capabilities - with over 300 processing channels - S6L delivers unrelenting performance and reliability through its advanced engine design and backs it up with modern touchscreen workflows and scalability to meet any challenge.

The S6L engine contains dedicated HDX-powered DSPs handling all plug-in processing and supports 64-bit AAX DSP plug-ins.

12.56.4.2 VENUE | S3L-X

The VENUE | S3L-X System is a modular live sound solution including an HDX-powered processing engine, scalable remote I/O, and a EUCON-enabled control surface.

At the heart of the S3L system lies the E3 engine. The E3 runs Windows Embedded and a version of VENUE software that can load AAX DSP plug-ins onto a built-in HDX platform. Accompanying the E3 engine is the [S3](#) control surface and one or more Stage 16 remote I/O boxes.

Most system parameters, including plug-ins, can be controlled using either the on-screen VENUE software or directly via encoders on the S3 control surface. When being used as part of a VENUE | S3L-X system, the S3 control surface is divided into three main sections:

- A - Channel Section The Channel section provides control of Input Channels, FX Returns, Output Channels, some channel parameters (such as Input Channel Gain and Aux Send levels), and channel banking. Channels are selected using the channel Select switches next to each fader.

- B - Channel Control The eight Channel Control encoders provide control of processing functions for the currently selected Input or Output Channel. Inserted Dynamics and EQ plug-ins can be selected and adjusted in Channel Control.

- C - Global Control The eight Global Control encoders provide control of system-wide parameters, including control of plug-ins. The Global Control encoders can be placed into Insert Mode, and can then be used to select and adjust any plug-ins.

Figure 4: Main control sections on the S3 control surface

12.56.4.2.1 Using Channel Control

If a channel has an EQ, Comp/Lim, or Expander/Gate plug-in inserted on it, it can be controlled using the eight Channel Control encoders. The user can toggle between controlling the built-in Dynamics or EQ processors and the plug-in versions.

Each Input and Output Channel has built-in EQ and Comp/Lim processors. Each Input Channel also has a built-in Expander/Gate. To adjust the built-in processors, the user selects a channel, assigns a processing function to Channel Control by pressing the corresponding encoder from the Channel Control main menu, then adjusts the available parameters.

Figure 5: The Channel Control main menu

See [Center Section Parameter Mapping on VENUE | S3L-X](#) for a description of how plug-in parameters are mapped to the S3L Channel Control encoders for EQ, Compressor/Limiter, and Expander/Gate plug-ins.

12.56.5 Host environment

12.56.5.1 Audio engine

The audio engine in VENUE is based around Avid's HDX technology. Each VENUE S6L and S3L System contains a specialized HDX core card. For more information about HDX, see the [TI Guide](#). Because the VENUE architecture is so similar to HDX, AAX DSP plug-ins are cross-compatible with VENUE and most plug-ins will run seamlessly on VENUE with little or no modification.

Each VENUE system operates at a single native sample rate. VENUE supports multiple processing block sizes at this sample rate. Like in Pro Tools | HDX systems, each DSP chip in the system will only be able to load plug-ins

using a single block size; plug-ins which process using different block sizes cannot be allocated to the same DSP.

12.56.5.2 Available DSP resources

The following information reflects plug-in processing abilities of VENUE systems:

- S3L-X
 - 4 TI C6727 DSP chips are available for plug-in processing
 - 40 plug-in rack slots are available
- S6L
 - All HDX DSP cards are dedicated to plug-in processing; each HDX DSP card has 18 TI C6727 DSP Chips
 - Depending on E6L engine type, 125 (E6L-144) or 200 (E6L-192) plug-in rack slots are available

12.56.5.3 Operating system

The core host software in a VENUE system is built upon Windows Embedded 8. The installation used on VENUE systems is a customized version of Windows 8 that includes only what is necessary for the VENUE software.

Core services from Windows 8 are available, such as the Win32 API, but some advanced services may not be available. Such services include MIDI, printing, video codec, .NET, etc. If your code relies on advanced Win32 APIs, or you are in doubt about specific APIs, please contact Avid for more information.

Using unavailable services may cause a plug-in to not load (e.g. if it attempts to link against DLLs that aren't included in the Windows Embedded 8 image) or to fail during run-time. Whenever possible, before using any advanced Win32 API, you should verify the availability of the service and/or handle the fact that the service might not be functional.

See the [VENUE Plug-in installer specification](#) section for more information about ensuring that all required run-time components are available to your plug-in.

12.56.5.4 Display

VENUE S6L requires that plug-in windows be restricted to a certain size. If a plug-in exceeds this size, it will overlap VENUE's GUI and may possibly be truncated. The specifications are as follows:

- S3L-X
 - Total GUI size: 1024 W x 768 H
 - Max plug-in window size (w/o sidechain support): 749 W x 617 H
 - Max plug-in window size (with sidechain support): 749 W x 565 H
- S6L
 - Total GUI size: 1920 W x 1080 H
 - Max plug-in window size (w/o sidechain support): 1436 W x 855 H
 - Max plug-in window size (with sidechain support): 1436 W x 796 H

VENUE will dispatch a [AAX_eNotificationEvent_MaxViewSizeChanged](#) notification indicating the maximum size for a plug-in's GUI. Calls to [AAX_IViewContainer::SetViewSize\(\)](#) will fail with an error if the plug-in attempts to set its view size to a larger value than the system supports, though the plug-in's initial GUI will be displayed (and possibly truncated) at its normal size before any resize requests are made.

The actual hardware and graphical acceleration available in VENUE systems is as follows:

	S3L-X	S6L
CPU model	Celeron P4500	Core i5-2510E
GPU model	Intel HD Graphics	Intel HD Graphics 3000
DirectX support	10.1	10.1
OpenGL support	2.1	3.1
OpenCL support	None	None
Shader model	4	4.1

12.56.5.5 Page tables

- VENUE S3L-X uses 'PCTL' (ProControl) page tables
- VENUE S6L uses 'Av46', a EUCON-style page table with a 4x6 knob cell configuration.

Note

In S6L, the 'FrTL' (C|24) page table is used as a fallback when 4x6 is not available. This is only a temporary solution to support legacy plug-ins. All plug-ins targeting VENUE S6L support must support the 4x6 knob cell layout and should not rely on this C|24 fallback behavior.

Page table design guidelines Primary plug-in parameters should be located on the first page in the page tables for a surface. This is especially true for the 4x6 knob cell layout used by S6L. Users should not be required to navigate between pages for the majority of common operations.

For more information about page tables, including additional guidelines for good page table design, see the [Page Table Guide](#).

12.56.5.6 Network communications

Some plug-ins may require interaction with other devices in a network. VENUE systems have two Gigabit Ethernet ports available:

1. **ECx port** Intended for connection of VNC Viewer to control the VENUE system remotely. The IP address and network mask for this port are user-configurable in the VENUE UI.
2. **AVB port** Intended for connection of all other VENUE system components, as well as a computer running Pro Tools software. This port always uses link-local addressing. Because of AVB traffic, the effective bandwidth of this port is limited to 100 Mb/s.

Note

Plug-ins must not use a significant portion of the available bandwidth on the AVB port, since it will affect mission-critical control connections of a VENUE system.

Both S3L-X and S6L systems include the Apple Bonjour service. Plug-ins may use Bonjour for interfacing with other software in the network. Plug-ins must not install their own version of Bonjour or attempt to modify the Bonjour installation on the system.

12.56.5.7 Host environment summary

	S3L-X	S6L
Operating System	Windows Embedded 8	Windows Embedded 8
Sample Rate	48 kHz	96 kHz
Max GUI size	749 W x 617 H (no sidechain) 749 W x 565 H (sidechain)	1436 W x 855 H (no sidechain) 1436 W x 796 H (sidechain)

Page table	'PcTL'	'Av46'
------------	--------	--------

12.56.6 AAX feature support and compatibility

VENUE supports many of the same AAX features as Pro Tools. However, some features are not available in VENUE, and other features are managed differently between the two applications. This section describes how VENUE handles various optional AAX features.

12.56.6.1 Processing configurations

Architectures VENUE supports 64-bit AAX DSP plug-ins only. AAX Native and AAX Hybrid plug-ins are not supported. Plug-ins compiled for 32-bit processors are not supported, though they may be included in a VENUE-compatible .aaxplugin bundle alongside the plug-in's 64-bit binary.

Stem formats

- Mono plug-ins may be inserted as channel inserts on mono input strips and output busses.
- Stereo plug-ins can be inserted as channel inserts on stereo input strips and output busses.
- Greater-than-stereo formats are not supported by VENUE
- Multi-mono processing is not supported; an operator must use the stereo version of a plug-in in stereo processing locations.

Width-changing plug-ins Width Changing plug-ins are not allowed as inserts except in mix busses. Unlike in Pro Tools, a plug-in cannot change the output stem format of a strip or bus by using a mono-to-stereo plug-in on a mono track.

However, width-changing plug-ins are supported on output busses. For these, the outputs of the plug-in can either be routed back to an FX return or routed out to physical outputs. This functionality does not require any additional implementation specific to VENUE.

12.56.6.2 Presets and automation

Plug-In settings are persisted (saved & restored) the same way for Show files, snapshots & settings files, using a single method to extract settings and a single method to apply setting. The code uses the "chunk" APIs. That's similar to what Pro Tools does, except that for automation, VENUE uses exclusively snapshots (that is, VENUE does not record & playback individual control changes).

Many VENUE snapshots users are known to store settings for every Plug-In in every snapshot (or almost). This causes performance issues because settings are often fairly slow to load, making a snapshot recall last too long (sometimes 30 seconds or more!) whereas users expect a snapshot recall to take instantly. However, extremely frequently, settings are not actually changing from a snapshot to the next one (that is, from one snapshot to the next one, the vast majority of Plug-Ins contain the same settings). To mitigate this, VENUE calls [AAX_IEffectParameters::CompareActiveChunk\(\)](#) to determine whether a chunk from an incoming snapshot would result in any change to the plug-in's current settings. If not, the new chunk will not be loaded onto the plug-in. This optimization is extremely effective, but requires that Plug-Ins implement the [CompareActiveChunk\(\)](#) method properly at any time (in particular regardless of whether the plug-in is visible or not). With VENUE, you must implement this API very well or the Plug-In may not be controllable. This optimization will affect settings application when a show is loaded, when presets are loaded and when snapshots are recalled.

All chunks are first compared one by one to the active chunks, until one is different or an error is returned. If any chunk compare fails (not equal or error returned), then all chunks are sent in sequence.

As it is the basic method for plug-in settings manipulation, it is critical that plug-ins process chunks as accurately and efficiently as possible.

Plug-In Chunks The size of a plug-in chunk cannot exceed 64KB in VENUE. If a Plug-In requires more than 64KB of chunk data total (all chunk sizes added), settings for this plug-in won't be persisted, snapshots won't work for this

plug-in and users won't be able to load or save settings. If you can not meet this requirement, you should detect that you are running on VENUE and not declared the process type as it won't be usable.

12.56.6.3 Unsupported features

The following AAX features are not supported by VENUE. Plug-ins that require these features will not be compatible with VENUE systems. If your plug-ins use these features for advanced functionality but not for basic operation then you should document this restriction for VENUE users.

- Advanced audio routing VENUE does not support [Auxiliary Output Stems](#) from plug-ins.

Warning

[Description callback](#) calls to register auxiliary output stems will return an error code on VENUE systems, indicating that the host will not provide audio buffers for auxiliary output stems during processing. A plug-in must not attempt to write data into auxiliary output stem buffers which have not been provided by the host!

- Transport interface VENUE operates entirely in real-time and does not contain a timeline of pre-recorded audio. Therefore VENUE does not support the [AAX_ITransport](#) interface. VENUE will return [AAX_ERROR_UNIMPLEMENTED](#) to unsupported transport interface method calls.
- MIDI VENUE does not support MIDI routing to and from plug-in instances, and no [AAX MIDI features](#) are supported by VENUE.

12.56.7 VENUE Plug-in installer specification

To install plug-ins, the VENUE software includes a simple installation interface. To install a plug-in, the operator simply plugs in a USB Flash Drive with an installer for the plug-in and the plug-in will show up in an installer menu on the VENUE interface (shown below).

This menu will automatically list all the installable plug-ins on the drive. With the click of a button, the user can install the plug-ins onto his VENUE system.

Figure 6: The VENUE plug-in installer tab

For this custom installation to work properly, the plug-in installation USB Key must follow a certain layout. This layout is designed to be as flexible and as expandable as possible, giving the developer many options while retaining the simplicity that makes VENUE's plug-in installation appealing to the user. This layout is also designed to coexist on a drive with a Pro Tools plug-in install. The following is a detailed description of how the file hierarchy should be laid out.

12.56.7.1 Overview

The VENUE Plug-In installer specification is an extension of an AAX plug-in bundle, i.e. of the *MyPlugIn.aaxplugin* folder.

A standard .aaxplugin directory forms a basic, compatible plug-in installer for VENUE. See [.aaxplugin Directory Structure](#) for more information about this folder.

The following optional items can be added to the .aaxplugin folder to extend its functionality when used as a VENUE plug-in installer:

- License file that will need to be accepted by end user
- Pre-install action (either .bat script or executable or both)
- Post-install action (either .bat script or executable or both)
- Pre-uninstall action (either .bat script or executable or both)

- Post-uninstall action (either .bat script or executable or both)
- PACE Eden installer to update the version pre-installed on the VENUE system
- Factory presets
- Registry entries in a form of .reg files
- Program files to be placed in the system's C:\Program Files folder
- Plug-in thumbnails to be shown in the rack in VENUE Software UI

12.56.7.2 Directory structure

Here is a layout of the optional elements in the .aaxplugin plug-in installer directory:

- /Contents
 - *standard AAX plug-in contents*
- /Pace Eden
 - Setup.exe
 - Setup.bat
 - Version.txt
- /Program Files
 - ...
- /Thumbnails
 - *id1.bmp* (example: 424644204C41324131314C41.bmp)
 - *id2.bmp*
- /License.rtf or License.txt
- /Install_before.bat
- /Install_before.exe
- /Install_after.bat
- /Install_after.exe
- /SomeSettings.reg
- /Uninstall_before.bat
- /Uninstall_before.exe
- /Uninstall_after.bat
- /Uninstall_after.exe

12.56.7.3 Optional installer files

12.56.7.3.1 License terms

A license stored as a file of either RTF or ASCII plain text format. The file must be located in the root folder of the installer. Depending on the text file format, the file name must be either *License.rtf* or *License.txt*.

12.56.7.3.2 Registry entries

Registry settings to be applied during installation need to be stored in .reg files in the root folder of installer. All such files must be of the Windows Registry format. Particular file names does not matter.

Registry files are applied during plug-in installation.

It is important to not alter any system settings or settings of other software installed.

Note

Changes in the registry are not reverted during the plugin uninstallation.

12.56.7.3.3 Program files

Files under the *Program Files* subfolder in the plug-in installer will be copied to the system's *C:\Program Files* folder. All contents of the *Program Files* subfolder are copied as-is into *C:\Program Files*, retaining the internal folder structure of nested directories.

These changes are undone when the plug-in is uninstalled.

12.56.7.3.4 Plug-in thumbnails

VENUE uses thumbnail images to display plug-in GUIs in the plug-in rack while the full-size plug-in GUI is hidden.

If thumbnail images are not provided in the plug-in installer then VENUE will display a generic thumbnail image for the plug-in until it has been focused in Zoom Mode in the VENUE interface. In this case VENUE will create and cache a thumbnail image for the plug-in GUI the first time that it is focused.

The user may regenerate a thumbnail by right-clicking a rack with a plug-in and choosing "Recreate Thumbnail".

Including thumbnails in plug-in installers

A separate thumbnail should be provided in the plug-in installer for each variant supported by the plug-in, i.e. each unique AAX DSP ID triad registered by the plug-in. Use the "Recreate Thumbnail" feature to create the initial versions of your plug-in thumbnail images. Package these thumbnail images into your plug-in installer in order to guarantee that thumbnail images will be available to users immediately upon installing the plug-in.

Each thumbnail bitmap file is named after the following plug-in parameters:

1. [AAX_eProperty_ManufacturerID](#)
2. [AAX_eProperty_ProductID](#)
3. [AAX_eProperty_PluginID_TI](#)

All of three are converted to hexadecimal representation and concatenated to form a file name that uniquely identifies a plug-in variant. For example, the Avid Channel Strip plug-in has a thumbnail file named 41564944 43685374 434D5469.bmp (no spaces).

The "Recreate Thumbnail" feature in VENUE will ensure that the generated thumbnail images use the correct file names, resolution, and image format.

12.56.7.3.5 Actions

A plug-in installer may define custom actions for the following cases:

1. Pre-install action - executed when plug-in installation starts
2. Post-install action - executed when plug-in installation finishes
3. Pre-uninstall action - executed when plug-in uninstallation starts
4. Post-uninstall action - executed when plug-in uninstallation finishes

Each action is defined by either a .bat file or an Win32/64 executable file (.exe). Action files must be placed in the root folder of installer and use these file names:

1. Pre-install action - *Install_before.bat*, *Install_before.exe*
2. Post-install action - *Install_after.bat*, *Install_after.exe*
3. Pre-uninstall action - *Uninstall_before.bat*, *Uninstall_before.exe*
4. Post-uninstall action - *Uninstall_after.bat*, *Uninstall_after.exe*

If both .exe and .bat files are present for a certain action, then both are executed, with the .bat file being run before the .exe.

When executing an action, no exit code is tested. In order to report errors, the following needs to be done:

1. .bat files:

The following line should be used to report an error message from a script: `reg add HKCU\Software\Digidesign\tmp /f /v InstallResult /d "Error description"`

2. .exe files:

The error message needs to be added to Windows registry as a REG_SZ value in `HKEY_CURRENT_USER\Software\Digidesign\tmp` and named `InstallResult`.

The presence of the error message will abort a plug-in installation. Any error strings will be written to the VENUE logs so that Avid support will be able to see them. Error strings from these actions are not shown to the user.

12.56.7.3.6 PACE software installer

Warning

This functionality must not be used without prior approval from Avid. Before releasing **any** VENUE plug-in update with a bundled PACE installer you must contact Avid to confirm that the bundled installer will not cause issues for deployed VENUE systems.

VENUE allows plug-ins to install updated version of PACE iLok software immediately after the plug-in installation. In general, Avid tries to provide the latest Pace software with each VENUE software release and update. Therefore this step should not be necessary in most cases.

The PACE installer files must be located in *Pace Eden* subfolder of the installer. This folder must contain the following files:

- *Version.txt* containing a version of the *LDSvc.exe* PACE executable being installed. The version information must be in the form of *1.2.3.4*
- *Setup.exe* - the PACE installer itself.
- (optional) *Setup.bat* containing an installation script. Usually used to run PACE installer in a silent (no UI, no interaction) mode.

During installation, *Setup.bat*, if present, is run. Otherwise *Setup.exe* is executed with the following command line arguments:

```
Setup.exe /s /v"REINSTALLMODE=vamus REBOOT=ReallySuppress /qn"
```

If the version of installer is not higher than the version installed in system, the installation will not be performed.

An OS reboot is prompted in VENUE UI after PACE was installed.

12.56.7.4 Using a VENUE plug-in installer

In order to install a plug-in to the VENUE system, end user is expected to perform the following steps:

1. Download a VENUE Plug-in Installer(s) in a form of archive (Zip is suggested). Is it ok to have multiple plug-ins in one archive as soon as each plug-in is in own VENUE Plug-in Installer (i.e. in own .aaxplugin folder).

2. Unpack archive and copy installers to USB drive in the following way:
 - (a) "AAX Plug-Ins" folder must be placed in the root of USB drive.
 - (b) Each installer needs to be copied directly the "AAX Plug-Ins" folder. In the end, resulting folder structure will look like this:
3. Install plug-ins in a way described in documentation of a particular VENUE Software version.

12.56.8 Additional plug-in guidelines

12.56.8.1 General Reliability and Fault Tolerance

Since VENUE is a more "mission critical" type of application where there is no room for error during a live show, additional precautions have to be taken with respect to reliability of its various components. We have built provisions in VENUE to protect the system from catastrophic failure due to a plug-in crashing and bringing down the entire system. On top of this, extra care should be taken in developing stable software when targeting VENUE as a platform.

If a plug-in crashes, the user will be warned through a dialog. A crash brings down all plug-in processes, but audio keeps flowing through the console and through the DSPs, including the plug-ins' DSPs. All the effects continue to be effective, but their parameters can't be accessed or modified anymore (the show goes on...).

At this point, audio should be totally unaffected, even for the effect that caused the crash (assuming the crash took place in the host code, not the DSP code, of course). At the user's discretion, all plug-ins will be bypassed or muted (depending on where they are used in the system), any dependencies on the plug-ins' DSPs will be removed, the plug-ins' DSPs will be reset, and all the plug-ins will start again. When the rebuilding operation is complete, the user will be prompted to decide when he wishes the new plug-ins to be connected.

12.56.8.2 Plug-In Dialogs

Plug-ins should avoid invoking dialog windows in VENUE. We strongly suggest that any unnecessary dialog window your plug-in creates, whether at installation or instantiation, be removed. For VENUE-only plug-ins, we strongly suggest to not make use of any dialog windows.

Should you nevertheless need to make use of additional windows or dialogs, you need to make sure that they are front-most, so that they will not be hidden behind VENUE's GUI. The VENUE software will try to force your windows to be front-most, but it is safer if your plug-in enforces this in the first place.

12.56.8.3 Online Help

VENUE currently doesn't include any standardized help menu for plug-ins. We recommend that you use tooltips and other "live" help techniques similar to what plug-ins like ReVibe II, Reverb One, and Smack! use to help the user. For instance, when a user clicks on the "Side-Chain EQ" label of the Smack! Plug-In, here's what they see:

Figure 7: Tooltip help in Avid's Smack! plug-in

One of the major benefits of this technique is that it is supported across platforms and will work the same in all [AAX](#) hosts.

12.56.9 System details

12.56.9.1 External dependencies

[AAX](#) plug-in may rely on a presence of the following items in the VENUE system:

- Bonjour service and library

Note

Plug-in installers are forbidden from installing over or modifying the pre-installed version of Bonjour on the VENUE system.

- VC 2005 x64 runtime
- VC 2008 x64 runtime
- VC 2010 x64 runtime
- VC 2012 x64 runtime
- VC 2013 x64 runtime

Because VENUE does not execute standard software installers for plug-ins, Avid tries to keep VC runtime versions up to date relative to the moment of release of a particular VENUE Software version.

12.56.9.2 Environment variables

Both plug-in installers and actual plug-ins may rely on a presence of the following environment variables in a VENUE system:

- **DAEPLUGINSFOLDER** - is always set to the Installed Plug-ins location. Currently this is *C:\Program Files\Common Files\Digidesign\DAE\Plug-Ins*. Final backslash is absent.
- **JEX_HOST_TYPE** - equals "venue". If required, this may be used to provide a custom behavior of the plug-in when it's run on VENUE system.

12.56.9.3 Plug-in file locations

Installed Plug-Ins Located at *C:\Program Files\Common Files\Digidesign\DAE\Plug-Ins*

This folder is the only location used by VENUE software to instantiate a plug-in.

This location is different from the one used by Pro Tools and Media Composer for 64-bit [AAX](#) plug-ins. The only way for a plug-in to appear at that location is to be installed from VENUE Software's "Options">>"Plug-Ins" page; standard plug-in installers will place the plug-in into a different directory.

Note

This location may change in future VENUE software releases. Plug-ins should not make any assumptions about the install directory and should rely on the VENUE plug-in installer to place them in the correct location.

Plug-ins available for installation

- Local: Located at *C:\Program Files\Common Files\Avid\Audio\Plug-Ins*
- On USB drive: Located at (*USB drive letter*):\[AAX](#) Plug-Ins

These locations can be chosen as sources for plug-in installation on VENUE Software's "Options">>"Plug-Ins" page.

Cached plug-in installers Located at *D:\D-Show\Plug-In Installers*

Contains copies of plug-in installers installed via VENUE Software's "Options">>"Plug-Ins" page.

This location can be chosen as source for plug-in installation on VENUE Software's "Options">>"Plug-Ins" page under the name "Previous Installs".

Factory presets Located at *D:\D-Show\User Data\Effect Presets\Factory Presets*

Contains preset files for plug-ins, as well as for certain VENUE parameters. Presets are organized in folders.

Each subfolder corresponds to a particular preset type. Plug-in presets are named after the plug-in's name and the plug-in's [AAX_SPlugInChunkHeader::fProductID](#) value. For example, for an Avid Channel Strip plug-in the subfolder name is *Channel Strip [31313736]*, where "31313736" is an unsigned integer of the Channel Strip product ID.

Contents of subfolders are .tfx files of plug-in presets. Each file name will be visible to end user as a preset name.

Presets are copied into file location during a plug-in installation.

Plug-in thumbnails Located at *C:\Program Files\Digidesign\Plug-In Icons*

Contains .bmp files of plug-in thumbnails generated by VENUE Software as a result of saving current plug-in graphics into a bitmap. See [Plug-in thumbnails](#).

12.56.9.4 Installation process

12.56.9.4.1 Plug-in installation

These are the steps followed by VENUE when installing a plug-in:

1. First, a VENUE plug-in installer is cached. This is done by copying a plug-in from installation source to the Cached VENUE Plug-In Installers location. All files are copied with an exception of the "Documentation" and "Pro Tools" folders.

All of the following steps are executed from the cached installer location, not from the original source location.

2. The pre-install batch script ("Install_before.bat"), if present, is executed. Execution assumes running the script without a console window.

Note

The pre-install script must not contain any installation steps, since it's executed before the plug-in license is accepted. In general, you should only use the pre-install script for pre-install checks.

3. The pre-install executable ("Install_before.exe"), if present, is executed. Execution assumes running the executable without a console window.

Note

The pre-install executable must not perform any installation steps, since it's executed before the plug-in license is accepted. In general, you should only use the pre-install script for pre-install checks.

4. A license ("License.rtf" or "License.txt"), if present, is shown to the user. If "License.rtf" is not found, "← License.txt" is used. A license, if present, must be accepted by user; otherwise installation will be aborted.

5. All files of the VENUE plug-in installer are copied to the system Installed Plug-Ins location, keeping the .aaxplugin folder structure. Failure to copy any of the items results in installation being aborted.

6. If the plug-in installer contains a subfolder named "Program Files", its contents are copied into "C:\\Program Files". Failure to copy any of items results in installation being aborted.

7. If the plug-in installer contains a subfolder named "Contents\\Factory Presets", its contents are imported as plug-in presets. The "Factory Presets" folder must contain only valid plug-in .tfx preset files in an arbitrary folder structure. All preset files are read and copied into the system's Factory Presets location.

Note

It is important for a plug-in installer to contain only plug-in presets corresponding to plug-in being installed.

8. Plug-in thumbnails, if present, are copied from the "Thumbnails" subfolder of the installer to the system Plug-in Thumbnails location.

9. Registry files, if any, are imported. Every file with .reg extension in the root of plug-in installer is treated as a Windows Registry file and gets imported by calling

```
regedit /s "&lt;file.reg&gt;"
```

No error checking is performed.

10. The post-install batch script ("Install_after.bat"), if present, is executed. Execution assumes running the script without a console window.
11. The post-install executable ("Install_after.exe"), if present, is executed. Execution assumes running the executable without a console window.
12. The PACE software installer, if present, is run. If the version of the installer is not higher than the version installed in system, the installation is not performed.
When installing multiple plug-ins at once, PACE installation happens only after installing the final plug-in. VENUE will use the PACE installer with the highest available version among the installed plug-ins.
13. Plug-in installation is considered successful.

If errors occur during installation, the following happens:

1. Plug-in files are removed from the disk (see "File removal" section for details).
2. Cached plug-in installer is removed from the Cached VENUE Plug-in Installers location.

12.56.9.4.2 File removal

File removal happens either in case of plug-in uninstallation or in case of a failed installation cleanup.

The following happens:

1. Plug-in files, as present in Cached VENUE Plug-in Installers location, are removed from Installed Plug-Ins location.
2. Plug-in Program Files files, as present in Cached VENUE Plug-in Installers location, are removed from Installed Plug-Ins location.

12.56.9.4.3 Plug-in uninstallation

Plug-in installation process is done by VENUE Software. It removes a plug-in from the Installed Plugins location. Here's a step by step process of uninstalling plug-in:

1. The pre-uninstall batch script ("Uninstall_before.bat"), if present, is executed. Execution assumes running the script without a console window. NO return code is examined. Instead, a script is allowed to report an error string that will be visible to Avid support when examining VENUE log files. The following line should be used to write an error message from a script:

```
reg add HKCU\Software\Digidesign\tmp /f /v InstallResult /d "Error description"
```

The presence of this string means an error has occurred and a plug-in uninstallation will abort.

2. The pre-uninstall executable ("Uninstall_before.exe"), if present, is executed. Execution assumes running the executable without a console window. NO return code is examined. Instead, an executable is allowed to report an error string that will be visible to Avid support when examining VENUE log files. The error needs to be added to Windows registry as a REG_SZ value in HKEY_CURRENT_USER\Software\Digidesign\tmp and named *InstallResult*. The presence of this string means an error has occurred and a plug-in uninstallation will abort.

3. Plug-in files are removed. See [File removal](#) for details.

4. The post-uninstall batch script ("Install_after.bat"), if present, is executed. Execution assumes running the script without a console window. NO return code is examined. Instead, a script is allowed to report an error string that will be visible to Avid support when examining VENUE log files. The following line should be used to write an error message from a script:

```
reg add HKCU\Software\Digidesign\tmp /f /v InstallResult /d "Error description"
```

The presence of this string means an error has occurred and a plug-in uninstallation will abort.

5. The post-uninstall executable ("Install_after.exe"), if present, is executed. Execution assumes running the executable without a console window. NO return code is examined. Instead, an executable is allowed to report an error string that will be visible to Avid support when examining VENUE log files. The error needs to be added to Windows registry as a REG_SZ value in `HKEY_CURRENT_USER\Software\Digidesign\tmp` and named `InstallResult`. The presence of this string means an error has occurred and a plug-in uninstallation will abort.
6. Plug-in removal is complete.

Please note that plug-in being uninstalled is not being removed from the cache. Removal from the Cached VENUE Plug-in Installers is possible for plug-ins being not installed. In order to accomplish this, end user needs to go to VENUE Software's "Options">"Plug-Ins" page, right click on cached installer, and choose "Delete plug-in name".

12.56.10 Additional Information

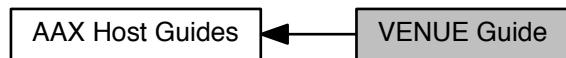
12.56.10.1 Metering

For metering displays, VENUE uses dB units referenced to VENUE's nominal operating level of +4dBu. A signal at the nominal level in VENUE (i.e. registers 0dB on the VENUE meters) will, at unity gain, generate a +4dBu analog output signal (-20dBFS digital output signal).

As a result, a signal that registers +20dB on the VENUE meters will register 0dBFS on the plug-in meters. A signal at 0 dB in VENUE will be -20dBFS in the plug-in.

To map between dBFS units used in plug-ins and dB units used in VENUE the operator simply needs to add 20 to any plug-in dBFS value.

Collaboration diagram for VENUE Guide:



Chapter 13

Namespace Documentation

13.1 AAX Namespace Reference

Namespaces

- `Exception`

AAX exception classes

Enumerations

- enum `EStatusNibble` { `eStatusNibble_NoteOff` = 0x80, `eStatusNibble_NoteOn` = 0x90, `eStatusNibble_KeyPressure` = 0xA0, `eStatusNibble_ControlChange` = 0xB0, `eStatusNibble_ChannelMode` = 0xB0, `eStatusNibble_ProgramChange` = 0xC0, `eStatusNibble_ChannelPressure` = 0xD0, `eStatusNibble_PitchBend` = 0xE0, `eStatusNibble_SystemCommon` = 0xF0, `eStatusNibble_SystemRealTime` = 0xF0 }

Values for the status nibble in a MIDI packet.

- enum `EStatusByte` { `eStatusByte_SysExBegin` = 0xF0, `eStatusByte_MTCQuarterFrame` = 0xF1, `eStatusByte_SongPosition` = 0xF2, `eStatusByte_SongSelect` = 0xF3, `eStatusByte_TuneRequest` = 0xF6, `eStatusByte_SysExEnd` = 0xF7, `eStatusByte_TimingClock` = 0xF8, `eStatusByte_Start` = 0xFA, `eStatusByte_Continue` = 0xFB, `eStatusByte_Stop` = 0xFC, `eStatusByte_ActiveSensing` = 0xFE, `eStatusByte_Reset` = 0xFF }

Values for the status byte in a MIDI packet.

- enum `EChannelModeData` { `eChannelModeData_AllSoundOff` = 120, `eChannelModeData_ResetControllers` = 121, `eChannelModeData_LocalControl` = 122, `eChannelModeData_AllNotesOff` = 123, `eChannelModeData_OmniOff` = 124, `eChannelModeData_OmniOn` = 125, `eChannelModeData_PolyOff` = 126, `eChannelModeData_PolyOn` = 127 }

Values for the first data byte in a Channel Mode Message MIDI packet.

- enum `ESpecialData` { `eSpecialData_AccentedClick` = 0x00, `eSpecialData_UnaccentedClick` = 0x01 }
- Special message data for the first data byte in a message.*
- enum `ESampleRates` { `e44100SampleRate` = 44100, `e48000SampleRate` = 48000, `e88200SampleRate` = 88200, `e96000SampleRate` = 96000, `e176400SampleRate` = 176400, `e192000SampleRate` = 192000 }

Functions

- std::string `AsString` (const char *inStr)
- const std::string & `AsString` (const std::string &inStr)
- const std::string & `AsString` (const `Exception::Any` &inStr)
- bool `IsNoteOn` (const `AAX_CMidiPacket` *inPacket)

Returns true if inPacket is a Note On message.

- bool `IsNoteOff` (const `AAX_CMidiPacket` *inPacket)

- **bool IsAllNotesOff (const AAX_CMidiPacket *inPacket)**

Returns true if `inPacket` is a Note Off message, or a Note On message with velocity zero.
- **bool IsAccentedClick (const AAX_CMidiPacket *inPacket)**

Returns true if `inPacket` is an All Sound Off or All Notes Off message.
- **bool IsUnaccentedClick (const AAX_CMidiPacket *inPacket)**

Returns true if `inPacket` is a special Pro Tools accented click message.
- **bool IsClick (const AAX_CMidiPacket *inPacket)**

Returns true if `inPacket` is a special Pro Tools unaccented click message.
- **template<class T1 , class T2 > bool PageTableParameterMappingsAreEqual (const T1 &inL, const T2 &inR)**
- **template<class T1 , class T2 > bool PageTableParameterNameVariationsAreEqual (const T1 &inL, const T2 &inR)**
- **template<class T1 , class T2 > bool PageTablesAreEqual (const T1 &inL, const T2 &inR)**
- **template<class T > void CopyPageTable (T &to, const T &from)**
- **template<class T > std::vector< std::pair< int32_t, int32_t > > FindParameterMappingsInPageTable (const T &inTable, AAX_CParamID inParameterID)**
- **template<class T > void ClearMappedParameterByID (T &ioTable, AAX_CParamID inParameterID)**
- **void GetCStringOfLength (char *stringOut, const char *stringIn, int32_t aMaxChars)**

=====
- **int32_t Caseless_strcmp (const char *cs, const char *ct)**
- **std::string Binary2String (uint32_t binaryValue, int32_t numBits)**
- **uint32_t String2Binary (const AAX_IString &s)**
- **bool IsASCII (char inChar)**
- **bool IsFourCharASCII (uint32_t inFourChar)**
- **std::string AsStringFourChar (uint32_t inFourChar)**
- **std::string AsStringPropertyValue (AAX_EProperty inProperty, AAX_CPropertyValue inPropertyValue)**
- **std::string AsStringInt32 (int32_t inInt32)**
- **std::string AsStringUInt32 (uint32_t inUInt32)**
- **std::string AsStringIDTriad (const AAX_SPlugInIdentifierTriad &inIDTriad)**
- **std::string AsStringStemFormat (AAX_EStemFormat inStemFormat, bool inAbbreviate=false)**
- **std::string AsStringStemChannel (AAX_EStemFormat inStemFormat, uint32_t inChannelIndex, bool inAbbreviate)**
- **std::string AsStringResult (AAX_Result inResult)**
- **double SafeLog (double aValue)**

Double-precision safe log function. Returns zero for input values that are <= 0.0.
- **float SafeLogf (float aValue)**

Single-precision safe log function. Returns zero for input values that are <= 0.0.
- **AAX_CBoolean IsParameterIDEqual (AAX_CParamID iParam1, AAX_CParamID iParam2)**

Helper function to check if two parameter IDs are equivalent.
- **AAX_CBoolean IsEffectIDEqual (const AAX_IString *iEffectID1, const AAX_IString *iEffectID2)**

Helper function to check if two Effect IDs are equivalent.
- **AAX_CBoolean IsAvidNotification (AAX_CTypeID inNotificationID)**

Helper function to check if a notification ID is reserved for host notifications.
- **void alignFree (void *p)**
- **template<class T > T * alignMalloc (int iArraySize, int iAlignment)**
- **void DeDenormal (double &iValue)**

Clamps very small floating point values to zero.
- **void DeDenormal (float &iValue)**

Clamps very small floating point values to zero.
- **void DeDenormalFine (float &iValue)**
- **void FilterDenormals (float *inSamples, int32_t inLength)**

Round all denormal/subnormal samples in a buffer to zero.
- **template<class GFLOAT > GFLOAT ClampToZero (GFLOAT iValue, GFLOAT iClampThreshold)**
- **void ZeroMemory (void *iPointer, int iNumBytes)**

- void **ZeroMemoryDW** (void *iPointer, int iNumBytes)
- template<typename T , int N> void **Fill** (T *iArray, const T *iVal)
- template<typename T , int M, int N> void **Fill** (T *iArray, const T *iVal)
- template<typename T , int L, int M, int N> void **Fill** (T *iArray, const T *iVal)
- double **fabs** (double iVal)
- float **fabs** (float iVal)
- float **fabsf** (float iVal)
- template<class T > T **AbsMax** (const T &iValue, const T &iMax)
- template<class T > T **MinMax** (const T &iValue, const T &iMin, const T &iMax)
- template<class T > T **Max** (const T &iValue1, const T &iValue2)
- template<class T > T **Min** (const T &iValue1, const T &iValue2)
- template<class T > T **Sign** (const T &iValue)
- double **PolyEval** (double x, const double *coefs, int numCoefs)
- double **CeilLog2** (double iValue)
- void **SinCosMix** (float aLinearMix, float &aSinMix, float &aCosMix)
- int32_t **FastRound2Int32** (double iVal)

Round to Int32.
- int32_t **FastRound2Int32** (float iVal)

Round to Int32.
- int32_t **FastRndDbl2Int32** (double iVal)
- int32_t **FastTrunc2Int32** (double iVal)

Float to Int conversion with truncation.
- int32_t **FastTrunc2Int32** (float iVal)

Float to Int conversion with truncation.
- int64_t **FastRound2Int64** (double iVal)

Round to Int64.
- int32_t **GetInt32RPDF** (int32_t *iSeed)

CALL: Calculate pseudo-random 32 bit number based on linear congruential method.
- float **GetRPDFWithAmplitudeOneHalf** (int32_t *iSeed)
- float **GetRPDFWithAmplitudeOne** (int32_t *iSeed)
- float **GetFastRPDFWithAmplitudeOne** (int32_t *iSeed)
- float **GetTPDFWithAmplitudeOne** (int32_t *iSeed)

MIDI logging utilities

- void **AsStringMIDIStream_Debug** (const **AAX_CMidiStream** &inStream, char *outBuffer, int32_t inBufferSize)

Filesystem utilities

- bool **GetPathToPlugInBundle** (const char *iBundleName, int iMaxLength, char *oModuleName)

Retrieve the file path of the .aaxplugin bundle.

Variables

- const int **cBigEndian** =0
- const int **cLittleEndian** =1
- const double **cPi** = 3.1415926535897932384626433832795
- const double **cTwoPi** = 6.2831853071795862319959269370884
- const double **cHalfPi** = 1.5707963267948965579989817342721
- const double **cQuarterPi** = 0.78539816339744827899949086713605
- const double **cRootTwo** = 1.4142135623730950488016887242097
- const double **cOneOverRootTwo** = 0.70710678118654752440084436210485

- const double `cPos3dB` =1.4142135623730950488016887242097
- const double `cNeg3dB` =0.70710678118654752440084436210485
- const double `cPos6dB` =2.0
- const double `cNeg6dB` =0.5
- const double `cNormalizeLongToAmplitudeOneHalf` = 0.00000000023283064365386962890625
- const double `cNormalizeLongToAmplitudeOne` = 1.0/double(1<<31)
- const double `cMilli` =0.001
- const double `cMicro` =0.001*0.001
- const double `cNano` =0.001*0.001*0.001
- const double `cPico` =0.001*0.001*0.001*0.001
- const double `cKilo` =1000.0
- const double `cMega` =1000.0*1000.0
- const double `cGiga` =1000.0*1000.0*1000.0
- const double `cDenormalAvoidanceOffset` =3.0e-34
- const float `cFloatDenormalAvoidanceOffset` =3.0e-20f
- const unsigned int `kPowExtent` = 9
- const unsigned int `kPowTableSize` = 1 << `kPowExtent`
- const float `cSeedDivisor` = 1/127773.0f
- const int32_t `cInitialSeedValue` =0x00F54321

13.1.1 Enumeration Type Documentation

13.1.1.1 enum AAX::EStatusNibble

Values for the status nibble in a MIDI packet.

Enumerator

- `eStatusNibble_NoteOff`
- `eStatusNibble_NoteOn`
- `eStatusNibble_KeyPressure`
- `eStatusNibble_ControlChange`
- `eStatusNibble_ChannelMode`
- `eStatusNibble_ProgramChange`
- `eStatusNibble_ChannelPressure`
- `eStatusNibble_PitchBend`
- `eStatusNibble_SystemCommon`
- `eStatusNibble_SystemRealTime`

13.1.1.2 enum AAX::EStatusByte

Values for the status byte in a MIDI packet.

Enumerator

- `eStatusByte_SysExBegin`
- `eStatusByte_MTCQuarterFrame`
- `eStatusByte_SongPosition`
- `eStatusByte_SongSelect`
- `eStatusByte_TuneRequest`
- `eStatusByte_SysExEnd`

eStatusByte_TimingClock
eStatusByte_Start
eStatusByte_Continue
eStatusByte_Stop
eStatusByte_ActiveSensing
eStatusByte_Reset

13.1.1.3 enum AAX::EChannelModeData

Values for the first data byte in a Channel Mode Message MIDI packet.

Enumerator

eChannelModeData_AllSoundOff
eChannelModeData_ResetControllers
eChannelModeData_LocalControl
eChannelModeData_AllNotesOff
eChannelModeData_OmniOff
eChannelModeData_OmniOn
eChannelModeData_PolyOff
eChannelModeData_PolyOn

13.1.1.4 enum AAX::ESpecialData

Special message data for the first data byte in a message.

Enumerator

eSpecialData_AccentedClick For use when the high status nibble is [eStatusNibble_NoteOn](#) and the low status nibble is zero.
eSpecialData_UnaccentedClick For use when the high status nibble is [eStatusNibble_NoteOn](#) and the low status nibble is zero.

13.1.1.5 enum AAX::ESampleRates

Enumerator

e44100SampleRate
e48000SampleRate
e88200SampleRate
e96000SampleRate
e176400SampleRate
e192000SampleRate

13.1.2 Function Documentation

13.1.2.1 std::string AAX::AsString (const char * *inStr*) [inline]

Generic conversion of a string-like object to a std::string

13.1.2.2 const std::string & AAX::AsString (const std::string & inStr) [inline]

Generic conversion of a string-like object to a std::string

13.1.2.3 const std::string & AAX::AsString (const Exception::Any & inStr) [inline]

Generic conversion of a string-like object to a std::string

References AAX::Exception::Any::What().

Here is the call graph for this function:



13.1.2.4 bool AAX::IsNoteOn (const AAX_CMidiPacket * inPacket) [inline]

Returns true if `inPacket` is a Note On message.

References eStatusNibble_NoteOn, and AAX_CMidiPacket::mData.

13.1.2.5 bool AAX::IsNoteOff (const AAX_CMidiPacket * inPacket) [inline]

Returns true if `inPacket` is a Note Off message, or a Note On message with velocity zero.

References eStatusNibble_NoteOff, eStatusNibble_NoteOn, and AAX_CMidiPacket::mData.

13.1.2.6 bool AAX::IsAllNotesOff (const AAX_CMidiPacket * inPacket) [inline]

Returns true if `inPacket` is an All Sound Off or All Notes Off message.

References eChannelModeData_AllNotesOff, eChannelModeData_AllSoundOff, eChannelModeData_OmniOff, eChannelModeData_OmniOn, eChannelModeData_PolyOff, eChannelModeData_PolyOn, eStatusNibble_ChannelMode, and AAX_CMidiPacket::mData.

13.1.2.7 bool AAX::IsAccentedClick (const AAX_CMidiPacket * inPacket) [inline]

Returns true if `inPacket` is a special Pro Tools accented click message.

References eSpecialData_AccentedClick, eStatusNibble_NoteOn, and AAX_CMidiPacket::mData.

Referenced by IsClick().

Here is the caller graph for this function:



13.1.2.8 bool AAX::IsUnaccentedClick (const AAX_CMidiPacket * inPacket) [inline]

Returns true if `inPacket` is a special Pro Tools unaccented click message.

References `eSpecialData_UnaccentedClick`, `eStatusNibble_NoteOn`, and `AAX_CMidiPacket::mData`.

Referenced by `IsClick()`.

Here is the caller graph for this function:

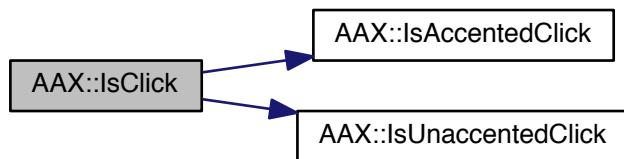


13.1.2.9 bool AAX::IsClick (const AAX_CMidiPacket * inPacket) [inline]

Returns true if `inPacket` is a special Pro Tools click message.

References `IsAccentedClick()`, and `IsUnaccentedClick()`.

Here is the call graph for this function:



13.1.2.10 template<class T1 , class T2 > bool AAX::PageTableParameterMappingsAreEqual (const T1 & *inL*, const T2 & *inR*) [inline]

Compare the parameter mappings in two page tables

T1 and T2 : Page table class types (e.g. [AAX_IACFPageTable](#), [AAX_IPageTable](#))

References AAX_SUCCESS.

Referenced by PageTablesAreEqual().

Here is the caller graph for this function:



13.1.2.11 template<class T1 , class T2 > bool AAX::PageTableParameterNameVariationsAreEqual (const T1 & *inL*, const T2 & *inR*) [inline]

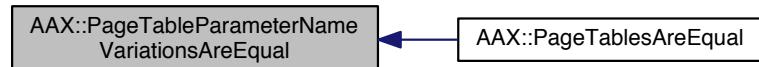
References AAX_SUCCESS, and [AAX_CString::Get\(\)](#).

Referenced by PageTablesAreEqual().

Here is the call graph for this function:



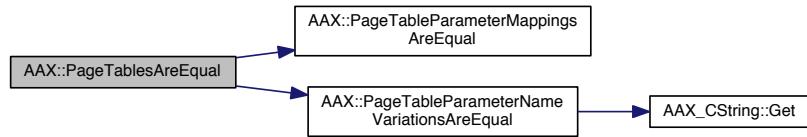
Here is the caller graph for this function:



13.1.2.12 template<class T1 , class T2 > bool AAX::PageTablesAreEqual (const T1 & *inL*, const T2 & *inR*) [inline]

References PageTableParameterMappingsAreEqual(), and PageTableParameterNameVariationsAreEqual().

Here is the call graph for this function:



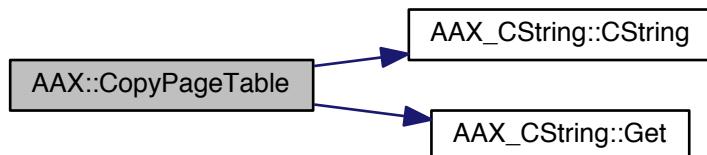
13.1.2.13 template<class T > void AAX::CopyPageTable (T & to, const T & from) [inline]

Copy a page table

T : A page table class type (e.g. [AAX_IACFPageTable](#), [AAX_IPageTable](#))

References AAX_SUCCESS, AAX_CString::CString(), and AAX_CString::Get().

Here is the call graph for this function:



13.1.2.14 template<class T > std::vector<std::pair<int32_t, int32_t> > AAX::FindParameterMappingsInPageTable (const T & inTable, AAX_CParamID inParameterID) [inline]

Find all slots where a particular parameter is mapped

T : A page table class type (e.g. [AAX_IACFPageTable](#), [AAX_IPageTable](#))

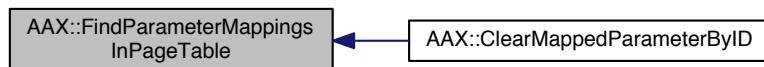
Returns

A vector of pairs of [page index, slot index] each representing a single mapping of the parameter

References AAX_SUCCESS.

Referenced by ClearMappedParameterByID().

Here is the caller graph for this function:



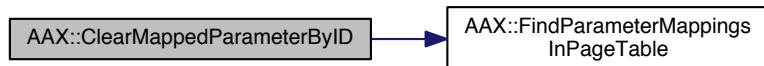
13.1.2.15 `template<class T > void AAX::ClearMappedParameterByID (T & ioTable, AAX_CParamID inParameterID) [inline]`

Remove all mappings of a particular from a page table

T : A page table class type (e.g. [AAX_IACFPageTable](#), [AAX_IPageTable](#))

References FindParameterMappingsInPageTable().

Here is the call graph for this function:



13.1.2.16 `void AAX::GetCStringOfLength (char * stringOut, const char * stringIn, int32_t aMaxChars) [inline]`

References AAX_ASSERT.

13.1.2.17 `int32_t AAX::Caseless_strcmp (const char * cs, const char * ct) [inline]`

13.1.2.18 `std::string AAX::Binary2String (uint32_t binaryValue, int32_t numBits) [inline]`

Referenced by AsStringPropertyValue().

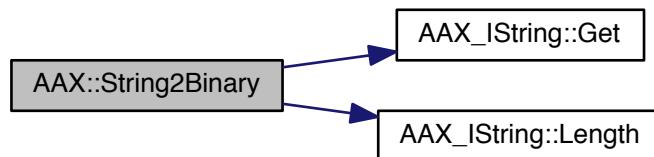
Here is the caller graph for this function:



13.1.2.19 `uint32_t AAX::String2Binary (const AAX_IString & s) [inline]`

References `AAX_ASSERT`, `AAX_IString::Get()`, and `AAX_IString::Length()`.

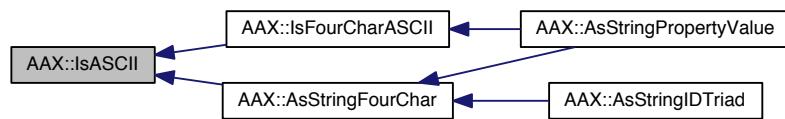
Here is the call graph for this function:



13.1.2.20 `bool AAX::IsASCII (char inChar) [inline]`

Referenced by `AsStringFourChar()`, and `IsFourCharASCII()`.

Here is the caller graph for this function:

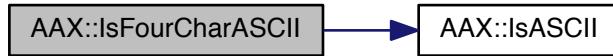


13.1.2.21 `bool AAX::IsFourCharASCII (uint32_t inFourChar) [inline]`

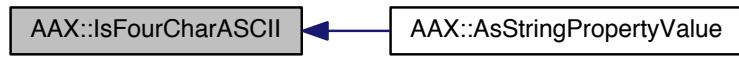
References `IsASCII()`.

Referenced by `AsStringPropertyValue()`.

Here is the call graph for this function:



Here is the caller graph for this function:



13.1.2.22 `std::string AAX::AsStringFourChar(uint32_t inFourChar) [inline]`

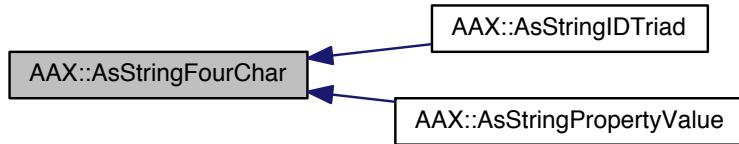
References `IsASCII()`.

Referenced by `AsStringIDTriad()`, and `AsStringPropertyValue()`.

Here is the call graph for this function:



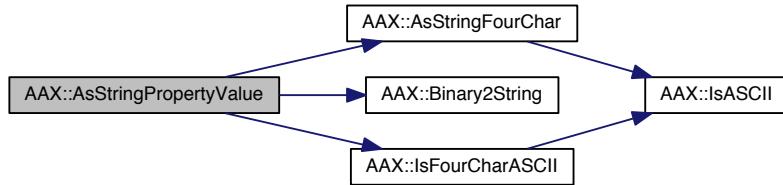
Here is the caller graph for this function:



13.1.2.23 `std::string AAX::AsStringPropertyValue (AAX_EProperty inProperty, AAX_CPropertyValue inPropertyValue) [inline]`

References `AAX_eProperty_Constraint_Location`, `AAX_eProperty_SampleRate`, `AsStringFourChar()`, `Binary2String()`, and `IsFourCharASCII()`.

Here is the call graph for this function:



13.1.2.24 `std::string AAX::AsStringInt32 (int32_t inInt32) [inline]`

Referenced by `AAX::Exception::ResultError::FormatResult()`.

Here is the caller graph for this function:



13.1.2.25 `std::string AAX::AsStringUInt32 (uint32_t inUInt32) [inline]`

13.1.2.26 `std::string AAX::AsStringIDTriad (const AAX_SPlugInIdentifierTriad & inIDTriad) [inline]`

References `AsStringFourChar()`, `AAX_SPlugInIdentifierTriad::mManufacturerID`, `AAX_SPlugInIdentifierTriad::mPlugInID`, and `AAX_SPlugInIdentifierTriad::mProductID`.

Here is the call graph for this function:



13.1.2.27 `std::string AAX::AsStringStemFormat (AAX_EStemFormat inStemFormat, bool inAbbreviate = false) [inline]`

References AAX_eStemFormat_5_0, AAX_eStemFormat_5_1, AAX_eStemFormat_6_0, AAX_eStemFormat_6_1, AAX_eStemFormat_7_0_2, AAX_eStemFormat_7_0_DTS, AAX_eStemFormat_7_0_SDDS, AAX_eStemFormat_7_1_2, AAX_eStemFormat_7_1_DTS, AAX_eStemFormat_7_1_SDDS, AAX_eStemFormat_Ambi_1_ACN, AAX_eStemFormat_Ambi_2_ACN, AAX_eStemFormat_Ambi_3_ACN, AAX_eStemFormat_Any, AAX_eStemFormat_INT32_MAX, AAX_eStemFormat_LCR, AAX_eStemFormat_LCRS, AAX_eStemFormat_Mono, AAX_eStemFormat_None, AAX_eStemFormat_Quad, AAX_eStemFormat_Reserved_1, AAX_eStemFormat_Reserved_2, AAX_eStemFormat_Reserved_3, AAX_eStemFormat_Stereo, and AAX_eStemFormatNum.

13.1.2.28 `std::string AAX::AsStringStemChannel (AAX_EStemFormat inStemFormat, uint32_t inChannelIndex, bool inAbbreviate) [inline]`

References AAX_eStemFormat_5_0, AAX_eStemFormat_5_1, AAX_eStemFormat_6_0, AAX_eStemFormat_6_1, AAX_eStemFormat_7_0_2, AAX_eStemFormat_7_0_DTS, AAX_eStemFormat_7_0_SDDS, AAX_eStemFormat_7_1_2, AAX_eStemFormat_7_1_DTS, AAX_eStemFormat_7_1_SDDS, AAX_eStemFormat_Ambi_1_ACN, AAX_eStemFormat_Ambi_2_ACN, AAX_eStemFormat_Ambi_3_ACN, AAX_eStemFormat_Any, AAX_eStemFormat_INT32_MAX, AAX_eStemFormat_LCR, AAX_eStemFormat_LCRS, AAX_eStemFormat_Mono, AAX_eStemFormat_None, AAX_eStemFormat_Quad, AAX_eStemFormat_Reserved_1, AAX_eStemFormat_Reserved_2, AAX_eStemFormat_Reserved_3, AAX_eStemFormat_Stereo, and AAX_eStemFormatNum.

13.1.2.29 `std::string AAX::AsStringResult (AAX_Result inResult) [inline]`

References AAX_ERROR_ACF_ERROR, AAX_ERROR_ARGUMENT_BUFFER_OVERFLOW, AAX_ERROR_CONTEXT_ALREADY_HAS_METERS, AAX_ERROR_DIRECT_ACCESS_OUT_OF_BOUNDS, AAX_ERROR_DUPLICATE_EFFECT_ID, AAX_ERROR_DUPLICATE_TYPE_ID, AAX_ERROR_EMPTY_EFFECT_NAME, AAX_ERROR_FIELD_TYPE_DOES_NOT_SUPPORT_DIRECT_ACCESS, AAX_ERROR_FIFO_FULL, AAX_ERROR_INCORRECT_CHUNK_SIZE, AAX_ERROR_INITIALIZING_PACKET_STREAM_THREAD, AAX_ERROR_INVALID_ARGUMENT, AAX_ERROR_INVALID_CHUNK_ID, AAX_ERROR_INVALID_CHUNK_INDEX, AAX_ERROR_INVALID_FIELD_INDEX, AAX_ERROR_INVALID_INTERNAL_DATA, AAX_ERROR_INVALID_METER_INDEX, AAX_ERROR_INVALID_METER_TYPE, AAX_ERROR_INVALID_PARAMETER_ID, AAX_ERROR_INVALID_PARAMETER_INDEX, AAX_ERROR_INVALID_PATH, AAX_ERROR_INVALID_STRING_CONVERSION, AAX_ERROR_INVALID_VIEW_SIZE, AAX_ERROR_MALFORMED_CHUNK, AAX_ERROR_MIXER_THREAD_FALLING_BEHIND, AAX_ERROR_NO_COMPONENTS, AAX_ERROR_NOT_INITIALIZED, AAX_ERROR_NOTIFICATION_FAILED, AAX_ERROR_NULL_ARGUMENT, AAX_ERROR_NULL_COMPONENT, AAX_ERROR_NULL_OBJECT, AAX_ERROR_Older_Version, AAX_ERROR_PLUGIN_NOT_AUTHORIZED, AAX_ERROR_PLUGIN_NULL_PARAMETER, AAX_ERROR_PORT_ID_OUT_OF_RANGE, AAX_ERROR_POST_PACKET_FAILED, AAX_ERROR_PROPERTY_UNDEFINED, AAX_ERROR_SIGNED_INT_OVERFLOW, AAX_ERROR_TOD_BEHIND, AAX_ERROR_UNIMPLEMENTED, AAX_ERROR_UNKNOWN_EXCEPTION, AAX_ERROR_UNKNOWN_ID, AAX_ERROR_UNKNOWN_PLUGIN, AAX_RESULT_ADD_FIELD_UNSUPPORTED_FIELD_TYPE, AAX_RESULT_NEW_PACKET_POSTED, AAX_RESULT_PACKET_STREAM_NOT_EMPTY, AAX_SUCCESS, and DEFINE_AAX_ERROR_STRING.

Referenced by AAX::Exception::ResultError::FormatResult().

Here is the caller graph for this function:

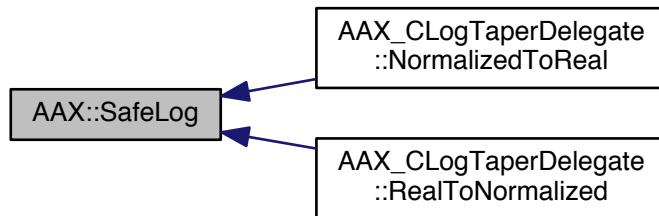


13.1.2.30 double AAX::SafeLog (double aValue) [inline]

Double-precision safe log function. Returns zero for input values that are ≤ 0.0 .

Referenced by `AAX_CLogTaperDelegate< T, RealPrecision >::NormalizedToReal()`, and `AAX_CLogTaperDelegate< T, RealPrecision >::RealToNormalized()`.

Here is the caller graph for this function:



13.1.2.31 float AAX::SafeLogf (float aValue) [inline]

Single-precision safe log function. Returns zero for input values that are ≤ 0.0 .

13.1.2.32 AAX_CBoolean AAX::IsParameterIDEqual (AAX_CParamID iParam1, AAX_CParamID iParam2) [inline]

Helper function to check if two parameter IDs are equivalent.

13.1.2.33 AAX_CBoolean AAX::IsEffectIDEqual (const AAX_IString * iEffectID1, const AAX_IString * iEffectID2) [inline]

Helper function to check if two Effect IDs are equivalent.

References `AAX_IString::Get()`.

Here is the call graph for this function:



13.1.2.34 AAX_CBoolean AAX::IsAvidNotification (AAX_CTypeID inNotificationID) [inline]

Helper function to check if a notification ID is reserved for host notifications.

13.1.2.35 void AAX::alignFree(void * *p*) [inline]

13.1.2.36 template<class T > T* AAX::alignMalloc(int *iArraySize*, int *iAlignment*)

13.1.2.37 void AAX::DeDenormal(double & *iValue*) [inline]

Clamps very small floating point values to zero.

On Pentiums and Pentium IIs the generation of denormal floats causes enormous performance losses. This routine removes denormals by clamping very small values to zero. The clamping threshold is very small, but is not the absolute minimum. If absolute minimum clamping is desired, use [AAX::DeDenormalFine\(\)](#)

13.1.2.38 void AAX::DeDenormal(float & *iValue*) [inline]

Clamps very small floating point values to zero.

On Pentiums and Pentium IIs the generation of denormal floats causes enormous performance losses. This routine removes denormals by clamping very small values to zero. The clamping threshold is very small, but is not the absolute minimum. If absolute minimum clamping is desired, use [AAX::DeDenormalFine\(\)](#)

13.1.2.39 void AAX::DeDenormalFine(float & *iValue*) [inline]

Similar to [AAX::DeDenormal\(\)](#), but uses the minimum possible normal float value as the clamping threshold

13.1.2.40 void AAX::FilterDenormals(float * *inSamples*, int32_t *inLength*) [inline]

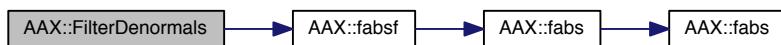
Round all denormal/subnormal samples in a buffer to zero.

Parameters

in	<i>inSamples</i>	Samples to convert
in	<i>inLength</i>	Number of samples in <i>inSamples</i>

References fabsf().

Here is the call graph for this function:



13.1.2.41 template<class GFLOAT > GFLOAT AAX::ClampToZero(GFLOAT *iValue*, GFLOAT *iClampThreshold*) [inline]

13.1.2.42 void AAX::ZeroMemory(void * *iPointer*, int *iNumBytes*) [inline]

13.1.2.43 void AAX::ZeroMemoryDW(void * *iPointer*, int *iNumBytes*) [inline]

13.1.2.44 template<typename T , int N> void AAX::Fill(T * *iArray*, const T * *iVal*)

Referenced by Fill().

Here is the caller graph for this function:



13.1.2.45 template<typename T , int M, int N> void AAX::Fill (*T * iArray, const T * iVal*) [inline]

References Fill().

Here is the call graph for this function:



13.1.2.46 template<typename T , int L, int M, int N> void AAX::Fill (*T * iArray, const T * iVal*) [inline]

References Fill().

Here is the call graph for this function:



13.1.2.47 double AAX::fabs (*double iVal*) [inline]

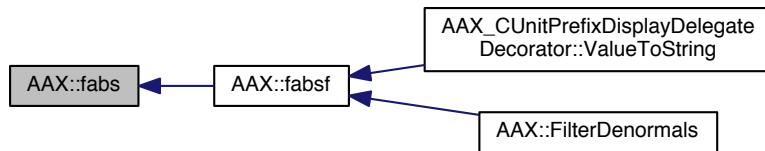
References fabs().

Referenced by fabsf().

Here is the call graph for this function:



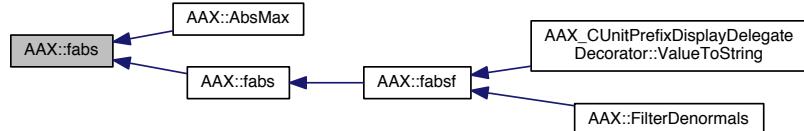
Here is the caller graph for this function:



13.1.2.48 float AAX::fabs (float *iVal*) [inline]

Referenced by AbsMax(), and fabs().

Here is the caller graph for this function:



13.1.2.49 float AAX::fabsf (float *iVal*) [inline]

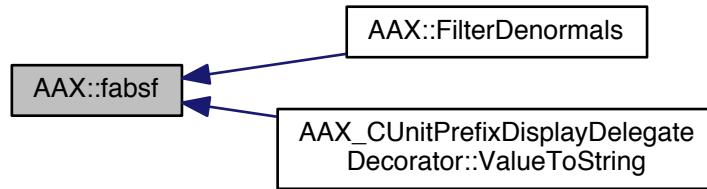
References fabs().

Referenced by FilterDenormals(), and AAX_CUnitPrefixDisplayDelegateDecorator< T >::ValueToString().

Here is the call graph for this function:



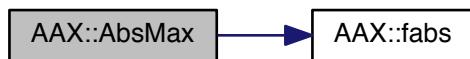
Here is the caller graph for this function:



13.1.2.50 template<class T > T AAX::AbsMax (const T & iValue, const T & iMax) [inline]

References fabs().

Here is the call graph for this function:



13.1.2.51 template<class T > T AAX::MinMax (const T & iValue, const T & iMin, const T & iMax) [inline]

13.1.2.52 template<class T > T AAX::Max (const T & iValue1, const T & iValue2) [inline]

13.1.2.53 template<class T > T AAX::Min (const T & iValue1, const T & iValue2) [inline]

13.1.2.54 template<class T > T AAX::Sign (const T & iValue) [inline]

13.1.2.55 double AAX::PolyEval (double x, const double * coefs, int numCoefs) [inline]

13.1.2.56 double AAX::CeilLog2 (double *iValue*) [inline]

13.1.2.57 void AAX::SinCosMix (float *aLinearMix*, float & *aSinMix*, float & *aCosMix*) [inline]

References cHalfPi.

13.1.2.58 int32_t AAX::FastRound2Int32 (double *iVal*) [inline]

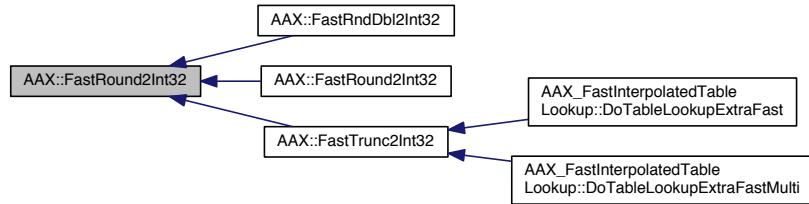
Round to Int32.

Parameters

in	<i>iVal</i>	Value to convert
----	-------------	------------------

Referenced by FastRndDbl2Int32(), FastRound2Int32(), and FastTrunc2Int32().

Here is the caller graph for this function:



13.1.2.59 int32_t AAX::FastRound2Int32 (float *iVal*) [inline]

Round to Int32.

Parameters

in	<i>iVal</i>	Value to convert
----	-------------	------------------

References FastRound2Int32().

Here is the call graph for this function:



13.1.2.60 int32_t AAX::FastRndDbl2Int32 (double *iVal*) [inline]

Deprecated

References FastRound2Int32().

Here is the call graph for this function:



13.1.2.61 int32_t AAX::FastTrunc2Int32 (double iVal) [inline]

Float to Int conversion with truncation.

Parameters

in	iVal	Value to convert
----	------	------------------

Note

This truncation is NOT identical to C style casting. Because the Intel (and I would assume PowerPC) processors use convergent rounding by default, exactly whole odd numbers will truncate down by 1.0 (e.g. 0.0->0, 1.0->0, 2.0->2, 3.0->2). Surprisingly, even with these limitations this fast float to int conversion is often very useful in practice, as long as one is aware of these issues.

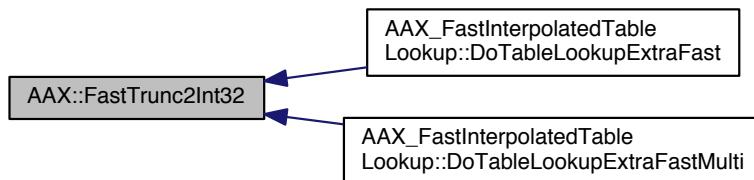
References FastRound2Int32().

Referenced by AAX_FastInterpolatedTableLookup< TFLOAT, DFLOAT >::DoTableLookupExtraFast(), and AAX_FastInterpolatedTableLookup< TFLOAT, DFLOAT >::DoTableLookupExtraFastMulti().

Here is the call graph for this function:



Here is the caller graph for this function:



13.1.2.62 int32_t AAX::FastTrunc2Int32 (float *iVal*) [inline]

Float to Int conversion with truncation.

Parameters

in	<i>iVal</i>	Value to convert
----	-------------	------------------

13.1.2.63 int64_t AAX::FastRound2Int64 (double *iVal*) [inline]

Round to Int64.

Taken from Paul V's implementation in Sys_VecUtils. This only works on values smaller than 2^{52} .

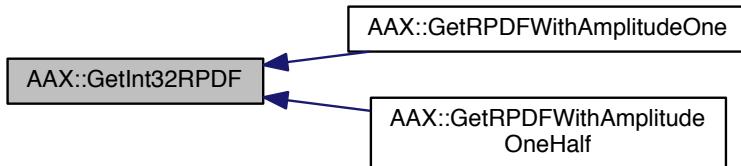
Parameters

in	<i>iVal</i>	Value to convert
----	-------------	------------------

13.1.2.64 int32_t AAX::GetInt32RPDF (int32_t * *iSeed*) [inline]

Referenced by GetRPDFWithAmplitudeOne(), and GetRPDFWithAmplitudeOneHalf().

Here is the caller graph for this function:



13.1.2.65 int32_t AAX::GetFastInt32RPDF (int32_t * *iSeed*) [inline]

CALL: Calculate pseudo-random 32 bit number based on linear congruential method.

This is required if you want our master bypass functionality in the host to hook up to your bypass parameters.

Parameters

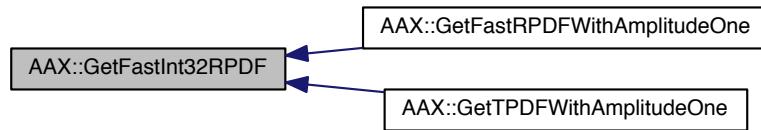
in	<i>iSeed</i>	Seed for random generator
----	--------------	---------------------------

Note

This method produces lower quality random numbers (i.e. less random) than plain old GetInt32RPDF, but in many cases it should be plenty good.

Referenced by GetFastRPDFWithAmplitudeOne(), and GetTPDFWithAmplitudeOne().

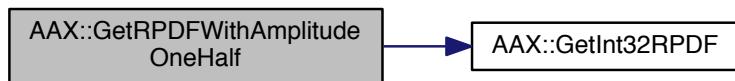
Here is the caller graph for this function:



13.1.2.66 float AAX::GetRPDFWithAmplitudeOneHalf (int32_t * iSeed) [inline]

References cNormalizeLongToAmplitudeOneHalf, and GetInt32RPDF().

Here is the call graph for this function:



13.1.2.67 float AAX::GetRPDFWithAmplitudeOne (int32_t * iSeed) [inline]

References cNormalizeLongToAmplitudeOne, and GetInt32RPDF().

Here is the call graph for this function:



13.1.2.68 float AAX::GetFastRPDFWithAmplitudeOne (int32_t * iSeed) [inline]

References cNormalizeLongToAmplitudeOne, and GetFastInt32RPDF().

Here is the call graph for this function:



13.1.2.69 float AAX::GetTPDFWithAmplitudeOne (int32_t * *iSeed*) [inline]

References cNormalizeLongToAmplitudeOne, and GetFastInt32RPDF().

Here is the call graph for this function:



13.1.3 Variable Documentation

13.1.3.1 const int AAX::cBigEndian =0

13.1.3.2 const int AAX::cLittleEndian =1

13.1.3.3 const double AAX::cPi = 3.1415926535897932384626433832795

13.1.3.4 const double AAX::cTwoPi = 6.2831853071795862319959269370884

13.1.3.5 const double AAX::cHalfPi = 1.5707963267948965579989817342721

Referenced by SinCosMix().

13.1.3.6 const double AAX::cQuarterPi = 0.78539816339744827899949086713605

13.1.3.7 const double AAX::cRootTwo = 1.4142135623730950488016887242097

13.1.3.8 const double AAX::cOneOverRootTwo = 0.70710678118654752440084436210485

13.1.3.9 const double AAX::cPos3dB = 1.4142135623730950488016887242097

13.1.3.10 const double AAX::cNeg3dB = 0.70710678118654752440084436210485

13.1.3.11 const double AAX::cPos6dB = 2.0

13.1.3.12 const double AAX::cNeg6dB = 0.5

13.1.3.13 const double AAX::cNormalizeLongToAmplitudeOneHalf = 0.00000000023283064365386962890625

Referenced by GetRPDFWithAmplitudeOneHalf().

13.1.3.14 const double AAX::cNormalizeLongToAmplitudeOne = 1.0/double(1<<31)

Referenced by GetFastRPDFWithAmplitudeOne(), GetRPDFWithAmplitudeOne(), and GetTPDFWithAmplitudeOne().

13.1.3.15 const double AAX::cMilli =0.001

13.1.3.16 const double AAX::cMicro =0.001*0.001

13.1.3.17 const double AAX::cNano =0.001*0.001*0.001

13.1.3.18 const double AAX::cPico =0.001*0.001*0.001*0.001

13.1.3.19 const double AAX::cKilo =1000.0

13.1.3.20 const double AAX::cMega =1000.0*1000.0

13.1.3.21 const double AAX::cGiga =1000.0*1000.0*1000.0

13.1.3.22 const double AAX::cDenormalAvoidanceOffset =3.0e-34

13.1.3.23 const float AAX::cFloatDenormalAvoidanceOffset =3.0e-20f

13.1.3.24 const unsigned int AAX::kPowExtent = 9

13.1.3.25 const unsigned int AAX::kPowTableSize = 1 << kPowExtent

13.1.3.26 const float AAX::cSeedDivisor = 1/127773.0f

13.1.3.27 const int32_t AAX::cInitialSeedValue =0x00F54321

13.2 AAX::Exception Namespace Reference

13.2.1 Description

AAX exception classes

All AAX exception classes inherit from [AAX::Exception::Any](#)

Classes

- class [Any](#)
- class [ResultError](#)

13.3 AAX_ChunkDataParserDefs Namespace Reference

13.3.1 Description

Constants used by ChunkDataParser class.

Variables

- const int32_t **FLOAT_TYPE** = 1
- const char **FLOAT_STRING_IDENTIFIER** [] = "f_"
- const int32_t **LONG_TYPE** = 2
- const char **LONG_STRING_IDENTIFIER** [] = "l_"
- const int32_t **DOUBLE_TYPE** = 3
- const char **DOUBLE_STRING_IDENTIFIER** [] = "d_"
- const size_t **DOUBLE_TYPE_SIZE** = 8
- const size_t **DOUBLE_TYPE_INCR** = 8
- const int32_t **SHORT_TYPE** = 4
- const char **SHORT_STRING_IDENTIFIER** [] = "s_"
- const size_t **SHORT_TYPE_SIZE** = 2
- const size_t **SHORT_TYPE_INCR** = 4
- const int32_t **STRING_TYPE** = 5
- const char **STRING_STRING_IDENTIFIER** [] = "r_"
- const size_t **MAX_STRINGDATA_LENGTH** = 255
- const size_t **DEFAULT32BIT_TYPE_SIZE** = 4
- const size_t **DEFAULT32BIT_TYPE_INCR** = 4
- const size_t **STRING_IDENTIFIER_SIZE** = 2
- const int32_t **NAME_NOT_FOUND** = -1
- const size_t **MAX_NAME_LENGTH** = 255
- const int32_t **BUILD_DATA_FAILED** = -333
- const int32_t **HEADER_SIZE** = 4
- const int32_t **VERSION_ID_1** = 0x01010101

13.3.2 Variable Documentation

- 13.3.2.1 const int32_t **AAX_ChunkDataParserDefs::FLOAT_TYPE** = 1
- 13.3.2.2 const char **AAX_ChunkDataParserDefs::FLOAT_STRING_IDENTIFIER**[] = "f_"
- 13.3.2.3 const int32_t **AAX_ChunkDataParserDefs::LONG_TYPE** = 2
- 13.3.2.4 const char **AAX_ChunkDataParserDefs::LONG_STRING_IDENTIFIER**[] = "l_"
- 13.3.2.5 const int32_t **AAX_ChunkDataParserDefs::DOUBLE_TYPE** = 3
- 13.3.2.6 const char **AAX_ChunkDataParserDefs::DOUBLE_STRING_IDENTIFIER**[] = "d_"
- 13.3.2.7 const size_t **AAX_ChunkDataParserDefs::DOUBLE_TYPE_SIZE** = 8
- 13.3.2.8 const size_t **AAX_ChunkDataParserDefs::DOUBLE_TYPE_INCR** = 8
- 13.3.2.9 const int32_t **AAX_ChunkDataParserDefs::SHORT_TYPE** = 4
- 13.3.2.10 const char **AAX_ChunkDataParserDefs::SHORT_STRING_IDENTIFIER**[] = "s_"
- 13.3.2.11 const size_t **AAX_ChunkDataParserDefs::SHORT_TYPE_SIZE** = 2
- 13.3.2.12 const size_t **AAX_ChunkDataParserDefs::SHORT_TYPE_INCR** = 4
- 13.3.2.13 const int32_t **AAX_ChunkDataParserDefs::STRING_TYPE** = 5
- 13.3.2.14 const char **AAX_ChunkDataParserDefs::STRING_STRING_IDENTIFIER**[] = "r_"

- 13.3.2.15 const size_t AAX_ChunkDataParserDefs::MAX_STRINGDATA_LENGTH = 255
- 13.3.2.16 const size_t AAX_ChunkDataParserDefs::DEFAULT32BIT_TYPE_SIZE = 4
- 13.3.2.17 const size_t AAX_ChunkDataParserDefs::DEFAULT32BIT_TYPE_INCR = 4
- 13.3.2.18 const size_t AAX_ChunkDataParserDefs::STRING_IDENTIFIER_SIZE = 2
- 13.3.2.19 const int32_t AAX_ChunkDataParserDefs::NAME_NOT_FOUND = -1
- 13.3.2.20 const size_t AAX_ChunkDataParserDefs::MAX_NAME_LENGTH = 255
- 13.3.2.21 const int32_t AAX_ChunkDataParserDefs::BUILD_DATA_FAILED = -333
- 13.3.2.22 const int32_t AAX_ChunkDataParserDefs::HEADER_SIZE = 4
- 13.3.2.23 const int32_t AAX_ChunkDataParserDefs::VERSION_ID_1 = 0x01010101

Chapter 14

Class Documentation

14.1 _acfUID Struct Reference

Public Attributes

- `uint32_t Data1`
- `uint16_t Data2`
- `uint16_t Data3`
- `uint8_t Data4 [8]`

14.1.1 Member Data Documentation

14.1.1.1 `uint32_t _acfUID::Data1`

14.1.1.2 `uint16_t _acfUID::Data2`

14.1.1.3 `uint16_t _acfUID::Data3`

14.1.1.4 `uint8_t _acfUID::Data4[8]`

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

- [AAX_ACFInterface.dox](#)

14.2 AAX_AggregateResult Class Reference

```
#include <AAX_Exception.h>
```

14.2.1 Description

RAll failure count convenience class for use with [AAX_CAPTURE\(\)](#) or [AAX_CAPTURE_MULT\(\)](#)

Pass this object as the first argument in a series of [AAX_CAPTURE\(\)](#) calls to count the number of failures that occur and to re-throw the last error if zero of the attempted calls succeed.

```
// example A: throw if all operations fail
AAX_AggregateResult agg;
AAX_CAPTURE( agg, RegisterThingA(); );
AAX_CAPTURE( agg, RegisterThingB(); );
AAX_CAPTURE( agg, RegisterThingC(); );
```

In this example, when `agg` goes out of scope it checks whether any of A, B, or C succeeded. If none succeeded then the last error that was encountered is raised via an `AAX_CheckedResult::Exception`. If at least one of the calls succeeded then any failures are swallowed and execution continues as normal. This approach can be useful in cases where you want to run every operation in a group and you only want a failure to be returned if all of the operations failed.

```
// example B: throw if any operation fails
AAX_AggregateResult agg;
AAX_CAPTURE( agg, ImportantOperationW(); );
AAX_CAPTURE( agg, ImportantOperationX(); );
AAX_CAPTURE( agg, ImportantOperationY(); );
AAX_CheckedResult err = agg.LastFailure();
```

In this example, the last error encountered by `agg` is converted to an `AAX_CheckedResult`. This will result in an `AAX_CheckedResult::Exception` even if at least one of the attempted operations succeeded. This approach can be useful in cases where you want all operations in a group to be executed before an error is raised for any failure within the group.

Public Member Functions

- `AAX_AggregateResult ()`
- `~AAX_AggregateResult ()`
- `AAX_AggregateResult & operator= (AAX_Result inResult)`
Overloaded `operator=()` for conversion from `AAX_Result`.
- `AAX_Result LastFailure () const`
- `int NumFailed () const`
- `int NumSucceeded () const`
- `int NumAttempted () const`

14.2.2 Constructor & Destructor Documentation

14.2.2.1 `AAX_AggregateResult::AAX_AggregateResult () [inline]`

14.2.2.2 `AAX_AggregateResult::~AAX_AggregateResult () [inline]`

14.2.3 Member Function Documentation

14.2.3.1 `AAX_AggregateResult& AAX_AggregateResult::operator= (AAX_Result inResult) [inline]`

Overloaded `operator=()` for conversion from `AAX_Result`.

References `AAX_SUCCESS`.

14.2.3.2 `AAX_Result AAX_AggregateResult::LastFailure () const [inline]`

14.2.3.3 `int AAX_AggregateResult::NumFailed () const [inline]`

14.2.3.4 `int AAX_AggregateResult::NumSucceeded () const [inline]`

14.2.3.5 `int AAX_AggregateResult::NumAttempted () const [inline]`

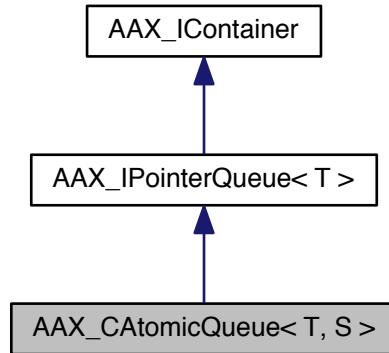
The documentation for this class was generated from the following file:

- `AAX_Exception.h`

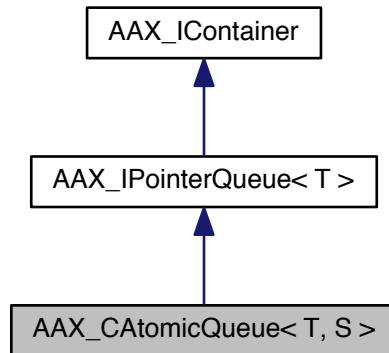
14.3 AAX_CAtomicQueue< T, S > Class Template Reference

```
#include <AAx_CAtomicQueue.h>
```

Inheritance diagram for AAX_CAtomicQueue< T, S >:



Collaboration diagram for AAX_CAtomicQueue< T, S >:



14.3.1 Description

```
template<typename T, size_t S>class AAX_CAtomicQueue< T, S >
```

Multi-writer, single-reader implementation of [AAX_IPointerQueue](#)

Template parameters:

- `T`: Type of the objects pointed to by this queue
- `S`: Size of the queue's ring buffer. Should be a power of two less than `UINT_32_MAX`

Properties:

- Read operations are non-blocking
- Write operations are synchronized, but very fast
- Supports only one read thread - do not call `Pop()` or `Peek()` concurrently
- Supports any number of write threads
- Does not support placing `NULL` values onto the queue. `Push` will return `eStatus_Unsupported` if a `NULL` value is pushed onto the queue, and the value will be ignored.

Public Types

- `typedef AAX_IPointerQueue< T >::template_type template_type`
The type used for this template instance.
- `typedef AAX_IPointerQueue< T >::value_type value_type`
The type of values stored in this queue.

Public Member Functions

- `virtual ~AAX_CAtomicQueue ()`
- `AAX_CAtomicQueue ()`
- `virtual void Clear ()`
- `virtual AAX.IContainer::EStatus Push (value_type inElem)`
- `virtual value_type Pop ()`
- `virtual value_type Peek () const`

Static Public Attributes

- `static const size_t template_size = S`
The size used for this template instance.

14.3.2 Member Typedef Documentation

14.3.2.1 `template<typename T, size_t S> typedef AAX_IPointerQueue<T>::template_type AAX_CAtomicQueue< T, S >::template_type`

The type used for this template instance.

14.3.2.2 `template<typename T, size_t S> typedef AAX_IPointerQueue<T>::value_type AAX_CAtomicQueue< T, S >::value_type`

The type of values stored in this queue.

14.3.3 Constructor & Destructor Documentation

14.3.3.1 `template<typename T, size_t S> virtual AAX_CAtomicQueue< T, S >::~AAX_CAtomicQueue () [inline], [virtual]`

14.3.3.2 `template<typename T, size_t S> AAX_CAtomicQueue< T, S >::AAX_CAtomicQueue ()`

14.3.4 Member Function Documentation

14.3.4.1 template<typename T, size_t S> virtual void **AAX_CAtomicQueue< T, S >::Clear()** [virtual]

Note

This operation is NOT atomic

This does NOT call the destructor for any pointed-to elements; it only clears the pointer values in the queue

Implements [AAX_IPointerQueue< T >](#).

14.3.4.2 template<typename T, size_t S> virtual AAX.IContainer::EStatus **AAX_CAtomicQueue< T, S >::Push(value_type inElem)** [virtual]

Push an element onto the queue

Call from: Write thread

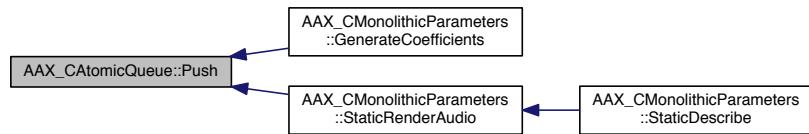
Returns

[AAX.IContainer::eStatus_Success](#) if the push succeeded

Implements [AAX_IPointerQueue< T >](#).

Referenced by [AAX_CMonolithicParameters::GenerateCoefficients\(\)](#), and [AAX_CMonolithicParameters::StaticRenderAudio\(\)](#).

Here is the caller graph for this function:



14.3.4.3 template<typename T, size_t S> virtual value_type **AAX_CAtomicQueue< T, S >::Pop()** [virtual]

Pop the front element from the queue

Call from: Read thread

Returns

NULL if no element is available

Implements [AAX_IPointerQueue< T >](#).

14.3.4.4 template<typename T, size_t S> virtual value_type **AAX_CAtomicQueue< T, S >::Peek() const** [virtual]

Get the current top element without popping it off of the queue

Call from: Read thread

Note

This value will change if another thread calls [Pop\(\)](#)

Implements [AAX_IPointerQueue< T >](#).

14.3.5 Member Data Documentation

14.3.5.1 template<typename T, size_t S> const size_t AAX_CAtomicQueue< T, S >::template_size = S [static]

The size used for this template instance.

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

- [AAX_CAtomicQueue.h](#)

14.4 AAX_CAutoreleasePool Class Reference

```
#include <AAX_CAutoreleasePool.h>
```

Public Member Functions

- [AAX_CAutoreleasePool \(\)](#)
- [~AAX_CAutoreleasePool \(\)](#)

14.4.1 Constructor & Destructor Documentation

14.4.1.1 AAX_CAutoreleasePool::AAX_CAutoreleasePool ()

14.4.1.2 AAX_CAutoreleasePool::~AAX_CAutoreleasePool ()

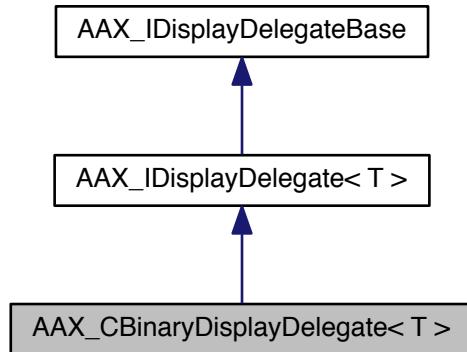
The documentation for this class was generated from the following file:

- [AAX_CAutoreleasePool.h](#)

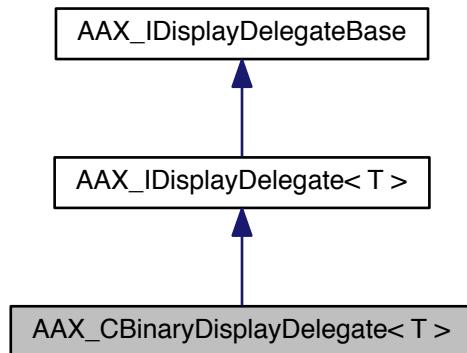
14.5 AAX_CBinaryDisplayDelegate< T > Class Template Reference

```
#include <AAX_CBinaryDisplayDelegate.h>
```

Inheritance diagram for AAX_CBinaryDisplayDelegate< T >:



Collaboration diagram for AAX_CBinaryDisplayDelegate< T >:



14.5.1 Description

`template<typename T> class AAX_CBinaryDisplayDelegate< T >`

A binary display format conforming to [AAX_IDisplayDelegate](#).

This display delegate converts a parameter value to one of two provided strings (e.g. "True" and "False".)

Public Member Functions

- [AAX_CBinaryDisplayDelegate](#) (const char *falseString, const char *trueString)
Constructor.
- [AAX_CBinaryDisplayDelegate](#) (const [AAX_CBinaryDisplayDelegate](#) &other)

- virtual `AAX_IDisplayDelegate< T > * Clone () const AAX_OVERRIDE`
Constructs and returns a copy of the display delegate.
- virtual bool `ValueToString (T value, AAX_CString *valueString) const AAX_OVERRIDE`
Converts a real parameter value to a string representation.
- virtual bool `ValueToString (T value, int32_t maxNumChars, AAX_CString *valueString) const AAX_OVERRIDE`
Converts a real parameter value to a string representation using a size hint, useful for control surfaces and other character limited displays.
- virtual bool `StringToValue (const AAX_CString &valueString, T *value) const AAX_OVERRIDE`
Converts a string to a real parameter value.
- virtual void `AddShortenedStrings (const char *falseString, const char *trueString, int iStrLength)`

14.5.2 Constructor & Destructor Documentation

14.5.2.1 `template<typename T> AAX_CBinaryDisplayDelegate< T >::AAX_CBinaryDisplayDelegate (const char * falseString, const char * trueString)`

Constructor.

Parameters

in	<code>falseString</code>	The string that will be associated with false parameter values
in	<code>trueString</code>	The string that will be associated with true parameter values

References `AAX_CString::Length()`.

Here is the call graph for this function:



14.5.2.2 `template<typename T> AAX_CBinaryDisplayDelegate< T >::AAX_CBinaryDisplayDelegate (const AAX_CBinaryDisplayDelegate< T > & other)`

14.5.3 Member Function Documentation

14.5.3.1 `template<typename T> AAX_IDisplayDelegate< T > * AAX_CBinaryDisplayDelegate< T >::Clone () const [virtual]`

Constructs and returns a copy of the display delegate.

In general, this method's implementation can use a simple copy constructor:

```

template <typename T>
AAX_CSubclassDisplayDelegate<T>* AAX_CSubclassDisplayDelegate<T>::Clone () const
{
    return new AAX_CSubclassDisplayDelegate(*this);
}

```

Implements `AAX_IDisplayDelegate< T >`.

```
14.5.3.2 template<typename T> bool AAX_CBinaryDisplayDelegate<T>::ValueToString( T value, AAX_CString *  
    valueString ) const [virtual]
```

Converts a real parameter value to a string representation.

Parameters

in	<i>value</i>	The real parameter value that will be converted
out	<i>valueString</i>	A string corresponding to value

Return values

<i>true</i>	The string conversion was successful
<i>false</i>	The string conversion was unsuccessful

Implements [AAX_IDisplayDelegate< T >](#).

14.5.3.3 template<typename T> bool **AAX_CBinaryDisplayDelegate< T >::ValueToString** (*T value*, *int32_t maxNumChars*, *AAX_CString * valueString*) const [virtual]

Converts a real parameter value to a string representation using a size hint, useful for control surfaces and other character limited displays.

Parameters

in	<i>value</i>	The real parameter value that will be converted
in	<i>maxNumChars</i>	Size hint for the desired maximum number of characters in the string (not including null termination)
out	<i>valueString</i>	A string corresponding to value

Return values

<i>true</i>	The string conversion was successful
<i>false</i>	The string conversion was unsuccessful

Implements [AAX_IDisplayDelegate< T >](#).

14.5.3.4 template<typename T> bool **AAX_CBinaryDisplayDelegate< T >::StringToValue** (*const AAX_CString & valueString*, *T * value*) const [virtual]

Converts a string to a real parameter value.

Parameters

in	<i>valueString</i>	The string that will be converted
out	<i>value</i>	The real parameter value corresponding to valueString

Return values

<i>true</i>	The string conversion was successful
<i>false</i>	The string conversion was unsuccessful

Implements [AAX_IDisplayDelegate< T >](#).

14.5.3.5 template<typename T> void **AAX_CBinaryDisplayDelegate< T >::AddShortenedStrings** (*const char * falseString*, *const char * trueString*, *int iStrLength*) [virtual]

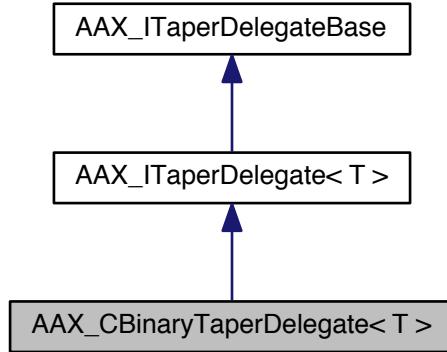
The documentation for this class was generated from the following file:

- [AAX_CBinaryDisplayDelegate.h](#)

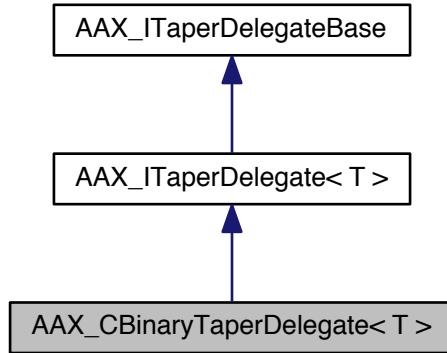
14.6 AAX_CBinaryTaperDelegate< T > Class Template Reference

```
#include <AAX_CBinaryTaperDelegate.h>
```

Inheritance diagram for AAX_CBinaryTaperDelegate< T >:



Collaboration diagram for AAX_CBinaryTaperDelegate< T >:



14.6.1 Description

```
template<typename T> class AAX_CBinaryTaperDelegate< T >
```

A binary taper conforming to [AAX_ITaperDelegate](#).

This taper maps positive real values to 1 and negative or zero real values to 0. This is the standard taper used on all bool parameters.

When this taper is constructed with a bool template type, its normalized values are automatically typecast to the proper boolean value.

Public Member Functions

- [AAX_CBinaryTaperDelegate \(\)](#)
Constructs a Binary Taper.
- virtual [AAX_ITaperDelegate< T > * Clone \(\) const AAX_OVERRIDE](#)
Constructs and returns a copy of the taper delegate.
- virtual T [GetMaximumValue \(\) const AAX_OVERRIDE](#)
Returns the taper's maximum real value.
- virtual T [GetMinimumValue \(\) const AAX_OVERRIDE](#)
Returns the taper's minimum real value.
- virtual T [ConstrainRealValue \(T value\) const AAX_OVERRIDE](#)
Applies a constraint to the value and returns the constrained value.
- virtual T [NormalizedToReal \(double normalizedValue\) const AAX_OVERRIDE](#)
Converts a normalized value to a real value.
- virtual double [RealToNormalized \(T realValue\) const AAX_OVERRIDE](#)
Normalizes a real parameter value.

14.6.2 Constructor & Destructor Documentation

14.6.2.1 template<typename T > AAX_CBinaryTaperDelegate< T >::AAX_CBinaryTaperDelegate()

Constructs a Binary Taper.

14.6.3 Member Function Documentation

14.6.3.1 template<typename T > AAX_ITaperDelegate< T > * AAX_CBinaryTaperDelegate< T >::Clone () const [virtual]

Constructs and returns a copy of the taper delegate.

In general, this method's implementation can use a simple copy constructor:

```
template <typename T>
AAX_CSubclassTaperDelegate<T>* AAX_CSubclassTaperDelegate<T>::Clone() const
{
    return new AAX_CSubclassTaperDelegate(*this);
}
```

Implements [AAX_ITaperDelegate< T >](#).

14.6.3.2 template<typename T > T AAX_CBinaryTaperDelegate< T >::GetMaximumValue () const [virtual]

Returns the taper's maximum real value.

Implements [AAX_ITaperDelegate< T >](#).

14.6.3.3 template<typename T > T AAX_CBinaryTaperDelegate< T >::GetMinimumValue () const [virtual]

Returns the taper's minimum real value.

Implements [AAX_ITaperDelegate< T >](#).

```
14.6.3.4 template<typename T > T AAX_CBinaryTaperDelegate< T >::ConstrainRealValue ( T value ) const
    [virtual]
```

Applies a constraint to the value and returns the constrained value.

This method is useful if the taper requires a constraint beyond simple minimum and maximum real value limits.

Note

This is the function that should actually enforces the constraints in `NormalizeToReal()` and `RealToNormalized()`.

Parameters

in	value	The unconstrained value
----	-------	-------------------------

Implements [AAX_ITaperDelegate< T >](#).

```
14.6.3.5 template<typename T > T AAX_CBinaryTaperDelegate< T >::NormalizedToReal ( double normalizedValue ) const
    [virtual]
```

Converts a normalized value to a real value.

This is where the actual taper algorithm is implemented.

This function should perform the exact inverse of [RealToNormalized\(\)](#), to within the roundoff precision of the individual taper implementation.

Parameters

in	normalizedValue	The normalized value that will be converted
----	-----------------	---

Implements [AAX_ITaperDelegate< T >](#).

```
14.6.3.6 template<typename T > double AAX_CBinaryTaperDelegate< T >::RealToNormalized ( T realValue ) const
    [virtual]
```

Normalizes a real parameter value.

This is where the actual taper algorithm is implemented.

This function should perform the exact inverse of [NormalizedToReal\(\)](#), to within the roundoff precision of the individual taper implementation.

Parameters

in	realValue	The real parameter value that will be normalized
----	-----------	--

Implements [AAX_ITaperDelegate< T >](#).

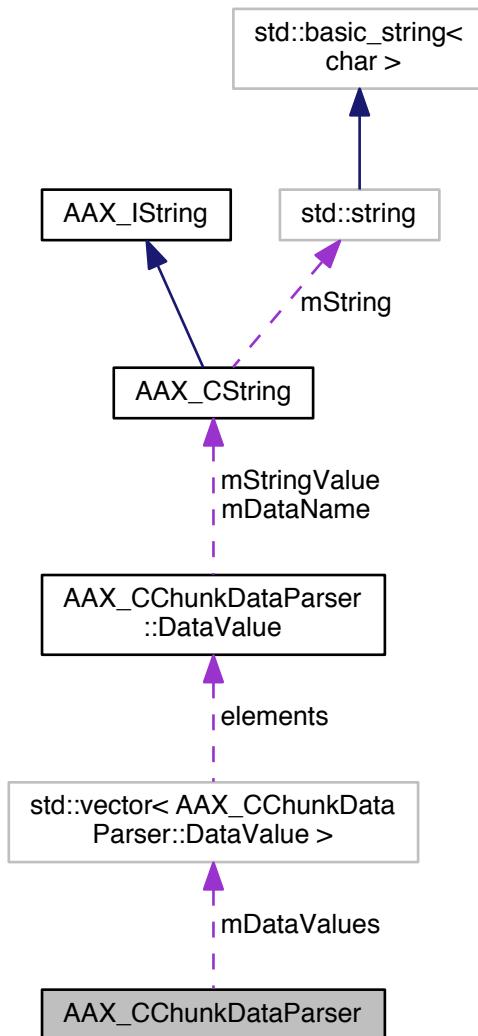
The documentation for this class was generated from the following file:

- [AAX_CBinaryTaperDelegate.h](#)

14.7 AAX_CChunkDataParser Class Reference

```
#include <AAX_CChunkDataParser.h>
```

Collaboration diagram for AAX_CChunkDataParser:



14.7.1 Description

Parser utility for plugin chunks.

Todo Update this documentation for [AAX](#)

This class acts as generic repository for data that is stuffed into or extracted from a SFicPlugInChunk. It has an abstracted Add/Find interface to add or retrieve data values, each uniquely referenced by a c-string. In conjunction with the Effect Layer and the "ControlManager" aspects of the CProcess class, this provides a more transparent & resilient system for performing save-and-restore on settings that won't break so easily from endian issues or from the hard-coded structs that have typically been used to build chunk data.

Format of the Chunk Data

The first 4 bytes of the data are the version number (repeated 4 times to be immune to byte swapping). Data

follows next.

Example: "f_bypa %\$#@d_gain #\$/#@\$\$s_omsi #\$"

```
type      name      value
-----
float    bypa      %$#@%
double   gain      #$/#@$$%
int16_t  omsi      #%
```

- The first character denotes the data type:

```
'f' = float
'd' = double
'l' = int32
's' = int16
```

- "_" is an empty placekeeper that could be used to addition future information. Currently, it's ignored when a chunk is parsed.
- The string name identifier follows next, and can up to 255 characters int32_t. The Effect Layer builds chunks it always converts the AAX_FourCharCode of the control to a string. So, this will always be 4 characters int32_t. The string is null terminated to indicate the start of the data value.
- The data value follows next, but is possible shifted to aligned word aligned. The size of is determined, of course, by the data type.

Classes

- struct [DataValue](#)

Public Member Functions

- [AAX_CChunkDataParser \(\)](#)
- virtual [~AAX_CChunkDataParser \(\)](#)
- void [AddFloat \(const char *name, float value\)](#)

CALL: Adds some data of type float with name and value to the current chunk.
- void [AddDouble \(const char *name, double value\)](#)

CALL: See [AddFloat\(\)](#)
- void [AddInt32 \(const char *name, int32_t value\)](#)

CALL: See [AddFloat\(\)](#)
- void [AddInt16 \(const char *name, int16_t value\)](#)

CALL: See [AddFloat\(\)](#)
- void [AddString \(const char *name, \[AAX_CString\]\(#\) value\)](#)
- bool [FindFloat \(const char *name, float *value\)](#)

CALL: Finds some data of type float with name and value in the current chunk.
- bool [FindDouble \(const char *name, double *value\)](#)

CALL: See [FindFloat\(\)](#)
- bool [FindInt32 \(const char *name, int32_t *value\)](#)

CALL: See [FindFloat\(\)](#)
- bool [FindInt16 \(const char *name, int16_t *value\)](#)

CALL: See [FindFloat\(\)](#)
- bool [FindString \(const char *name, \[AAX_CString\]\(#\) *value\)](#)
- bool [ReplaceDouble \(const char *name, double value\)](#)

- int32_t [GetChunkData \(AAX_SPlugInChunk *chunk\)](#)
CALL: Fills passed in chunk with data from current chunk; returns 0 if successful.
- int32_t [GetChunkContentSize \(\)](#)
CALL: Returns size of current chunk.
- int32_t [GetChunkVersion \(\)](#)
CALL: Lists fVersion in chunk header for convenience.
- bool [IsEmpty \(\)](#)
CALL: Returns true if no data is in the chunk.
- void [Clear \(\)](#)
Resets chunk.

Public Attributes

- std::vector< [DataValue](#) > mDataValues

Protected Attributes

- int32_t [mLastFoundIndex](#)
The last index found in the chunk.
- char * [mChunkData](#)
- int32_t [mChunkVersion](#)
Equal to fVersion from the chunk header. Equal to -1 if no chunk is loaded.

Internal Methods

An Effect Layer plugin can ignore these methods. They are handled by or used internally by the Effect Layer.

- void [LoadChunk \(const AAX_SPlugInChunk *chunk\)](#)
Sets current chunk to data in chunk parameter.
- void [WordAlign \(uint32_t &index\)](#)
sets index to 4-byte boundary
- void [WordAlign \(int32_t &index\)](#)
sets index to 4-byte boundary
- int32_t [FindName \(const AAX_CString &Name\)](#)
used by public Find methods

14.7.2 Constructor & Destructor Documentation

14.7.2.1 [AAX_CChunkDataParser::AAX_CChunkDataParser \(\)](#)

14.7.2.2 [virtual AAX_CChunkDataParser::~AAX_CChunkDataParser \(\) \[virtual\]](#)

14.7.3 Member Function Documentation

14.7.3.1 [void AAX_CChunkDataParser::AddFloat \(const char * name, float value \)](#)

CALL: Adds some data of type float with *name* and *value* to the current chunk.

See also

[AddDouble\(\)](#), [AddInt32\(\)](#), and [AddInt16\(\)](#) are the same but with different data types.

14.7.3.2 void AAX_CChunkDataParser::AddDouble (const char * *name*, double *value*)

CALL: See [AddFloat\(\)](#)

14.7.3.3 void AAX_CChunkDataParser::AddInt32 (const char * *name*, int32_t *value*)

CALL: See [AddFloat\(\)](#)

14.7.3.4 void AAX_CChunkDataParser::AddInt16 (const char * *name*, int16_t *value*)

CALL: See [AddFloat\(\)](#)

14.7.3.5 void AAX_CChunkDataParser::AddString (const char * *name*, AAX_CString *value*)

14.7.3.6 bool AAX_CChunkDataParser::FindFloat (const char * *name*, float * *value*)

CALL: Finds some data of type float with *name* and *value* in the current chunk.

See also

[FindDouble\(\)](#), [FindInt32\(\)](#), and [FindInt16\(\)](#) are the same but with different data types.

14.7.3.7 bool AAX_CChunkDataParser::FindDouble (const char * *name*, double * *value*)

CALL: See [FindFloat\(\)](#)

14.7.3.8 bool AAX_CChunkDataParser::FindInt32 (const char * *name*, int32_t * *value*)

CALL: See [FindFloat\(\)](#)

14.7.3.9 bool AAX_CChunkDataParser::FindInt16 (const char * *name*, int16_t * *value*)

CALL: See [FindFloat\(\)](#)

14.7.3.10 bool AAX_CChunkDataParser::FindString (const char * *name*, AAX_CString * *value*)

14.7.3.11 bool AAX_CChunkDataParser::ReplaceDouble (const char * *name*, double *value*)

14.7.3.12 int32_t AAX_CChunkDataParser::GetChunkData (AAX_SPlugInChunk * *chunk*)

CALL: Fills passed in *chunk* with data from current chunk; returns 0 if successful.

14.7.3.13 int32_t AAX_CChunkDataParser::GetChunkDataSize ()

CALL: Returns size of current chunk.

14.7.3.14 int32_t AAX_CChunkDataParser::GetChunkVersion () [inline]

CALL: Lists fVersion in chunk header for convenience.

References mChunkVersion.

14.7.3.15 `bool AAX_CChunkDataParser::IsEmpty()`

CALL: Returns true if no data is in the chunk.

14.7.3.16 `void AAX_CChunkDataParser::Clear()`

Resets chunk.

14.7.3.17 `void AAX_CChunkDataParser::LoadChunk(const AAX_SPlugInChunk * chunk)`

Sets current chunk to data in *chunk* parameter.

14.7.3.18 `void AAX_CChunkDataParser::WordAlign(uint32_t & index) [protected]`

sets *index* to 4-byte boundary

14.7.3.19 `void AAX_CChunkDataParser::WordAlign(int32_t & index) [protected]`

sets *index* to 4-byte boundary

14.7.3.20 `int32_t AAX_CChunkDataParser::FindName(const AAX_CString & Name) [protected]`

used by public Find methods

14.7.4 Member Data Documentation

14.7.4.1 `int32_t AAX_CChunkDataParser::mLastFoundIndex [protected]`

The last index found in the chunk.

Since control values in chunks should tend to stay in order and in sync with the way they're checked with controls within the plug-in, we'll keep track of the value index to speed up searching.

14.7.4.2 `char* AAX_CChunkDataParser::mChunkData [protected]`

14.7.4.3 `int32_t AAX_CChunkDataParser::mChunkVersion [protected]`

Equal to fVersion from the chunk header. Equal to -1 if no chunk is loaded.

Referenced by GetChunkVersion().

14.7.4.4 `std::vector<DataValue> AAX_CChunkDataParser::mDataValues`

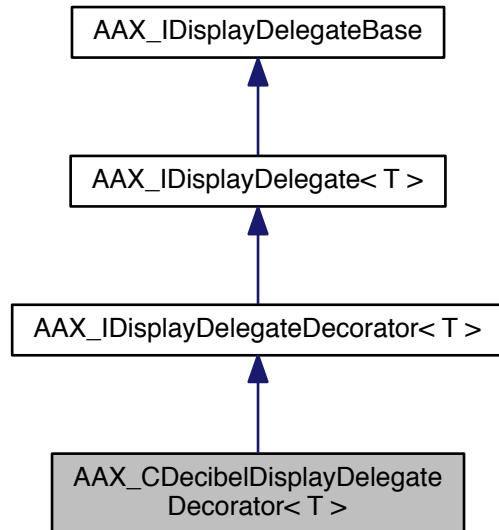
The documentation for this class was generated from the following file:

- [AAX_CChunkDataParser.h](#)

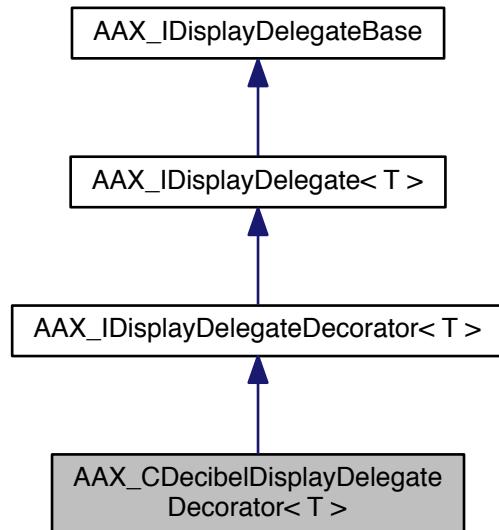
14.8 AAX_CDecibelDisplayDelegateDecorator< T > Class Template Reference

```
#include <AAX_CDecibelDisplayDelegateDecorator.h>
```

Inheritance diagram for AAX_CDecibelDisplayDelegateDecorator< T >:



Collaboration diagram for AAX_CDecibelDisplayDelegateDecorator< T >:



14.8.1 Description

```
template<typename T> class AAX_CDecibelDisplayDelegateDecorator< T >
```

A percent decorator conforming to [AAX_IDisplayDelegateDecorator](#).

This class is an [AAX_IDisplayDelegateDecorator](#), meaning that it acts as a wrapper for other display delegates or concrete display types. For more information about display delegate decorators in [AAX](#), see [Display delegate decorators](#)

The behavior of this class is to provide conversion to and from dB values. It performs a decibel conversion on the square of the provided value (i.e. 20 log) before passing the value on to a concrete display delegate to get a value string. This class then appends the "dB" suffix to signify that the value was converted. This allows something like a gain value to remain internally linear at all times even though its display is converted to decibels.

The inverse is also supported; this class can convert a decibel-formatted string into its associated real value. The string will first be converted to a number, then that number will have the inverse dB calculation applied to it to retrieve the parameter's real value.

Public Member Functions

- [AAX_CDecibelDisplayDelegateDecorator](#) (const [AAX_IDisplayDelegate](#)< T > &displayDelegate)
- virtual [AAX_CDecibelDisplayDelegateDecorator](#)< T > * [Clone](#) () const [AAX_OVERRIDE](#)
Constructs and returns a copy of the display delegate decorator.
- virtual bool [ValueToString](#) (T value, [AAX_CString](#) *valueString) const [AAX_OVERRIDE](#)
Converts a string to a real parameter value.
- virtual bool [ValueToString](#) (T value, int32_t maxNumChars, [AAX_CString](#) *valueString) const [AAX_OVERRIDE](#)
Converts a string to a real parameter value with a size constraint.
- virtual bool [StringToValue](#) (const [AAX_CString](#) &valueString, T *value) const [AAX_OVERRIDE](#)
Converts a string to a real parameter value.

14.8.2 Constructor & Destructor Documentation

14.8.2.1 template<typename T > [AAX_CDecibelDisplayDelegateDecorator](#)< T >::[AAX_CDecibelDisplayDelegateDecorator](#) (const [AAX_IDisplayDelegate](#)< T > & displayDelegate)

14.8.3 Member Function Documentation

14.8.3.1 template<typename T > [AAX_CDecibelDisplayDelegateDecorator](#)< T > * [AAX_CDecibelDisplayDelegateDecorator](#)< T >::[Clone](#) () const [virtual]

Constructs and returns a copy of the display delegate decorator.

In general, this method's implementation can use a simple copy constructor:

```
template <typename T>
AAX_CSubclassDisplayDelegate<T>*      AAX_CSubclassDisplayDelegate<T>::Clone() const
{
    return new AAX_CSubclassDisplayDelegate(*this);
}
```

Note

This is an idiomatic method in the decorator pattern, so watch for potential problems if this method is ever changed or removed.

Reimplemented from [AAX_IDisplayDelegateDecorator](#)< T >.

14.8.3.2 template<typename T> bool AAX_CDecibelDisplayDelegateDecorator< T >::ValueToString (T value, AAX_CString * valueString) const [virtual]

Converts a string to a real parameter value.

Override of the [AAX_IDisplayDelegate](#) implementation to call into the wrapped object. Display delegate decorators should call into this implementation to pass [ValueToString\(\)](#) calls on to the wrapped object after applying their own value-to-string decoration.

Parameters

in	valueString	The string that will be converted
out	value	The real parameter value corresponding to valueString

Return values

true	The string conversion was successful
false	The string conversion was unsuccessful

Reimplemented from [AAX_IDisplayDelegateDecorator< T >](#).

References [AAX_IDisplayDelegateDecorator< T >::ValueToString\(\)](#).

Here is the call graph for this function:



14.8.3.3 template<typename T> bool AAX_CDecibelDisplayDelegateDecorator< T >::ValueToString (T value, int32_t maxNumChars, AAX_CString * valueString) const [virtual]

Converts a string to a real parameter value with a size constraint.

Override of the [AAX_IDisplayDelegate](#) implementation to call into the wrapped object. Display delegate decorators should call into this implementation to pass [ValueToString\(\)](#) calls on to the wrapped object after applying their own value-to-string decoration.

Parameters

in	valueString	The string that will be converted
in	maxNumChars	Size hint for the desired maximum number of characters in the string (not including null termination)
out	value	The real parameter value corresponding to valueString

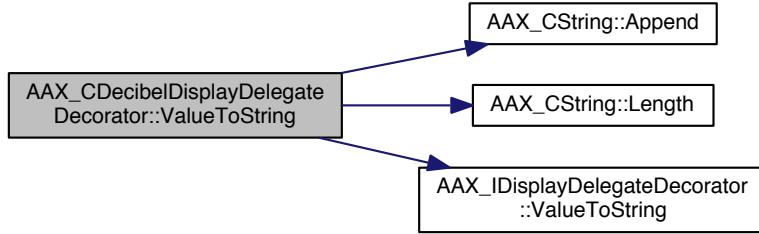
Return values

true	The string conversion was successful
false	The string conversion was unsuccessful

Reimplemented from [AAX_IDisplayDelegateDecorator< T >](#).

References [AAX_CString::Append\(\)](#), [AAX_CString::Length\(\)](#), and [AAX_IDisplayDelegateDecorator< T >::ValueToString\(\)](#).

Here is the call graph for this function:



14.8.3.4 template<typename T> bool AAX_CDecibelDisplayDelegateDecorator< T >::StringToValue (const AAX_CString & valueString, T * value) const [virtual]

Converts a string to a real parameter value.

Override of the DisplayDecorator implementation to call into the wrapped object. Display delegate decorators should call into this implementation to pass [StringToValue\(\)](#) calls on to the wrapped object after applying their own string-to-value decoding.

Parameters

in	<code>valueString</code>	The string that will be converted
out	<code>value</code>	The real parameter value corresponding to valueString

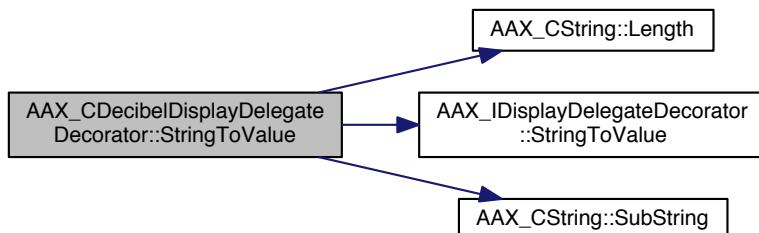
Return values

<code>true</code>	The string conversion was successful
<code>false</code>	The string conversion was unsuccessful

Reimplemented from [AAX_IDisplayDelegateDecorator< T >](#).

References [AAX_CString::Length\(\)](#), [AAX_IDisplayDelegateDecorator< T >::StringToValue\(\)](#), and [AAX_CString::SubString\(\)](#).

Here is the call graph for this function:



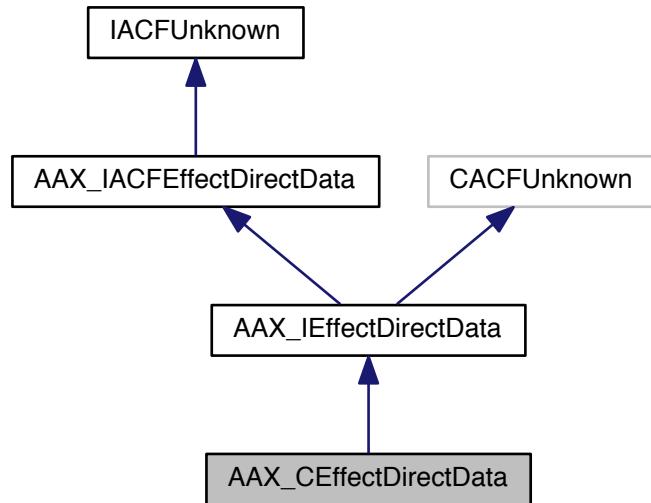
The documentation for this class was generated from the following file:

- [AAX_CDecibelDisplayDelegateDecorator.h](#)

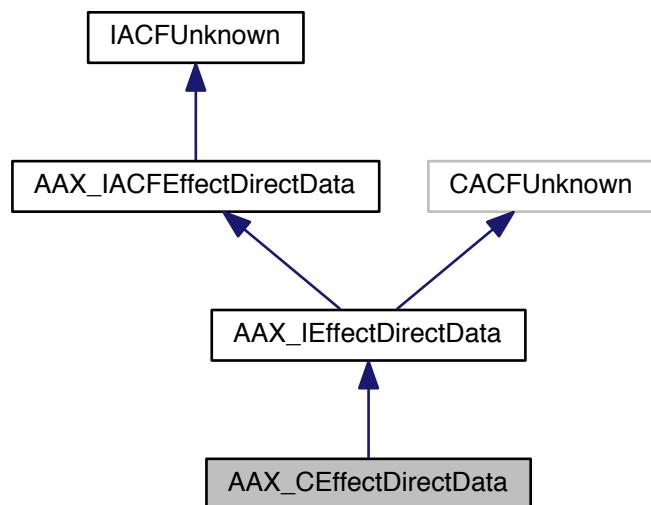
14.9 AAX_CEffectDirectData Class Reference

```
#include <AAC_EffectDirectData.h>
```

Inheritance diagram for AAX_CEffectDirectData:



Collaboration diagram for AAX_CEffectDirectData:



14.9.1 Description

Default implementation of the [AAX_IEffectDirectData](#) interface.

This class provides a default implementation of the [AAX_IEffectDirectData](#) interface.

Public Member Functions

- [AAX_CEffectDirectData](#) (void)
- virtual ~[AAX_CEffectDirectData](#) (void)

Initialization and uninitialization

- [AAX_Result Initialize \(IACFUnknown *iController\) AAX_OVERRIDE AAX_FINAL](#)
Non-virtual implementation of AAX_IEffectDirectData::Initialize()
- virtual [AAX_Result Uninitialize \(void\) AAX_OVERRIDE](#)
Main uninitialization.

Data update callbacks

- [AAX_Result TimerWakeup \(IACFUnknown *iDataAccessInterface\) AAX_OVERRIDE](#)
Non-virtual implementation of AAX_IEffectDirectData::TimerWakeup()

Private member accessors

- [AAX_IController * Controller \(void\)](#)
Returns a pointer to the plug-in's controller interface.
- [AAX_IEffectParameters * EffectParameters \(void\)](#)
Returns a pointer to the plug-in's data model interface.

Protected Member Functions

AAX_CEffectDirectData virtual interface

- virtual [AAX_Result Initialize_PrivateDataAccess \(\)](#)
Initialization routine for classes that inherit from [AAX_CEffectDirectData](#). This method is called by the default [Initialize\(\)](#) implementation after all internal members have been initialized, and provides a safe location in which to perform any additional initialization tasks.
- virtual [AAX_Result TimerWakeup_PrivateDataAccess \(AAX_IPrivateDataAccess *iPrivateDataAccess\)](#)
Callback provided with an [AAX_IPrivateDataAccess](#). Override this method to access the algorithm's private data using the [AAX_IPrivateDataAccess](#) interface.

Additional Inherited Members

14.9.2 Constructor & Destructor Documentation

14.9.2.1 [AAX_CEffectDirectData::AAX_CEffectDirectData \(void \)](#)

14.9.2.2 virtual [AAX_CEffectDirectData::~AAX_CEffectDirectData \(void \) \[virtual\]](#)

14.9.3 Member Function Documentation

14.9.3.1 [AAX_Result AAX_CEffectDirectData::Initialize \(IACFUnknown * iController \) \[virtual\]](#)

Non-virtual implementation of [AAX_IEffectDirectData::Initialize\(\)](#)

This implementation initializes all private [AAX_CEffectDirectData](#) members and calls [Initialize_PrivateDataAccess\(\)](#). For custom initialization, inherited classes should override [Initialize_PrivateDataAccess\(\)](#).

Parameters

in	<i>iController</i>	Unknown pointer that resolves to an AAX_IController .
----	--------------------	---

Implements [AAX_IACFEffectDirectData](#).

14.9.3.2 virtual **AAX_Result** AAX_CEffectDirectData::Uninitialize(void) [virtual]

Main uninitialized.

Called when the interface is destroyed.

Implements [AAX_IACFEffectDirectData](#).

14.9.3.3 **AAX_Result** AAX_CEffectDirectData::TimerWakeup(**IACFUnknown** * *iDataAccessInterface*) [virtual]

Non-virtual implementation of [AAX_IEffectDirectData::TimerWakeup\(\)](#)

This implementation interprets the [IACFUnknown](#) and forwards the resulting [AAX_IPrivateDataAccess](#) to [TimerWakeup_PrivateDataAccess\(\)](#)

Parameters

in	<i>iDataAccessInterface</i>	Unknown pointer that resolves to an AAX_IPrivateDataAccess . This interface is only valid for the duration of this method's execution and is discarded when the method returns.
----	-----------------------------	---

Implements [AAX_IACFEffectDirectData](#).

14.9.3.4 **AAX_IController*** AAX_CEffectDirectData::Controller(void)

Returns a pointer to the plug-in's controller interface.

Todo Change to GetController to match other AAX_CEffect modules

14.9.3.5 **AAX_IEffectParameters*** AAX_CEffectDirectData::EffectParameters(void)

Returns a pointer to the plug-in's data model interface.

Todo Change to GetController to match other AAX_CEffect modules

14.9.3.6 virtual **AAX_Result** AAX_CEffectDirectData::Initialize_PrivateDataAccess() [protected], [virtual]

Initialization routine for classes that inherit from [AAX_CEffectDirectData](#). This method is called by the default [Initialize\(\)](#) implementation after all internal members have been initialized, and provides a safe location in which to perform any additional initialization tasks.

14.9.3.7 virtual **AAX_Result** AAX_CEffectDirectData::TimerWakeup_PrivateDataAccess(**AAX_IPrivateDataAccess** * *iPrivateDataAccess*) [protected], [virtual]

Callback provided with an [AAX_IPrivateDataAccess](#). Override this method to access the algorithm's private data using the [AAX_IPrivateDataAccess](#) interface.

Parameters

in	iPrivateDataAccess	Pointer to an AAX_IPrivateDataAccess interface. This interface is only valid for the duration of this method.
----	--------------------	---

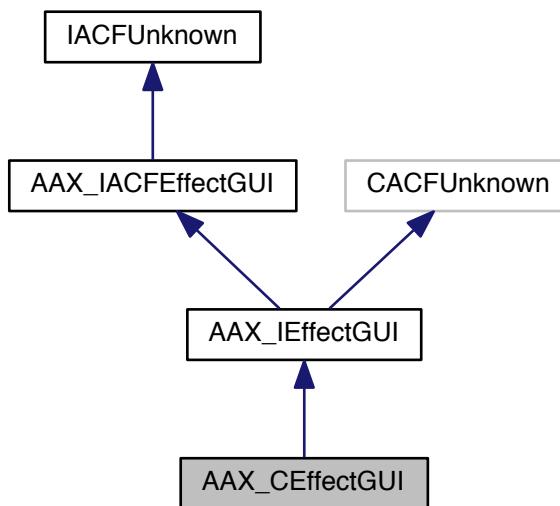
The documentation for this class was generated from the following file:

- [AAX_CEffectDirectData.h](#)

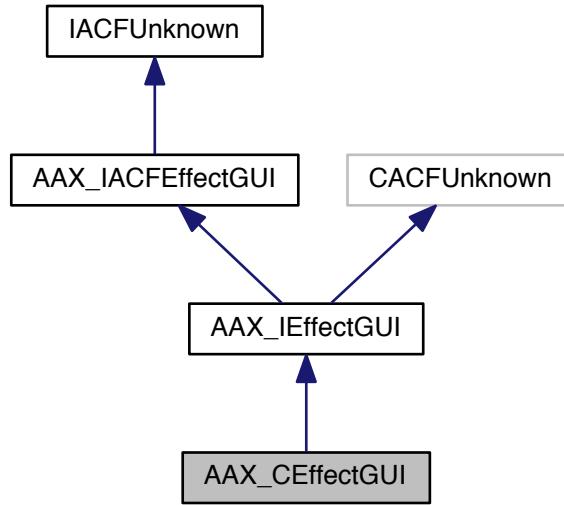
14.10 AAX_CEffectGUI Class Reference

```
#include <AAX_CEffectGUI.h>
```

Inheritance diagram for AAX_CEffectGUI:



Collaboration diagram for AAX_CEffectGUI:



14.10.1 Description

Default implementation of the [AAX_IEffectGUI](#) interface.

This class provides a default implementation of the [AAX_IEffectGUI](#) interface.

Legacy Porting Notes The default implementations in this class are mostly derived from their equivalent implementations in `CProcess` and `CEffectProcess`. For additional `CProcess`-derived implementations, see [AAX_CEffectParameters](#).

Note

See [AAX_IACFEffectGUI](#) for further information.

Public Member Functions

- [AAX_CEffectGUI \(void\)](#)
- virtual [~AAX_CEffectGUI \(void\)](#)

Initialization and uninitialization

- virtual [AAX_Result Initialize \(IACFUnknown *iController\) AAX_OVERRIDE](#)
Main GUI initialization.
- virtual [AAX_Result Uninitialize \(void\) AAX_OVERRIDE](#)
Main GUI uninitialization.

AAX host and plug-in event notification

- virtual [AAX_Result NotificationReceived \(AAX_CTypeID inNotificationType, const void *inNotificationData, uint32_t inNotificationDataSize\) AAX_OVERRIDE](#)

Notification Hook.

View accessors

- virtual [AAX_Result SetViewContainer \(IACFUnknown *iViewContainer\) AAX_OVERRIDE](#)
Provides a handle to the main plug-in window.
- virtual [AAX_Result GetViewSize \(AAX_Point *\) const AAX_OVERRIDE](#)
Retrieves the size of the plug-in window.

GUI update methods

- virtual [AAX_Result Draw \(AAX_Rect *\) AAX_OVERRIDE](#)
DEPRECATED, Not called from host any longer. Your chosen graphics framework should be directly handling draw events from the OS.
- virtual [AAX_Result TimerWakeup \(void\) AAX_OVERRIDE](#)
Periodic wakeup callback for idle-time operations.
- virtual [AAX_Result ParameterUpdated \(AAX_CParamID paramID\) AAX_OVERRIDE](#)
Notifies the GUI that a parameter value has changed.

Host interface methods

Miscellaneous methods to provide host-specific functionality

- virtual [AAX_Result GetCustomLabel \(AAX_EPlugInStrings iSelector, AAX_IString *oString\) const AAX_OVERRIDE](#)
Called by host application to retrieve a custom plug-in string.
- virtual [AAX_Result SetControlHighlightInfo \(AAX_CParamID, AAX_CBoolean, AAX_EHighlightColor\) AAX_OVERRIDE](#)
Called by host application. Indicates that a control widget should be updated with a highlight color.

Private member accessors

- [AAX_IController * GetController \(void\)](#)
Retrieves a reference to the plug-in's controller interface.
- const [AAX_IController * GetController \(void\) const](#)
- [AAX_IEffectParameters * GetEffectParameters \(void\)](#)
Retrieves a reference to the plug-in's data model interface.
- const [AAX_IEffectParameters * GetEffectParameters \(void\) const](#)
- [AAX_IViewContainer * GetViewContainer \(void\)](#)
Retrieves a reference to the plug-in's view container interface.
- const [AAX_IViewContainer * GetViewContainer \(void\) const](#)
- [AAX_ITransport * Transport \(\)](#)
Retrieves a reference to the plug-in's Transport interface.
- const [AAX_ITransport * Transport \(\) const](#)
- [AAX_EViewContainer_Type GetViewContainerType \(\)](#)
Retrieves the Container and it's type.
- void * [GetViewContainerPtr \(\)](#)

Protected Member Functions

AAX_CEffectGUI pure virtual interface

The implementations of these methods will be specific to the particular GUI framework that is being incorporated with AAX_CEffectGUI. Classes that inherit from AAX_CEffectGUI must override these methods with their own framework-specific implementations.

- virtual void [CreateViewContents \(void\)=0](#)
Creates any required top-level GUI components.
- virtual void [CreateViewContainer \(void\)=0](#)
Initializes the plug-in window and creates the main GUI view or frame.

- virtual void [DeleteViewContainer](#) (void)=0
Uninitializes the plug-in window and deletes the main GUI view or frame.

Helper methods

- virtual void [UpdateAllParameters](#) (void)
Requests an update to the GUI for every parameter view.

Additional Inherited Members

14.10.2 Constructor & Destructor Documentation

14.10.2.1 [AAX_CEffectGUI::AAX_CEffectGUI](#)(void)

14.10.2.2 virtual [AAX_CEffectGUI::~AAX_CEffectGUI](#)(void) [virtual]

14.10.3 Member Function Documentation

14.10.3.1 virtual [AAX_Result AAX_CEffectGUI::Initialize](#)([IACFUnknown](#) * *iController*) [virtual]

Main GUI initialization.

Called when the GUI is created

Parameters

in	<i>iController</i>	A versioned reference that resolves to an AAX_IController interface
----	--------------------	---

Implements [AAX_IACFEffctGUI](#).

14.10.3.2 virtual [AAX_Result AAX_CEffectGUI::Uninitialize](#)(void) [virtual]

Main GUI uninitialization.

Called when the GUI is destroyed. Frees the GUI.

Implements [AAX_IACFEffctGUI](#).

14.10.3.3 virtual [AAX_Result AAX_CEffectGUI::NotificationReceived](#)([AAX_CTypeID](#) *inNotificationType*, const void * *inNotificationData*, [uint32_t](#) *inNotificationDataSize*) [virtual]

Notification Hook.

Called from the host to deliver notifications to this object.

Look at the [AAX_ENotificationEvent](#) enumeration to see a description of events you can listen for and the data they come with.

Note

- some notifications are sent only to the plug-in GUI while other notifications are sent only to the plug-in data model. If you are not seeing an expected notification, try checking the other plug-in objects' [NotificationReceived\(\)](#) methods.

Note

- the host may dispatch notifications synchronously or asynchronously, and calls to this method may occur concurrently on multiple threads.

A plug-in may also dispatch custom notifications using [AAX_IController::SendNotification\(\)](#). Custom notifications will be posted back to the plug-in's other objects which support a [NotificationReceived\(\)](#) method (e.g. the data model).

Parameters

in	<i>inNotificationType</i>	Type of notification being received. Notifications from the host are one of AAX_ENotificationEvent
in	<i>inNotificationData</i>	Block of incoming notification data
in	<i>inNotificationDataSize</i>	Size of <i>inNotificationData</i> , in bytes

Note

The default implementation doesn't do anything at this point, but it is probably still a good idea to call into the base class [AAX_CEffectGUI::NotificationReceived\(\)](#) function in case we want to implement some default behaviors in the future.

Implements [AAX_IACFEffctGUI](#).

14.10.3.4 virtual AAX_Result AAX_CEffectGUI::SetViewContainer (IACFUnknown * *iViewContainer*) [virtual]

Provides a handle to the main plug-in window.

Parameters

in	<i>iViewContainer</i>	An AAX_IViewContainer providing a native handle to the plug-in's window
----	-----------------------	---

Implements [AAX_IACFEffctGUI](#).

14.10.3.5 virtual AAX_Result AAX_CEffectGUI::GetViewSize (AAX_Point * *oViewSize*) const [inline], [virtual]

Retrieves the size of the plug-in window.

See also

[AAX_IViewContainer::SetViewSize\(\)](#)

Parameters

out	<i>oViewSize</i>	The size of the plug-in window as a point (width, height)
-----	------------------	---

Implements [AAX_IACFEffctGUI](#).

References [AAX_SUCCESS](#).

14.10.3.6 virtual AAX_Result AAX_CEffectGUI::Draw (AAX_Rect * *iDrawRect*) [inline], [virtual]

DEPRECATED, Not called from host any longer. Your chosen graphics framework should be directly handling draw events from the OS.

Implements [AAX_IACFEffctGUI](#).

References [AAX_SUCCESS](#).

14.10.3.7 virtual AAX_Result AAX_CEffectGUI::TimerWakeup (void) [inline], [virtual]

Periodic wakeup callback for idle-time operations.

GUI animation events such as meter updates should be triggered from this method.

This method is called from the host's main thread. In general, it should be driven at approximately one call per 30 ms. However, the wakeup is not guaranteed to be called at any regular interval - for example, it could be held off by a high real-time processing load - and there is no host contract regarding maximum latency between wakeup calls.

This wakeup runs continuously and cannot be armed/disarmed by the plug-in.

Implements [AAX_IACFEfectGUI](#).

References AAX_SUCCESS.

14.10.3.8 virtual AAX_Result AAX_CEffectGUI::ParameterUpdated (*AAX_CParamID inParamID*) [virtual]

Notifies the GUI that a parameter value has changed.

This method is called by the host whenever a parameter value has been modified

This method may be called on a non-main thread

Implements [AAX_IACFEfectGUI](#).

14.10.3.9 virtual AAX_Result AAX_CEffectGUI::GetCustomLabel (*AAX_EPlugInStrings iSelector*, *AAX_IString * oString*) const [virtual]

Called by host application to retrieve a custom plug-in string.

If no string is provided then the host's default will be used.

Parameters

<i>in</i>	<i>iSelector</i>	The requested strong. One of AAX_EPlugInStrings
<i>out</i>	<i>oString</i>	The plug-in's custom value for the requested string

Implements [AAX_IACFEfectGUI](#).

14.10.3.10 virtual AAX_Result AAX_CEffectGUI::SetControlHighlightInfo (*AAX_CParamID iParameterID*, *AAX_CBoolean iIsHighlighted*, *AAX_EHighlightColor iColor*) [inline], [virtual]

Called by host application. Indicates that a control widget should be updated with a highlight color.

Todo Document this method

Legacy Porting Notes This method was re-named from `SetControlHighliteInfo()`, its name in the legacy plug-in SDK.

Parameters

<i>in</i>	<i>iParameterID</i>	ID of parameter whose widget(s) must be highlighted
<i>in</i>	<i>iIsHighlighted</i>	True if turning highlight on, false if turning it off
<i>in</i>	<i>iColor</i>	Desired highlight color. One of AAX_EHighlightColor

Implements [AAX_IACFEfectGUI](#).

References AAX_SUCCESS.

14.10.3.11 virtual void AAX_CEffectGUI::CreateViewContents (*void*) [protected], [pure virtual]

Creates any required top-level GUI components.

This method is called by default from [AAX_CEffectGUI::Initialize\(\)](#)

14.10.3.12 `virtual void AAX_CEffectGUI::CreateViewContainer(void) [protected], [pure virtual]`

Initializes the plug-in window and creates the main GUI view or frame.

This method is called by default from [AAX_CEffectGUI::SetViewContainer\(\)](#) when a valid window is present

14.10.3.13 `virtual void AAX_CEffectGUI::DeleteViewContainer(void) [protected], [pure virtual]`

Uninitializes the plug-in window and deletes the main GUI view or frame.

This method is called by default from [AAX_CEffectGUI::SetViewContainer\(\)](#) when no valid window is present. It may also be appropriate for inheriting classes to call this method from their destructors, depending on their own internal implementation.

14.10.3.14 `virtual void AAX_CEffectGUI::UpdateAllParameters(void) [protected], [virtual]`

Requests an update to the GUI for every parameter view.

By default, calls [AAX_CEffectGUI::ParameterUpdated\(\)](#) on every registered parameter.

By default, called from [AAX_CEffectGUI::SetViewContainer\(\)](#) after a new view container has been created.

Todo Rename to `UpdateAllParameterViews()` or another name that does not lead to confusion regarding what exactly this method should be doing.

14.10.3.15 `AAX_IController* AAX_CEffectGUI::GetController(void)`

Retrieves a reference to the plug-in's controller interface.

14.10.3.16 `const AAX_IController* AAX_CEffectGUI::GetController(void) const`

14.10.3.17 `AAX_IEffectParameters* AAX_CEffectGUI::GetEffectParameters(void)`

Retrieves a reference to the plug-in's data model interface.

14.10.3.18 `const AAX_IEffectParameters* AAX_CEffectGUI::GetEffectParameters(void) const`

14.10.3.19 `AAX_IViewContainer* AAX_CEffectGUI::GetViewContainer(void)`

Retrieves a reference to the plug-in's view container interface.

14.10.3.20 `const AAX_IViewContainer* AAX_CEffectGUI::GetViewContainer(void) const`

14.10.3.21 `AAX_ITransport* AAX_CEffectGUI::Transport()`

Retrieves a reference to the plug-in's Transport interface.

14.10.3.22 `const AAX_ITransport* AAX_CEffectGUI::Transport() const`

14.10.3.23 `AAX_EViewContainer_Type AAX_CEffectGUI::GetViewContainerType()`

Retrieves the Container and it's type.

14.10.3.24 void* AAX_CEffectGUI::GetViewContainerPtr ()

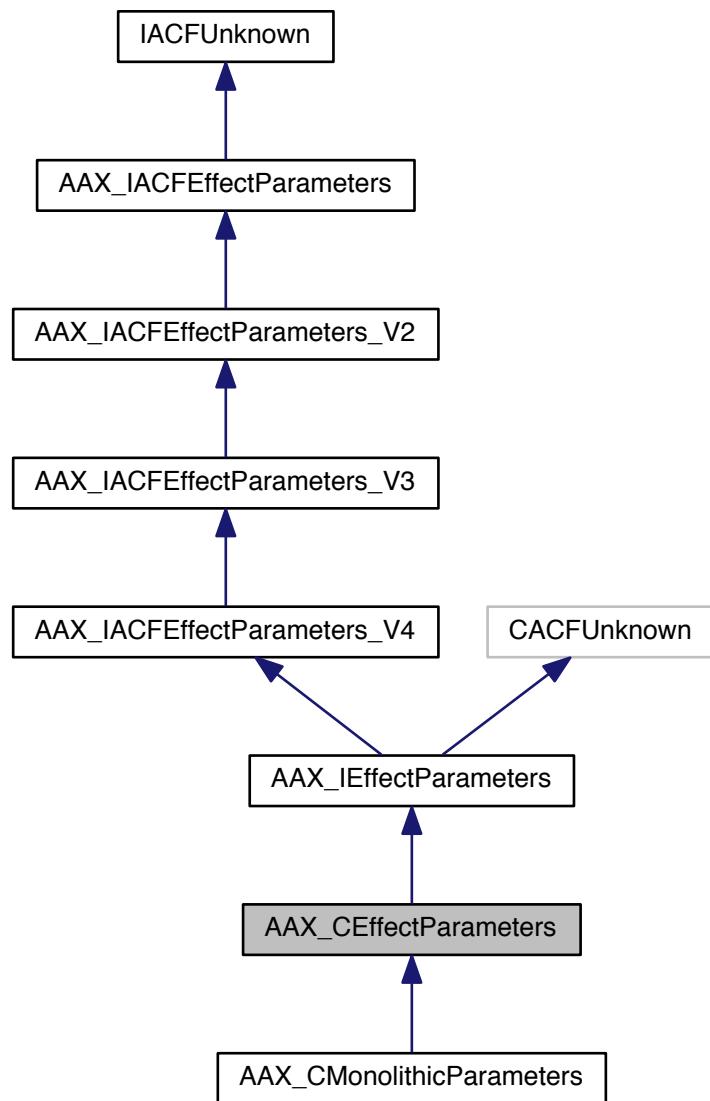
The documentation for this class was generated from the following file:

- [AAX_CEffectGUI.h](#)

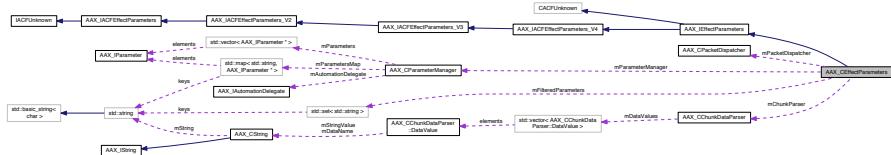
14.11 AAX_CEffectParameters Class Reference

```
#include <AAX_CEffectParameters.h>
```

Inheritance diagram for AAX_CEffectParameters:



Collaboration diagram for AAX_CEffectParameters:



14.11.1 Description

Default implementation of the [AAX_IEffectParameters](#) interface.

This class provides a default implementation of the [AAX_IEffectParameters](#) interface. In nearly all cases, your plugin's data model should inherit from this class and override only those functions that you wish to explicitly customize.

Legacy Porting Notes The default implementations in this class are mostly derived from their equivalent implementations in CProcess and CEfectProcess. For additional CProcess-derived implementations, see [AAX_CEffectGUI](#).

14.11.2 Related classes

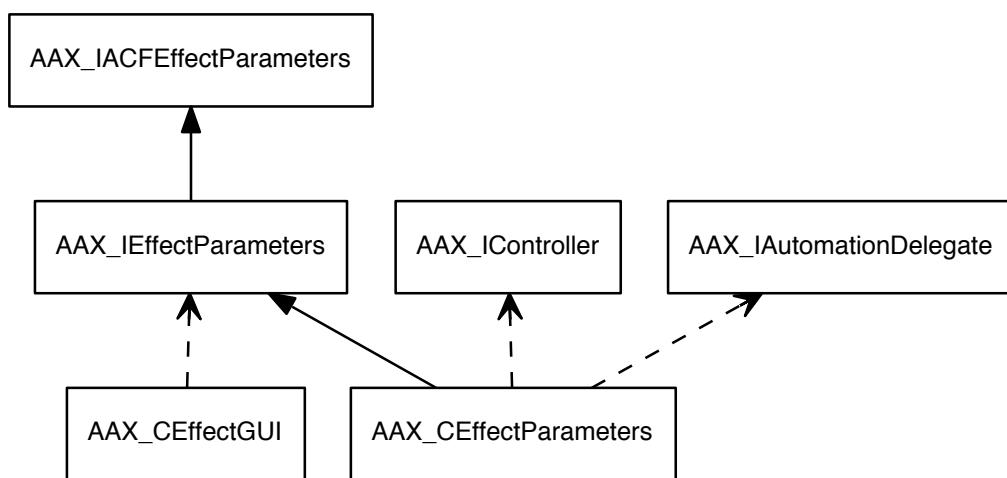


Figure 14.1: Classes related to AAX IEfectParameters by inheritance or composition

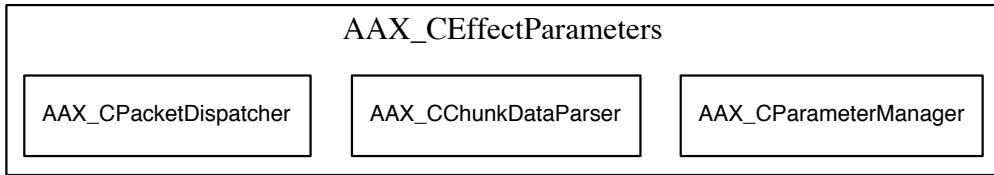


Figure 14.2: Classes owned as member objects of AAX_CEffectParameters

Public Member Functions

- [AAX_CEffectParameters \(void\)](#)
- [virtual ~AAX_CEffectParameters \(void\)](#)
- [AAX_CEffectParameters & operator= \(const AAX_CEffectParameters &other\)](#)

Initialization and uninitialized

- [virtual AAX_Result Initialize \(IACFUnknown *iController\) AAX_OVERRIDE](#)
Main data model initialization. Called when plug-in instance is first instantiated.
- [virtual AAX_Result Uninitialize \(void\) AAX_OVERRIDE](#)
Main data model uninitialization.

AAX host and plug-in event notification

- [virtual AAX_Result NotificationReceived \(AAX_CTypeID inNotificationType, const void *inNotificationData, uint32_t inNotificationDataSize\) AAX_OVERRIDE](#)
Notification Hook.

Parameter information

These methods are used by the AAX host to retrieve information about the plug-in's data model. For information about adding parameters to the plug-in and otherwise modifying the plug-in's data model, see [AAX_CParameterManager](#). For information about parameters, see [AAX_IParameter](#).

- [virtual AAX_Result GetNumberOfParameters \(int32_t *oNumControls\) const AAX_OVERRIDE](#)
CALL: Retrieves the total number of plug-in parameters.
- [virtual AAX_Result GetMasterBypassParameter \(AAX_IString *oIDString\) const AAX_OVERRIDE](#)
CALL: Retrieves the ID of the plug-in's Master Bypass parameter.
- [virtual AAX_Result GetParameterIsAutomatable \(AAX_CParamID iParameterID, AAX_CBoolean *oAutomatable\) const AAX_OVERRIDE](#)
CALL: Retrieves information about a parameter's automatable status.
- [virtual AAX_Result GetParameterNumberOfSteps \(AAX_CParamID iParameterID, int32_t *oNumSteps\) const AAX_OVERRIDE](#)
CALL: Retrieves the number of discrete steps for a parameter.
- [virtual AAX_Result GetParameterName \(AAX_CParamID iParameterID, AAX_IString *oName\) const AAX_OVERRIDE](#)
CALL: Retrieves the full name for a parameter.
- [virtual AAX_Result GetParameterNameOfLength \(AAX_CParamID iParameterID, AAX_IString *oName, int32_t iNameLength\) const AAX_OVERRIDE](#)
CALL: Retrieves an abbreviated name for a parameter.
- [virtual AAX_Result GetParameterDefaultNormalizedValue \(AAX_CParamID iParameterID, double *oValue\) const AAX_OVERRIDE](#)

- **virtual AAX_Result SetParameterDefaultNormalizedValue (AAX_CParamID iParameterID, double iValue) AAX_OVERRIDE**

CALL: Retrieves default value of a parameter.
- **virtual AAX_Result GetParameterType (AAX_CParamID iParameterID, AAX_EParameterType *oParameterType) const AAX_OVERRIDE**

CALL: Sets the default value of a parameter.
- **virtual AAX_Result GetParameterOrientation (AAX_CParamID iParameterID, AAX_EParameterOrientation *oParameterOrientation) const AAX_OVERRIDE**

CALL: Retrieves the type of a parameter.
- **virtual AAX_Result GetParameter (AAX_CParamID iParameterID, AAX_IParameter **oParameter) AA_OVERRIDE**

CALL: Retrieves the orientation that should be applied to a parameter's controls.
- **virtual AAX_Result GetParameterIndex (AAX_CParamID iParameterID, int32_t *oControlIndex) const AAX_OVERRIDE**

CALL: Retrieves an arbitrary setting within a parameter.
- **virtual AAX_Result GetParameterIDFromIndex (int32_t iControlIndex, AAX_IString *oParameterIDString) const AAX_OVERRIDE**

CALL: Retrieves the ID of a parameter.
- **virtual AAX_Result GetParameterValueInfo (AAX_CParamID iParameterID, int32_t iSelector, int32_t *oValue) const AAX_OVERRIDE**

CALL: Retrieves a property of a parameter.

Parameter setters and getters

These methods are used by the AAX host and by the plug-in's UI to retrieve and modify the values of the plug-in's parameters.

Note

The parameter setters in this section may generate asynchronous requests.

- **virtual AAX_Result GetParameterValueFromString (AAX_CParamID iParameterID, double *oValue, const AAX_IString &iValueString) const AAX_OVERRIDE**

CALL: Converts a value string to a value.
- **virtual AAX_Result GetParameterStringFromValue (AAX_CParamID iParameterID, double iValue, AAX_IString *oValueString, int32_t iMaxLength) const AAX_OVERRIDE**

CALL: Converts a normalized parameter value into a string representing its corresponding real value.
- **virtual AAX_Result GetParameterValueString (AAX_CParamID iParameterID, AAX_IString *oValueString, int32_t iMaxLength) const AAX_OVERRIDE**

CALL: Retrieves the value string associated with a parameter's current value.
- **virtual AAX_Result GetParameterNormalizedValue (AAX_CParamID iParameterID, double *oValuePtr) const AAX_OVERRIDE**

CALL: Retrieves a parameter's current value.
- **virtual AAX_Result SetParameterNormalizedValue (AAX_CParamID iParameterID, double iValue) AA_OVERRIDE**

CALL: Sets the specified parameter to a new value.
- **virtual AAX_Result SetParameterNormalizedRelative (AAX_CParamID iParameterID, double iValue) AA_OVERRIDE**

CALL: Sets the specified parameter to a new value relative to its current value.

Automated parameter helpers

These methods are used to lock and unlock automation system 'resources' when updating automatable parameters.

Note

You should never need to override these methods to extend their behavior beyond what is provided in [AAX_CEffectParameters](#) and [AAX_IParameter](#)

- virtual [AAX_Result TouchParameter \(AAX_CParamID iParameterID\) AAX_OVERRIDE](#)
"Touches" (locks) a parameter in the automation system to a particular control in preparation for updates
- virtual [AAX_Result ReleaseParameter \(AAX_CParamID iParameterID\) AAX_OVERRIDE](#)
Releases a parameter from a "touched" state.
- virtual [AAX_Result UpdateParameterTouch \(AAX_CParamID iParameterID, AAX_CBoolean iTouchState\) AAX_OVERRIDE](#)
Sets a "touched" state on a parameter.

Asynchronous parameter update methods

These methods are called by the AAX host when parameter values have been updated. They are called by the host and can be triggered by other plug-in modules via calls to [AAX_IParameter](#)'s `SetValue` methods, e.g. `SetValueWithFloat()`

These methods are responsible for updating parameter values.

Do not call these methods directly! To ensure proper synchronization and to avoid problematic dependency chains, other methods (e.g. `SetParameterNormalizedValue()`) and components (e.g. [AAX_IEffectGUI](#)) should always call a `SetValue` method on [AAX_IParameter](#) to update parameter values. The `SetValue` method will properly manage automation locks and other system resources.

- virtual [AAX_Result UpdateParameterNormalizedValue \(AAX_CParamID iParameterID, double iValue, AAX_EUpdateSource iSource\) AAX_OVERRIDE](#)
Updates a single parameter's state to its current value.
- virtual [AAX_Result UpdateParameterNormalizedRelative \(AAX_CParamID iParameterID, double iValue\) AAX_OVERRIDE](#)
Updates a single parameter's state to its current value, as a difference with the parameter's previous value.
- virtual [AAX_Result GenerateCoefficients \(void\) AAX_OVERRIDE](#)
Generates and dispatches new coefficient packets.

State reset handlers

- virtual [AAX_Result ResetFieldData \(AAX_CFieldIndex inFieldIndex, void *oData, uint32_t inDataSize\) const AAX_OVERRIDE](#)
Called by the host to reset a private data field in the plug-in's algorithm.

Chunk methods

These methods are used to save and restore collections of plug-in state information, known as chunks. Chunks are used by the host when saving or restoring presets and session settings and when providing "compare" functionality for plug-ins.

The default implementation of these methods in [AAX_CEffectParameters](#) supports a single chunk that includes state information for all of the plug-in's registered parameters. Override all of these methods to add support for additional chunks in your plug-in, for example if your plug-in contains any persistent state that is not encapsulated by its set of registered parameters.

For reference, see also:

- [AAX_CChunkDataParser](#)
- [AAX_SPlugInChunk](#)
- virtual [AAX_Result GetNumberOfChunks \(int32_t *oNumChunks\) const AAX_OVERRIDE](#)
Retrieves the number of chunks used by this plug-in.
- virtual [AAX_Result GetChunkIDFromIndex \(int32_t iIndex, AAX_CTypeID *oChunkID\) const AAX_OVERRIDE](#)
Retrieves the ID associated with a chunk index.
- virtual [AAX_Result GetChunkSize \(AAX_CTypeID iChunkID, uint32_t *oSize\) const AAX_OVERRIDE](#)

Get the size of the data structure that can hold all of a chunk's information.

- virtual [AAxResult GetChunk \(AAx_CTypeID iChunkID, AAX_SPlugInChunk *oChunk\) const AAX_OVERRIDE](#)

Fills a block of data with chunk information representing the plug-in's current state.

- virtual [AAxResult SetChunk \(AAx_CTypeID iChunkID, const AAX_SPlugInChunk *iChunk\) AAX_OVERRIDE](#)

Restores a set of plug-in parameters based on chunk information.

- virtual [AAxResult CompareActiveChunk \(const AAX_SPlugInChunk *iChunkP, AAX_CBoolean *olsEqual\) const AAX_OVERRIDE](#)

Determine if a chunk represents settings that are equivalent to the plug-in's current state.

- virtual [AAxResult GetNumberOfChanges \(int32_t *oNumChanges\) const AAX_OVERRIDE](#)

Retrieves the number of parameter changes made since the plug-in's creation.

Threads

Threading functions

- virtual [AAxResult TimerWakeUp \(\) AAX_OVERRIDE](#)

Periodic wakeup callback for idle-time operations.

Auxiliary UI methods

Methods defining the presentation of the plug-in on auxiliary UIs such as control surfaces

- virtual [AAxResult GetCurveData \(AAx_CTypeID iCurveType, const float *iValues, uint32_t iNumValues, float *oValues\) const AAX_OVERRIDE](#)

Generate a set of output values based on a set of given input values.

- virtual [AAxResult GetCurveDataMeterIds \(AAx_CTypeID iCurveType, uint32_t *oXMeterId, uint32_t *oYMeterId\) const AAX_OVERRIDE](#)

Indicates which meters correspond to the X and Y axes of the EQ or Dynamics graph.

- virtual [AAxResult GetCurveDataDisplayRange \(AAx_CTypeID iCurveType, float *oXMin, float *oXMax, float *oYMin, float *oYMax\) const AAX_OVERRIDE](#)

Determines the range of the graph shown by the plug-in.

- virtual [AAxResult UpdatePageTable \(uint32_t inTableType, int32_t inTablePageSize, IACFUnknown *iHostUnknown, IACFUnknown *ioPageTableUnknown\) const AAX_OVERRIDE AAX_FINAL](#)

Allow the plug-in to update its page tables.

Custom Data Methods

These functions exist as a proxiable way to move data between different modules (e.g. [AAxIEffectParameters](#) and [AAxIEffectGUI](#).) Using these, the GUI can query any data through [GetCustomData\(\)](#) with a plug-in defined typeID, void and size. This has an advantage over just sharing memory in that this function can work as a remote proxy as we enable those sorts of features later in the platform. Likewise, the GUI can also set arbitrary data on the data model by using the [SetCustomData\(\)](#) function with the same idea.*

Note

These are plug-in internal only. They are not called from the host right now, or likely ever.

- virtual [AAxResult GetCustomData \(AAx_CTypeID iDataBlockID, uint32_t inDataSize, void *oData, uint32_t *oDataWritten\) const AAX_OVERRIDE](#)

An optional interface hook for getting custom data from another module.

- virtual [AAxResult SetCustomData \(AAx_CTypeID iDataBlockID, uint32_t inDataSize, const void *iData\) AAX_OVERRIDE](#)

An optional interface hook for setting custom data for use by another module.

MIDI methods

- virtual [AAxResult DoMIDITransfers \(\) AAX_OVERRIDE](#)

MIDI update callback.

- virtual [AAxResult UpdateMIDINodes \(AAx_CFieldIndex inFieldIndex, AAX_CMidiPacket &iPacket\) AAX_OVERRIDE](#)

- virtual [AAX_Result UpdateControlMIDINodes](#) ([AAX_CTypeID](#) nodeID, [AAX_CMidiPacket](#) &iPacket) [AA_OVERRIDE](#)
MIDI update callback for control MIDI nodes.

Hybrid audio methods

- virtual [AAX_Result RenderAudio_Hybrid](#) ([AAX_SHybridRenderInfo](#) *ioRenderInfo) [AA_OVERRIDE](#)
Hybrid audio render function.

Private data accessors

- [AAX_IController](#) * Controller ()
Access to the Effect controller.
- const [AAX_IController](#) * Controller () const
const access to the Effect controller
- [AAX_ITransport](#) * Transport ()
Access to the Transport object.
- const [AAX_ITransport](#) * Transport () const
const access to the Transport object
- [AAX_IAutomationDelegate](#) * AutomationDelegate ()
Access to the Effect's automation delegate.
- const [AAX_IAutomationDelegate](#) * AutomationDelegate () const
const access to the Effect's automation delegate

Protected Member Functions

- void [BuildChunkData](#) (void) const
Clears out the current chunk in Chunk Parser and adds all of the new values. Used by default implementations of [GetChunk\(\)](#) and [GetChunkSize\(\)](#).

Parameter management methods

- [AAX_Result SetTaperDelegate](#) ([AAX_CParamID](#) iParameterID, [AAX_ITaperDelegateBase](#) &iTaperDelegate, bool iPreserveValue)
- [AAX_Result SetDisplayDelegate](#) ([AAX_CParamID](#) iParameterID, [AAX_IDisplayDelegateBase](#) &iDisplayDelegate)
- bool [IsParameterTouched](#) ([AAX_CParamID](#) iParameterID) const
- bool [IsParameterLinkReady](#) ([AAX_CParamID](#) inParameterID, [AAX_EUpdateSource](#) inSource) const

Convenience functions

These convenience functions provide quick access to various aspects of the default [AAX_CEffectParameters](#) implementation.

- virtual [AAX_Result EffectInit](#) (void)
Initialization helper routine. Called from [AAX_CEffectParameters::Initialize](#).
- virtual [AAX_Result UpdatePageTable](#) (uint32_t, int32_t, [AAX_IPageTable](#) &) const
- void [FilterParameterIDOnSave](#) ([AAX_CParamID](#) controlID)
CALL: Indicates the indices of parameters that should not be saved in the default [AAX_CEffectParameters](#) chunk.

Protected Attributes

- int32_t mNumPlugInChanges
- int32_t mChunkSize
- [AAX_CChunkDataParser](#) mChunkParser
- int32_t mNumChunkedParameters

- `AAX_CBoolean mClipped`
`Set by SetClipped() and returned by GetClipped().`
- `AAX_CPacketDispatcher mPacketDispatcher`
- `AAX_CParameterManager mParameterManager`
- `std::set< std::string > mFilteredParameters`

Additional Inherited Members

14.11.3 Constructor & Destructor Documentation

14.11.3.1 `AAX_CEffectParameters::AAX_CEffectParameters(void)`

14.11.3.2 `virtual AAX_CEffectParameters::~AAX_CEffectParameters(void) [virtual]`

14.11.4 Member Function Documentation

14.11.4.1 `AAX_CEffectParameters& AAX_CEffectParameters::operator=(const AAX_CEffectParameters & other)`

14.11.4.2 `virtual AAX_Result AAX_CEffectParameters::Initialize(IACFUnknown * iController) [virtual]`

Main data model initialization. Called when plug-in instance is first instantiated.

Note

Most plug-ins should override `AAX_CEffectParameters::EffectInit()` rather than directly overriding this method

Parameters

in	<code>iController</code>	A versioned reference that resolves to an <code>AAX_IController</code> interface
----	--------------------------	--

This default implementation calls `EffectInit()`. Only override `Initialize()` when additional initialization steps must be performed prior to `EffectInit()`.

Implements `AAX_IACFEFFECTPARAMETERS`.

14.11.4.3 `virtual AAX_Result AAX_CEffectParameters::Uninitialize(void) [virtual]`

Main data model uninitialization.

Todo Docs: When exactly is `AAX_IACFEFFECTPARAMETERS::Uninitialize()` called, and under what conditions?

Implements `AAX_IACFEFFECTPARAMETERS`.

14.11.4.4 `virtual AAX_Result AAX_CEffectParameters::NotificationReceived(AAX_CTypeID inNotificationType, const void * inNotificationData, uint32_t inNotificationDataSize) [virtual]`

Notification Hook.

Called from the host to deliver notifications to this object.

Look at the `AAX_ENOTIFICATIONEVENT` enumeration to see a description of events you can listen for and the data they come with.

Note

- some notifications are sent only to the plug-in GUI while other notifications are sent only to the plug-in data model. If you are not seeing an expected notification, try checking the other plug-in objects' `NotificationReceived()` methods.

Note

- the host may dispatch notifications synchronously or asynchronously, and calls to this method may occur concurrently on multiple threads.

A plug-in may also dispatch custom notifications using [AAX_IController::SendNotification\(\)](#). Custom notifications will be posted back to the plug-in's other objects which support a [NotificationReceived\(\)](#) method (e.g. the GUI).

Parameters

in	<i>inNotificationType</i>	Type of notification being received. Notifications from the host are one of AAX_ENotificationEvent
in	<i>inNotificationData</i>	Block of incoming notification data
in	<i>inNotificationDataSize</i>	Size of <i>inNotificationData</i> , in bytes

Implements [AAX_IACFEFFECTPARAMETERS](#).

14.11.4.5 virtual AAX_Result AAX_CEffectParameters::GetNumberOfParameters (int32_t * oNumControls) const [virtual]

CALL: Retrieves the total number of plug-in parameters.

Parameters

out	<i>oNumControls</i>	The number of parameters in the plug-in's Parameter Manager
-----	---------------------	---

Implements [AAX_IACFEFFECTPARAMETERS](#).

14.11.4.6 virtual AAX_Result AAX_CEffectParameters::GetMasterBypassParameter (AAX_IString * oIDString) const [virtual]

CALL: Retrieves the ID of the plug-in's Master Bypass parameter.

This is required if you want our master bypass functionality in the host to hook up to your bypass parameters.

Parameters

out	<i>oIDString</i>	The ID of the plug-in's Master Bypass control
-----	------------------	---

Implements [AAX_IACFEFFECTPARAMETERS](#).

14.11.4.7 virtual AAX_Result AAX_CEffectParameters::GetParameterIsAutomatable (AAX_CParamID iParameterID, AAX_CBoolean * oAutomatable) const [virtual]

CALL: Retrieves information about a parameter's automatable status.

Parameters

in	<i>iParameterID</i>	The ID of the parameter that is being queried
out	<i>oAutomatable</i>	True if the queried parameter is automatable, false if it is not

Implements [AAX_IACFEFFECTPARAMETERS](#).

14.11.4.8 virtual AAX_Result AAX_CEffectParameters::GetParameterNumberOfSteps (AAX_CParamID iParameterID, int32_t * oNumSteps) const [virtual]

CALL: Retrieves the number of discrete steps for a parameter.

Note

The value returned for `oNumSteps` MUST be greater than zero. All other values will be considered an error by the host.

Parameters

in	<i>iParameterID</i>	The ID of the parameter that is being queried
out	<i>oNumSteps</i>	The number of steps for this parameter

Implements [AAX_IACEffectParameters](#).

14.11.4.9 virtual AAX_Result AAX_CEffectParameters::GetParameterName (AAX_CParamID *iParameterID*,
[AAX_IString](#) * *oName*) const [virtual]

CALL: Retrieves the full name for a parameter.

Parameters

in	<i>iParameterID</i>	The ID of the parameter that is being queried
out	<i>oName</i>	Reference to an AAX_IString owned by the host. The plug-in must set this string equal to the parameter's full name.

Implements [AAX_IACEffectParameters](#).

14.11.4.10 virtual AAX_Result AAX_CEffectParameters::GetParameterNameOfLength (AAX_CParamID *iParameterID*,
[AAX_IString](#) * *oName*, int32_t *iNameLength*) const [virtual]

CALL: Retrieves an abbreviated name for a parameter.

In general, lengths of 3 through 8 and 31 should be specifically addressed.

Host Compatibility Notes In most cases, the AAX host will call [GetParameterName\(\)](#) or [GetParameterNameOfLength\(\)](#) to retrieve parameter names for display. However, when Pro Tools is retrieving a plug-in name for display on a control surface the XML data stored in the plug-in's page tables will be used in preference to values retrieved from these methods.

Parameters

in	<i>iParameterID</i>	The ID of the parameter that is being queried
out	<i>oName</i>	Reference to an AAX_IString owned by the host. The plug-in must set this string equal to an abbreviated name for the parameter, using <i>iNameLength</i> characters or fewer.
in	<i>iNameLength</i>	The maximum number of characters in <i>oName</i>

Implements [AAX_IACEffectParameters](#).

14.11.4.11 virtual AAX_Result AAX_CEffectParameters::GetParameterDefaultNormalizedValue (AAX_CParamID *iParameterID*, double * *oValue*) const [virtual]

CALL: Retrieves default value of a parameter.

Parameters

in	<i>iParameterID</i>	The ID of the parameter that is being queried
----	---------------------	---

out	<i>oValue</i>	The parameter's default value
-----	---------------	-------------------------------

Implements [AAX_IACFEffectParameters](#).

14.11.4.12 virtual AAX_Result AAX_CEffectParameters::SetParameterDefaultNormalizedValue (AAX_CParamID *iParameterID*, double *iValue*) [virtual]

CALL: Sets the default value of a parameter.

Parameters

in	<i>iParameterID</i>	The ID of the parameter that is being updated
out	<i>iValue</i>	The parameter's new default value

Todo THIS IS NOT CALLED FROM HOST. USEFUL FOR INTERNAL USE ONLY?

Implements [AAX_IACFEffectParameters](#).

14.11.4.13 virtual AAX_Result AAX_CEffectParameters::GetParameterType (AAX_CParamID *iParameterID*, AAX_EParameterType * *oParameterType*) const [virtual]

CALL: Retrieves the type of a parameter.

Todo The concept of parameter type needs more documentation

Parameters

in	<i>iParameterID</i>	The ID of the parameter that is being queried
out	<i>oParameterType</i>	The parameter's type

Implements [AAX_IACFEffectParameters](#).

14.11.4.14 virtual AAX_Result AAX_CEffectParameters::GetParameterOrientation (AAX_CParamID *iParameterID*, AAX_EParameterOrientation * *oParameterOrientation*) const [virtual]

CALL: Retrieves the orientation that should be applied to a parameter's controls.

Todo update this documentation

This method allows you to specify the orientation of knob controls that are managed by the host (e.g. knobs on an attached control surface.)

Here is an example override of this method that reverses the orientation of a control for a parameter:

```
// AAX_IParameter* myBackwardsParameter
if (iParameterID == myBackwardsParameter->Identifier())
{
    *oParameterType =
        AAX_eParameterOrientation_BottomMinTopMax |
        AAX_eParameterOrientation_LeftMinRightMax |
        AAX_eParameterOrientation_RotaryWrapMode |
        AAX_eParameterOrientation_RotaryLeftMinRightMax;
}
```

The orientation options are set according to [AAX_EParameterOrientationBits](#)

Legacy Porting Notes [AAX_IEffectParameters::GetParameterOrientation\(\)](#) corresponds to the GetControl←Orientation() method in the legacy RTAS/TDM SDK.

Parameters

in	<i>iParameterID</i>	The ID of the parameter that is being queried
out	<i>oParameter</i> \leftarrow <i>Orientation</i>	The orientation of the parameter

Implements [AAX_IACFEFFECTPARAMETERS](#).

14.11.4.15 virtual AAX_Result AAX_CEffectParameters::GetParameter (AAX_CParamID *iParameterID*, AAX_IParameter ** *oParameter*) [virtual]

CALL: Retrieves an arbitrary setting within a parameter.

This is a convenience function for accessing the richer parameter interface from the plug-in's other modules.

Note

This function must not be called by the host; [AAX_IParameter](#) is not safe for passing across the binary boundary with the host!

Parameters

in	<i>iParameterID</i>	The ID of the parameter that is being queried
out	<i>oParameter</i>	A pointer to the returned parameter

Implements [AAX_IACFEFFECTPARAMETERS](#).

14.11.4.16 virtual AAX_Result AAX_CEffectParameters::GetParameterIndex (AAX_CParamID *iParameterID*, int32_t * *oControlIndex*) const [virtual]

CALL: Retrieves the index of a parameter.

Although parameters are normally referenced by their AAX_CParamID, each parameter is also associated with a unique numeric index.

Parameters

in	<i>iParameterID</i>	The ID of the parameter that is being queried
out	<i>oControlIndex</i>	The parameter's numeric index

Implements [AAX_IACFEFFECTPARAMETERS](#).

14.11.4.17 virtual AAX_Result AAX_CEffectParameters::GetParameterIDFromIndex (int32_t *iControlIndex*, AAX_IString * *oParameterIDString*) const [virtual]

CALL: Retrieves the ID of a parameter.

This method can be used to convert a parameter's unique numeric index to its AAX_CParamID

Parameters

in	<i>iControlIndex</i>	The numeric index of the parameter that is being queried
out	<i>oParameterID</i> \leftarrow <i>String</i>	Reference to an AAX_IString owned by the host. The plug-in must set this string equal to the parameter's ID.

Implements [AAX_IACFEFFECTPARAMETERS](#).

14.11.4.18 virtual AAX_Result AAX_CEffectParameters::GetParameterValueInfo (AAX_CParamID *iParameterID*, int32_t *iSelector*, int32_t * *oValue*) const [virtual]

CALL: Retrieves a property of a parameter.

This is a general purpose query that is specialized based on the value of `iSelector`. The currently supported selector values are described by [AAX_EParameterValueInfoSelector](#). The meaning of `oValue` is dependent upon `iSelector`.

Parameters

in	<i>iParameterID</i>	The ID of the parameter that is being queried
in	<i>iSelector</i>	The selector of the parameter value to retrieve. See AAX_EParameterValueInfoSelector
out	<i>oValue</i>	The value of the specified parameter

Implements [AAX_IACFEffectParameters](#).

14.11.4.19 virtual AAX_Result AAX_CEffectParameters::GetParameterValueFromString (AAX_CParamID *iParameterID*, double * *oValue*, const AAX_IString & *iValueString*) const [virtual]

CALL: Converts a value string to a value.

This method uses the queried parameter's display delegate and taper to convert a `char*` string into its corresponding value. The formatting of `valueString` must be supported by the parameter's display delegate in order for this call to succeed.

Legacy Porting Notes This method corresponds to `CProcess::MapControlStringToVal()` in the RTAS/TDM SDK

Parameters

in	<i>iParameterID</i>	The ID of the parameter that is being queried
out	<i>oValue</i>	The value associated with <code>valueString</code>
in	<i>iValueString</i>	The formatted value string that will be converted into a value

Implements [AAX_IACFEffectParameters](#).

14.11.4.20 virtual AAX_Result AAX_CEffectParameters::GetParameterStringFromValue (AAX_CParamID *iParameterID*, double *iValue*, AAX_IString * *oValueString*, int32_t *iMaxLength*) const [virtual]

CALL: Converts a normalized parameter value into a string representing its corresponding real value.

This method uses the queried parameter's display delegate and taper to convert a normalized value into the corresponding `char*` value string for its real value.

Legacy Porting Notes This method corresponds to `CProcess::MapControlValToString()` in the RTAS/TDM SDK

Parameters

in	<i>iParameterID</i>	The ID of the parameter that is being queried
in	<i>iValue</i>	The normalized value that will be converted to a formatted <code>valueString</code>
out	<i>oValueString</i>	The formatted value string associated with <code>value</code>
in	<i>iMaxLength</i>	The maximum length of <code>valueString</code>

Implements [AAX_IACFEffectParameters](#).

14.11.4.21 virtual AAX_Result AAX_CEffectParameters::GetParameterValueString (AAX_CParamID *iParameterID*, AAX_IString * *oValueString*, int32_t *iMaxLength*) const [virtual]

CALL: Retrieves the value string associated with a parameter's current value.

This method uses the queried parameter's display delegate and taper to convert its current value into a corresponding `char*` value string.

Parameters

in	<i>iParameterID</i>	The ID of the parameter that is being queried
out	<i>oValueString</i>	The formatted value string associated with the parameter's current value
in	<i>iMaxLength</i>	The maximum length of valueString

Implements [AAX_IACEffectParameters](#).

14.11.4.22 virtual AAX_Result AAX_CEffectParameters::GetParameterNormalizedValue (AAX_CParamID *iParameterID*, double * *oValuePtr*) const [virtual]

CALL: Retrieves a parameter's current value.

Parameters

in	<i>iParameterID</i>	The ID of the parameter that is being queried
out	<i>oValuePtr</i>	The parameter's current value

Implements [AAX_IACEffectParameters](#).

14.11.4.23 virtual AAX_Result AAX_CEffectParameters::SetParameterNormalizedValue (AAX_CParamID *iParameterID*, double *iValue*) [virtual]

CALL: Sets the specified parameter to a new value.

[SetParameterNormalizedValue\(\)](#) is responsible for initiating any process that is required in order to update all of the parameter's controls (e.g. in the plug-in's GUI, on control surfaces, in automation lanes, etc.) In most cases, the parameter manager will handle this initiation step.

Parameters

in	<i>iParameterID</i>	The ID of the parameter that is being set
in	<i>iValue</i>	The value to which the parameter should be set

Implements [AAX_IACEffectParameters](#).

14.11.4.24 virtual AAX_Result AAX_CEffectParameters::SetParameterNormalizedRelative (AAX_CParamID *iParameterID*, double *iValue*) [virtual]

CALL: Sets the specified parameter to a new value relative to its current value.

This method is used in cases when a relative control value is more convenient, for example when updating a GUI control using a mouse wheel or the arrow keys. Note that the host may apply the parameter's step size prior to calling [SetParameterNormalizedRelative\(\)](#) in order to determine the correct value for aValue.

[SetParameterNormalizedRelative\(\)](#) can be used to incorporate "wrapping" behavior in a parameter's controls, if desired. If this behavior is not desired, then this method must properly account for overflow of the parameter's normalized value.

[SetParameterNormalizedRelative\(\)](#) is responsible for initiating any process that is required in order to update all of the parameter's controls (e.g. in the plug-in's GUI, on control surfaces, in automation lanes, etc.) In most cases, the parameter manager will handle this initiation step.

See also [UpdateParameterNormalizedRelative\(\)](#).

Todo REMOVE THIS METHOD (?)

Parameters

in	<i>iParameterID</i>	The ID of the parameter that is being queried
in	<i>iValue</i>	The change in value that should be applied to the parameter

Todo NOT CURRENTLY CALLED FROM THE HOST. USEFUL FOR INTERNAL USE ONLY?

Implements [AAX_IACFEffectParameters](#).

14.11.4.25 virtual AAX_Result AAX_CEffectParameters::TouchParameter (AAX_CParamID iParameterID) [virtual]

"Touches" (locks) a parameter in the automation system to a particular control in preparation for updates

This method is called by the Parameter Manager to prime a parameter for receiving new automation data. When an automatable parameter is touched by a control, it will reject input from other controls until it is released.

Note

You should never need to override this method when using [AAX_CEffectParameters](#).

Parameters

in	<i>iParameterID</i>	The parameter that is being touched
----	---------------------	-------------------------------------

Implements [AAX_IACFEffectParameters](#).

14.11.4.26 virtual AAX_Result AAX_CEffectParameters::ReleaseParameter (AAX_CParamID iParameterID) [virtual]

Releases a parameter from a "touched" state.

This method is called by the Parameter Manager to release a parameter so that any control may send updates to the parameter.

Note

You should never need to override this method when using [AAX_CEffectParameters](#).

Parameters

in	<i>iParameterID</i>	The parameter that is being released
----	---------------------	--------------------------------------

Implements [AAX_IACFEffectParameters](#).

14.11.4.27 virtual AAX_Result AAX_CEffectParameters::UpdateParameterTouch (AAX_CParamID iParameterID, AAX_CBoolean iTouchState) [virtual]

Sets a "touched" state on a parameter.

Note

This method should be overridden when dealing with linked parameters. Do NOT use this method to keep track of touch states. Use the [automation delegate](#) for that.

Parameters

in	<i>iParameterID</i>	The parameter that is changing touch states.
in	<i>iTouchState</i>	The touch state of the parameter.

Implements [AAX_IACFEffectorParameters](#).

14.11.4.28 virtual AAX_Result AAX_CEffectParameters::UpdateParameterNormalizedValue (AAX_CParamID *iParameterID*, double *iValue*, AAX_EUpdateSource *iSource*) [virtual]

Updates a single parameter's state to its current value.

Note

Do *not* call this method from the plug-in. This method should be called by the host only. To set parameter values from within the plug-in, use the [AAX_IParameter](#) interface.

Todo FLAGGED FOR CONSIDERATION OF REVISION

Parameters

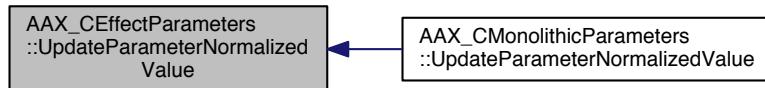
in	<i>iParameterID</i>	The ID of the parameter that is being updated
in	<i>iValue</i>	The parameter's current value, to which its internal state must be updated
in	<i>iSource</i>	The source of the update

Implements [AAX_IACFEffectorParameters](#).

Reimplemented in [AAX_CMonolithicParameters](#).

Referenced by [AAX_CMonolithicParameters::UpdateParameterNormalizedValue\(\)](#).

Here is the caller graph for this function:



14.11.4.29 virtual AAX_Result AAX_CEffectParameters::UpdateParameterNormalizedRelative (AAX_CParamID *iParameterID*, double *iValue*) [virtual]

Updates a single parameter's state to its current value, as a difference with the parameter's previous value.

Deprecated This is not called from the host. It *may* still be useful for internal calls within the plug-in, though it should only ever be used to update non-automatable parameters. Automatable parameters should always be updated through the [AAX_IParameter](#) interface, which will ensure proper coordination with other automation clients.

[UpdateParameterNormalizedRelative\(\)](#) can be used to incorporate "wraparound" behavior in a parameter's controls, if desired. If this behavior is not desired, then this method must properly account for overflow of the parameter's normalized value.

See also

[SetParameterNormalizedRelative\(\)](#)

Parameters

in	<i>iParameterID</i>	The ID of the parameter that is being updated
in	<i>iValue</i>	The difference between the parameter's current value and its previous value (normalized). The parameter's state must be updated to reflect this difference.

Implements [AAX_IACFEffectParameters](#).

14.11.4.30 virtual AAX_Result AAX_CEffectParameters::GenerateCoefficients (void) [virtual]

Generates and dispatches new coefficient packets.

This method is responsible for updating the coefficient packets associated with all parameters whose states have changed since the last call to [GenerateCoefficients\(\)](#). The host may call this method once for every parameter update, or it may "batch" parameter updates such that changes for several parameters are all handled by a single call to [GenerateCoefficients\(\)](#).

For more information on tracking parameters' statuses using the [AAX_CPacketDispatcher](#), helper class, see [AA_X_CPacketDispatcher::SetDirty\(\)](#).

Note

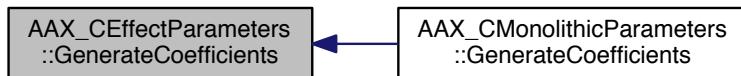
Do *not* call this method from the plug-in. This method should be called by the host only. To set parameter values from within the plug-in, use the [AAX_IParameter](#) interface.

Implements [AAX_IACFEffectParameters](#).

Reimplemented in [AAX_CMonolithicParameters](#).

Referenced by [AAX_CMonolithicParameters::GenerateCoefficients\(\)](#).

Here is the caller graph for this function:



14.11.4.31 virtual AAX_Result AAX_CEffectParameters::ResetFieldData (AAX_CFieldIndex *inFieldIndex*, void * *oData*, uint32_t *inDataSize*) const [virtual]

Called by the host to reset a private data field in the plug-in's algorithm.

This method is called sequentially for all private data fields on Effect initialization and during any "reset" event, such as priming for a non-real-time render. This method is called before the algorithm's optional initialization callback, and the initialized private data will be available within that callback via its context block.

See also

[Algorithm initialization](#).

Warning

Any data structures that will be passed between platforms (for example, sent to a TI DSP in an AAX DSP plug-in) must be properly data-aligned for compatibility across both platforms. See [AAX_ALIGN_FILE_AL←G](#) for more information about guaranteeing cross-platform compatibility of data structures used for algorithm processing.

Parameters

in	<i>inFieldIndex</i>	The index of the field that is being initialized
out	<i>oData</i>	The pre-allocated block of data that should be initialized
in	<i>inDataSize</i>	The size of the data block, in bytes

Implements [AAX_IACFEFFECTPARAMETERS](#).

Reimplemented in [AAX_CMonolithicParameters](#).

Referenced by [AAX_CMonolithicParameters::ResetFieldData\(\)](#).

Here is the caller graph for this function:



14.11.4.32 virtual AAX_Result AAX_CEffectParameters::GetNumberOfChunks (int32_t * *oNumChunks*) const [virtual]

Retrieves the number of chunks used by this plug-in.

Parameters

out	<i>oNumChunks</i>	The number of distinct chunks used by this plug-in
-----	-------------------	--

Implements [AAX_IACFEFFECTPARAMETERS](#).

14.11.4.33 virtual AAX_Result AAX_CEffectParameters::GetChunkIDFromIndex (int32_t *iIndex*, AAX_CTypeID * *oChunkID*) const [virtual]

Retrieves the ID associated with a chunk index.

Parameters

in	<i>iIndex</i>	Index of the queried chunk
out	<i>oChunkID</i>	ID of the queried chunk

Implements [AAX_IACFEFFECTPARAMETERS](#).

14.11.4.34 virtual AAX_Result AAX_CEffectParameters::GetChunkSize (AAX_CTypeID *iChunkID*, uint32_t * *oSize*) const [virtual]

Get the size of the data structure that can hold all of a chunk's information.

If *chunkID* is one of the plug-in's custom chunks, initialize **size* to the size of the chunk's data in bytes.

This method is invoked every time a chunk is saved, therefore it is possible to have dynamically sized chunks. However, note that each call to [GetChunkSize\(\)](#) will correspond to a following call to [GetChunk\(\)](#). The chunk provided in [GetChunk\(\)](#) *must* have the same size as the *size* provided by [GetChunkSize\(\)](#).

Legacy Porting Notes In AAX, the value provided by [GetChunkSize\(\)](#) should *NOT* include the size of the chunk header. The value should *ONLY* reflect the size of the chunk's data.

Parameters

in	<i>iChunkID</i>	ID of the queried chunk
out	<i>oSize</i>	The chunk's size in bytes

Implements [AAX_IACFEffectParameters](#).

14.11.4.35 virtual AAX_Result AAX_CEffectParameters::GetChunk (AAX_CTypeID *iChunkID*, AAX_SPlugInChunk * *oChunk*) const [virtual]

Fills a block of data with chunk information representing the plug-in's current state.

By calling this method, the host is requesting information about the current state of the plug-in. The following chunk fields should be explicitly populated in this method. Other fields will be populated by the host.

- [AAX_SPlugInChunk::fData](#)
- [AAX_SPlugInChunk::fVersion](#)
- [AAX_SPlugInChunk::fName](#) (Optional)
- [AAX_SPlugInChunk::fSize](#) (Data size only)

Warning

Remember that this chunk data may be loaded on a different platform from the one where it is saved. All data structures in the chunk must be properly data-aligned for compatibility across all platforms that the plug-in supports. See [AAX_ALIGN_FILE_ALG](#) for notes about common cross-platform pitfalls for data structure alignment.

Parameters

in	<i>iChunkID</i>	ID of the chunk that should be provided
out	<i>oChunk</i>	A preallocated block of memory that should be populated with the chunk's data.

Implements [AAX_IACFEffectParameters](#).

14.11.4.36 virtual AAX_Result AAX_CEffectParameters::SetChunk (AAX_CTypeID *iChunkID*, const AAX_SPlugInChunk * *iChunk*) [virtual]

Restores a set of plug-in parameters based on chunk information.

By calling this method, the host is attempting to update the plug-in's current state to match the data stored in a chunk. The plug-in should initialize itself to this new state by calling [SetParameterNormalizedValue\(\)](#) for each of the relevant parameters.

Parameters

in	<i>iChunkID</i>	ID of the chunk that is being set
in	<i>iChunk</i>	The chunk

Implements [AAX_IACFEffectParameters](#).

14.11.4.37 virtual AAX_Result AAX_CEffectParameters::CompareActiveChunk (const AAX_SPlugInChunk * *iChunkP*, AAX_CBoolean * *oIsEqual*) const [virtual]

Determine if a chunk represents settings that are equivalent to the plug-in's current state.

Host Compatibility Notes In Pro Tools, this method will only be called if a prior call to [GetNumberOfChanges\(\)](#) has indicated that the plug-in's state has changed. If the plug-in's current settings are different from the settings in *aChunkP* then the plug-in's Compare Light will be illuminated in the plug-in header, allowing users to toggle between the plug-in's custom state and its saved state.

Parameters

in	<i>iChunkP</i>	The chunk that is to be tested
out	<i>oIsEqual</i>	True if the chunk represents equivalent settings when compared with the plug-in's current state. False if the chunk represents non-equivalent settings

Implements [AAX_IACFEFFECTPARAMETERS](#).

14.11.4.38 virtual AAX_Result AAX_CEffectParameters::GetNumberOfChanges (int32_t * *oNumChanges*) const
[virtual]

Retrieves the number of parameter changes made since the plug-in's creation.

This method is polled regularly by the host, and can additionally be triggered by some events such as mouse clicks. When the number provided by this method changes, the host subsequently calls [CompareActiveChunk\(\)](#) to determine if the plug-in's Compare light should be activated.

The value provided by this method should increment with each call to [UpdateParameterNormalizedValue\(\)](#)

Parameters

out	<i>oNumChanges</i>	Must be set to indicate the number of parameter changes that have occurred since plug-in initialization.
-----	--------------------	--

Implements [AAX_IACFEFFECTPARAMETERS](#).

14.11.4.39 virtual AAX_Result AAX_CEffectParameters::TimerWakeup () [virtual]

Periodic wakeup callback for idle-time operations.

This method is called from the host using a non-main thread. In general, it should be driven at approximately one call per 30 ms. However, the wakeup is not guaranteed to be called at any regular interval - for example, it could be held off by a high real-time processing load - and there is no host contract regarding maximum latency between wakeup calls.

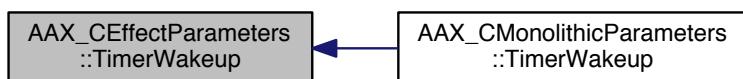
This wakeup thread runs continuously and cannot be armed/disarmed or by the plug-in.

Implements [AAX_IACFEFFECTPARAMETERS](#).

Reimplemented in [AAX_CMonolithicParameters](#).

Referenced by [AAX_CMonolithicParameters::TimerWakeup\(\)](#).

Here is the caller graph for this function:



```
14.11.4.40 virtual AAX_Result AAX_CEffectParameters::GetCurveData ( AAX_CTypeID iCurveType, const float * iValues,
    uint32_t iNumValues, float * oValues ) const [virtual]
```

Generate a set of output values based on a set of given input values.

This method is used by the host to generate graphical curves. Given a set of input values, e.g. frequencies in Hz, this method should generate a corresponding set of output values, e.g. dB gain at each frequency. The semantics of these input and output values are dictated by `iCurveType`. See [AAX_ECurveType](#).

Plug-ins may also define custom curve type IDs to use this method internally. For example, the plug-in's GUI could use this method to request curve data in an arbitrary format.

Note

- This method may be called by the host simultaneously from multiple threads with different `iValues`.

Note

- `oValues` must be allocated by caller with the same size as `iValues` (`iNumValues`).

Host Compatibility Notes Versions of S6 software which support the [GetCurveDataDisplayRange\(\)](#) method will not display a plug-in's curve data unless both [GetCurveData\(\)](#) and [GetCurveDataDisplayRange\(\)](#) are supported by the plug-in.

Warning

S6 currently polls this method to update a plug-in's EQ or dynamics curves based on changes to the parameters mapped to the plug-in's EQ or dynamics center section page tables. Parameters that are not included in these page tables will not trigger updates to the curves displayed on S6. ([GWSW-7314](#), [PTSW-195316](#) / [PT-218485](#))

Parameters

in	<code>iCurveType</code>	One of AAX_ECurveType
in	<code>iValues</code>	An array of input values
in	<code>iNumValues</code>	The size of <code>iValues</code>
out	<code>oValues</code>	An array of output values

Returns

This method must return [AAX_ERROR_UNIMPLEMENTED](#) if the plug-in does not support curve data for the requested `iCurveType`

Implements [AAX_IACFEffectParameters](#).

```
14.11.4.41 virtual AAX_Result AAX_CEffectParameters::GetCurveDataMeterIds ( AAX_CTypeID iCurveType, uint32_t * oXMeterId, uint32_t * oYMeterId ) const [virtual]
```

Indicates which meters correspond to the X and Y axes of the EQ or Dynamics graph.

These meters can be used by attached control surfaces to present an indicator in the same X/Y coordinate plane as the plug-in's curve data.

Parameters

in	<code>iCurveType</code>	One of AAX_ECurveType
----	-------------------------	---------------------------------------

out	<i>oXMeterId</i>	Id of the X-axis meter
out	<i>oYMeterId</i>	Id of the Y-axis meter

Returns

This method should return [AAX_ERROR_UNIMPLEMENTED](#) if the plug-in does not implement it.

Implements [AAX_IACFEFFECTPARAMETERS_V3](#).

14.11.4.42 virtual AAX_Result AAX_CEffectParameters::GetCurveDataDisplayRange (AAX_CTypeID *iCurveType*, float * *oXMin*, float * *oXMax*, float * *oYMin*, float * *oYMax*) const [virtual]

Determines the range of the graph shown by the plug-in.

Min/max arguments define the range of the axes of the graph.

Parameters

in	<i>iCurveType</i>	One of AAX_ECurveType
out	<i>oXMin</i>	Min value of X-axis range
out	<i>oXMax</i>	Max value of X-axis range
out	<i>oYMin</i>	Min value of Y-axis range
out	<i>oYMax</i>	Max value of Y-axis range

Returns

This method should return [AAX_ERROR_UNIMPLEMENTED](#) if the plug-in does not implement it.

Implements [AAX_IACFEFFECTPARAMETERS_V3](#).

14.11.4.43 virtual AAX_Result AAX_CEffectParameters::UpdatePageTable (uint32_t *inTableType*, int32_t *inTablePageSize*, IACFUnknown * *iHostUnknown*, IACFUnknown * *ioPageTableUnknown*) const [virtual]

Allow the plug-in to update its page tables.

Called by the plug-in host, usually in response to a [AAX_eNotificationEvent_ParameterMappingChanged](#) notification sent from the plug-in.

Use this method to change the page table mapping for the plug-in instance or to apply other changes to auxiliary UIs which use the plug-in page tables, such as setting focus to a new page.

See [Page Table Guide](#) for more information about page tables.

Parameters

in	<i>inTableType</i>	Four-char type identifier for the table type (e.g. 'PgTL', 'Av81', etc.)
in	<i>inTablePageSize</i>	Page size for the table
in	<i>iHostUnknown</i>	<p>Unknown interface from the host which may support interfaces providing additional features or information.</p> <p>All interfaces queried from this unknown will be valid only within the scope of this UpdatePageTable() execution and will be relevant for only the current plug-in instance.</p>

in, out	<i>ioPageTable</i> ← <i>Unknown</i>	Unknown interface which supports AAX_IPageTable . This object represents the page table data which is currently stored by the host for this plug-in instance for the given table type and page size. This data and may be edited within the scope of UpdatePageTable() to change the page table mapping for this plug-in instance.
---------	--	--

Returns

This method should return [AAX_ERROR_UNIMPLEMENTED](#) if the plug-in does not implement it or when no change is requested by the plug-in. This allows optimizations to be used in the host when no UI update is required following this call.

See also

[AAX_eNotificationEvent_ParameterMappingChanged](#)

Note

For convenience, do not override this method. Instead, override the [protected overload](#) which provides a prepared copy of the relevant [AAX_IPageTable](#) host interface.

Implements [AAX_IACFEFFECTPARAMETERS_V4](#).

14.11.4.44 virtual [AAX_Result](#) [AAX_CEffectParameters::GetCustomData](#)([AAX_CTypeID](#) *iDataBlockID*, [uint32_t](#)
inDataSize, void * *oData*, [uint32_t](#) * *oDataWritten*) const [virtual]

An optional interface hook for getting custom data from another module.

Parameters

in	<i>iDataBlockID</i>	Identifier for the requested block of custom data
in	<i>inDataSize</i>	Size of provided buffer, in bytes
out	<i>oData</i>	Pointer to an allocated buffer. Data will be written here.
out	<i>oDataWritten</i>	The number of bytes actually written

Implements [AAX_IACFEFFECTPARAMETERS](#).

14.11.4.45 virtual [AAX_Result](#) [AAX_CEffectParameters::SetCustomData](#)([AAX_CTypeID](#) *iDataBlockID*, [uint32_t](#)
inDataSize, const void * *iData*) [virtual]

An optional interface hook for setting custom data for use by another module.

Parameters

in	<i>iDataBlockID</i>	Identifier for the provided block of custom data
in	<i>inDataSize</i>	Size of provided buffer, in bytes
in	<i>iData</i>	Pointer to the data buffer

Implements [AAX_IACFEFFECTPARAMETERS](#).

14.11.4.46 virtual [AAX_Result](#) [AAX_CEffectParameters::DoMIDITransfers](#)() [inline], [virtual]

MIDI update callback.

Call [AAX_IController::GetNextMIDIpacket\(\)](#) from within this method to retrieve and process MIDI packets directly within the Effect's data model. MIDI data will also be delivered to the Effect algorithm.

This method is called regularly by the host, similarly to [AAX_IEffectParameters::TimerWakeup\(\)](#)

Implements [AAX_IACFEFFECTPARAMETERS](#).

References [AAX_SUCCESS](#).

**14.11.4.47 virtual AAX_Result AAX_CEffectParameters::UpdateMIDINodes (AAX_CFieldIndex *inFieldIndex*,
AAX_CMidiPacket & *iPacket*) [virtual]**

MIDI update callback.

This method is called by the host for each pending MIDI packet for MIDI nodes in algorithm context structure. Overwrite this method in Plug-In's EffectParameter class if you want to receive MIDI data packets directly in the data model. MIDI data will also be delivered to the Effect algorithm.

The host calls this method in Effects that register one or more MIDI nodes using [AAX_IComponentDescriptor::AddMIDINode\(\)](#). Effects that do not require MIDI data to be sent to the plug-in algorithm should override [UpdateControlMIDINodes\(\)](#).

Parameters

in	<i>inFieldIndex</i>	MIDI node field index in algorithm context structure
in	<i>iPacket</i>	The incoming MIDI packet for the node

Implements [AAX_IACFEFFECTPARAMETERS_V2](#).

**14.11.4.48 virtual AAX_Result AAX_CEffectParameters::UpdateControlMIDINodes (AAX_CTypeID *nodeID*,
AAX_CMidiPacket & *iPacket*) [virtual]**

MIDI update callback for control MIDI nodes.

This method is called by the host for each pending MIDI packet for Control MIDI nodes. Overwrite this method in Plug-In's EffectParameter class if you want to receive MIDI data packets directly in the data model.

The host calls this method in Effects that register one or more Control MIDI nodes using [AAX_IEffectDescriptor::AddControlMIDINode\(\)](#). Effects with algorithms that use MIDI data nodes should override [UpdateMIDINodes\(\)](#).

Note

This method will not be called if an Effect includes any MIDI nodes in its algorithm context structure.

Parameters

in	<i>nodeID</i>	Identifier for the MIDI node
in	<i>iPacket</i>	The incoming MIDI packet for the node

Implements [AAX_IACFEFFECTPARAMETERS_V2](#).

14.11.4.49 virtual AAX_Result AAX_CEffectParameters::RenderAudio_Hybrid (AAX_SHybridRenderInfo * *ioRenderInfo*) [virtual]

Hybrid audio render function.

This method is called from the host to render audio for the hybrid piece of the algorithm.

Note

To use this method plug-in should register some hybrid inputs and outputs in "Describe"

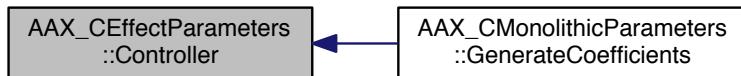
Implements [AAX_IACFEFFECTPARAMETERS_V2](#).

14.11.4.50 `AAX_IController* AAX_CEffectParameters::Controller()`

Access to the Effect controller.

Referenced by `AAX_CMonolithicParameters::GenerateCoefficients()`.

Here is the caller graph for this function:



14.11.4.51 `const AAX_IController* AAX_CEffectParameters::Controller() const`

`const` access to the Effect controller

14.11.4.52 `AAX_ITransport* AAX_CEffectParameters::Transport()`

Access to the Transport object.

14.11.4.53 `const AAX_ITransport* AAX_CEffectParameters::Transport() const`

`const` access to the Transport object

14.11.4.54 `AAX_IAutomationDelegate* AAX_CEffectParameters::AutomationDelegate()`

Access to the Effect's automation delegate.

14.11.4.55 `const AAX_IAutomationDelegate* AAX_CEffectParameters::AutomationDelegate() const`

`const` access to the Effect's automation delegate

14.11.4.56 `AAX_Result AAX_CEffectParameters::SetTaperDelegate(AAX_CParamID iParameterID, AAX_ITaperDelegateBase & iTaperDelegate, bool iPreserveValue) [protected]`

14.11.4.57 `AAX_Result AAX_CEffectParameters::SetDisplayDelegate(AAX_CParamID iParameterID, AAX_IDisplayDelegateBase & iDisplayDelegate) [protected]`

14.11.4.58 `bool AAX_CEffectParameters::IsParameterTouched(AAX_CParamID iParameterID) const [protected]`

14.11.4.59 `bool AAX_CEffectParameters::IsParameterLinkReady(AAX_CParamID inParameterID, AAX_EUpdateSource inSource) const [protected]`

14.11.4.60 `virtual AAX_Result AAX_CEffectParameters::EffectInit(void) [inline], [protected], [virtual]`

Initialization helper routine. Called from [AAX_CEffectParameters::Initialize](#).

Override to add parameters, packets, meters, and to do any other custom initialization.

Add custom parameters:

- Create an [AAX_CParameter](#) for each parameter in the plug-in
- Call [AAX_CParameterManager::AddParameter\(\)](#) using `mParameterManager` to add parameters to the Parameter Manager

Register packets:

- Call [AAX_CPacketDispatcher::RegisterPacket\(\)](#) using `mPacketDispatcher` to register a packet and handling callback.

References [AAX_SUCCESS](#).

14.11.4.61 virtual AAX_Result AAX_CEffectParameters::UpdatePageTable(uint32_t, int32_t, AAX_IPageTable &) const [inline], [protected], [virtual]

Protected overload of [UpdatePageTable\(\)](#)

Override this version of the method for convenience. This allows the default [UpdatePageTable\(\)](#) implementation to handle the interface conversion from [IACFUnknown](#) to [AAX_IPageTable](#).

Returns

This method should return [AAX_ERROR_UNIMPLEMENTED](#) if the plug-in does not implement it or when no change is made by the plug-in. This allows optimizations to be used in the host when no UI update is required following this call.

References [AAX_ERROR_UNIMPLEMENTED](#).

14.11.4.62 void AAX_CEffectParameters::FilterParameterIDOnSave(AAX_CParamID controlID) [protected]

CALL: Indicates the indices of parameters that should not be saved in the default [AAX_CEffectParameters](#) chunk.

Allows specific parameters to filtered out of the default [AAX_CEffectParameters](#) "Save Settings" functionality. This call is automatically invoked on the Master Bypass control when specified by the `DefineMasterBypassControlIndex()` call.

Parameters

in	<code>controlID</code>	The ID of the parameter that should be removed from the default chunk
----	------------------------	---

14.11.4.63 void AAX_CEffectParameters::BuildChunkData(void) const [protected]

Clears out the current chunk in Chunk Parser and adds all of the new values. Used by default implementations of [GetChunk\(\)](#) and [GetChunkSize\(\)](#).

14.11.5 Member Data Documentation

14.11.5.1 int32_t AAX_CEffectParameters::mNumPluginChanges [protected]

14.11.5.2 int32_t AAX_CEffectParameters::mChunkSize [mutable], [protected]

14.11.5.3 AAX_CChunkDataParser AAX_CEffectParameters::mChunkParser [mutable], [protected]

14.11.5.4 `int32_t AAX_CEffectParameters::mNumChunkedParameters` [protected]

14.11.5.5 `AAX_CBoolean AAX_CEffectParameters::mClipped` [protected]

Set by `SetClipped()` and returned by `GetClipped()`.

14.11.5.6 `AAX_CPacketDispatcher AAX_CEffectParameters::mPacketDispatcher` [protected]

14.11.5.7 `AAX_CParameterManager AAX_CEffectParameters::mParameterManager` [protected]

Referenced by `AAX_CMonolithicParameters::UpdateParameterNormalizedValue()`.

14.11.5.8 `std::set<std::string> AAX_CEffectParameters::mFilteredParameters` [protected]

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

- [AAX_CEffectParameters.h](#)

14.12 AAX_CheckedResult Class Reference

```
#include <AAX_Exception.h>
```

14.12.1 Description

Error checker convenience class for [AAX_Result](#)

Implicitly convertible to an [AAX_Result](#).

Provides an overloaded `operator=()` which will throw an [AAX::Exception::ResultError](#) if assigned a non-success result.

Warning

Never use this class outside of an exception catch scope

If the host supports [AAX_TRACE](#) tracing, a log is emitted when the exception is thrown. A stacktrace is added if the host's trace priority filter level is set to [kAAX_Trace_Priority_Lowest](#)

When an error is encountered, [AAX_CheckedResult](#) throws an [AAX_CheckedResult::Exception](#) exception and clears its internal result value.

```
#include "AAX_Exception.h"
AAX_Result SomeCheckedMethod()
{
    AAX_Result result = AAX_SUCCESS;
    try {
        AAX_CheckedResult cr;
        cr = ResultFunc1();
        cr = ResultFunc2();
    }
    catch (const AAX_CheckedResult::Exception& ex)
    {
        // handle exception; do not rethrow
        result = ex.Result();
    }
    catch (...){
        result = AAX_ERROR_UNKNOWN_EXCEPTION;
    }
    return result;
}
```

Note

The [AAX Library](#) method which calls `GetEffectDescriptions()` on the plug-in includes an appropriate exception handler, so [AAX_CheckedResult](#) objects may be used within a plug-in's describe code without additional catch scopes.

```
#include "AAX_Exception.h"
AAX_Result GetEffectDescriptions( AAX_ICollection *
    outCollection )
{
    AAX_CheckedResult cr;
    cr = MyDescriptionSubroutine();
    cr = outCollection->AddEffect(...);
    // etc.
    return cr;
}
```

It is assumed that the exception handler will resolve any error state and that the [AAX_CheckedResult](#) may therefore continue to be used from a clean state following the exception catch block.

If the previous error value is required then it can be retrieved using [AAX_CheckedResult::LastError\(\)](#).

```
// in this example, the exception is handled and
// success is returned from MyFunc1()
AAX_Result MyFunc1()
{
    AAX_CheckedResult cr;

    try {
        cr = MethodThatReturnsError();
    } catch (const AAX::Exception::ResultError& ex) {
        // exception is fully handled here
    }

    // cr now holds a success value
    return cr;
}

// in this example, MyFunc2() returns the first
// non-successful value which was encountered
AAX_Result MyFunc2()
{
    AAX_CheckedResult cr;

    try {
        AAX_SWALLOW(cr = MethodThatMayReturnError1());
        AAX_SWALLOW(cr = MethodThatMayReturnError2());
        cr = MethodThatMayReturnError3();
    } catch (const AAX::Exception::ResultError& ex) {
        // exception might not be fully handled
    }

    // pass the last error on to the caller
    return cr.LastError();
}
```

It is possible to add one or more accepted non-success values to an [AAX_CheckedResult](#) so that these values will not trigger exceptions:

```
AAX_CheckedResult cr;
try {
    cr.AddAcceptedResult(AcceptableErrCode);
    cr = MethodThatReturnsAcceptedError();
    cr = MethodThatReturnsAnotherError();
} catch (const AAX::Exception::ResultError& ex) {
    // handle the exception
}
```

Public Types

- `typedef AAX::Exception::ResultError Exception`

Public Member Functions

- `~AAX_CheckedResult ()`

- [AAX_CheckedResult \(\)](#)
Construct an AAX_CheckedResult in a success state.
- [AAX_CheckedResult \(AAX_Result inResult\)](#)
Implicit conversion constructor from AAX_Result.
- [void AddAcceptedResult \(AAX_Result inResult\)](#)
Add an expected result which will not result in a throw.
- [void ResetAcceptedResults \(\)](#)
- [AAX_CheckedResult & operator= \(AAX_Result inResult\)](#)
Assignment to AAX_Result.
- [AAX_CheckedResult & operator|= \(AAX_Result inResult\)](#)
bitwise-or assignment to AAX_Result
- [operator AAX_Result \(\) const](#)
Conversion to AAX_Result.
- [void Clear \(\)](#)
Clears the current result state.
- [AAX_Result LastError \(\) const](#)
Get the last non-success result which was stored in this object, or AAX_SUCCESS if no non-success result was ever stored in this object.

14.12.2 Member Typedef Documentation

14.12.2.1 [typedef AAX::Exception::ResultError AAX_CheckedResult::Exception](#)

14.12.3 Constructor & Destructor Documentation

14.12.3.1 [AAX_CheckedResult::~AAX_CheckedResult \(\) \[inline\]](#)

14.12.3.2 [AAX_CheckedResult::AAX_CheckedResult \(\) \[inline\]](#)

Construct an AAX_CheckedResult in a success state.

14.12.3.3 [AAX_CheckedResult::AAX_CheckedResult \(AAX_Result inResult \) \[inline\]](#)

Implicit conversion constructor from AAX_Result.

Implicit conversion is OK in order to support `AAX_CheckedResult cr = SomeFunc()`

14.12.4 Member Function Documentation

14.12.4.1 [void AAX_CheckedResult::AddAcceptedResult \(AAX_Result inResult \) \[inline\]](#)

Add an expected result which will not result in a throw.

It is acceptable for some methods to return certain non-success values such as `AAX_RESULT_PACKET_STRE→AM_NOT_EMPTY` or `AAX_RESULT_NEW_PACKET_POSTED`

14.12.4.2 [void AAX_CheckedResult::ResetAcceptedResults \(\) \[inline\]](#)

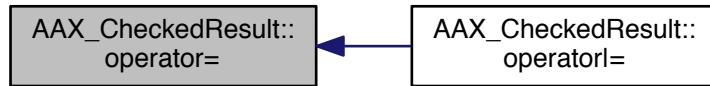
References AAX_SUCCESS.

14.12.4.3 AAX_CheckedResult& AAX_CheckedResult::operator=(AAX_Result *inResult*) [inline]

Assignment to [AAX_Result](#).

Referenced by [operator|=\(\)](#).

Here is the caller graph for this function:

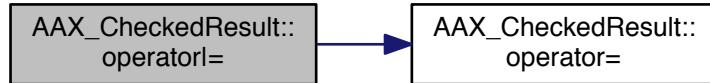
**14.12.4.4 AAX_CheckedResult& AAX_CheckedResult::operator|= (AAX_Result *inResult*) [inline]**

bitwise-or assignment to [AAX_Result](#)

Sometimes used in legacy code to aggregate results into a single AAX_Result value

References [operator=\(\)](#).

Here is the call graph for this function:

**14.12.4.5 AAX_CheckedResult::operator AAX_Result() const [inline]**

Conversion to [AAX_Result](#).

14.12.4.6 void AAX_CheckedResult::Clear() [inline]

Clears the current result state.

Does not affect the set of accepted results

References [AAX_SUCCESS](#).

14.12.4.7 AAX_Result AAX_CheckedResult::LastError() const [inline]

Get the last non-success result which was stored in this object, or [AAX_SUCCESS](#) if no non-success result was ever stored in this object.

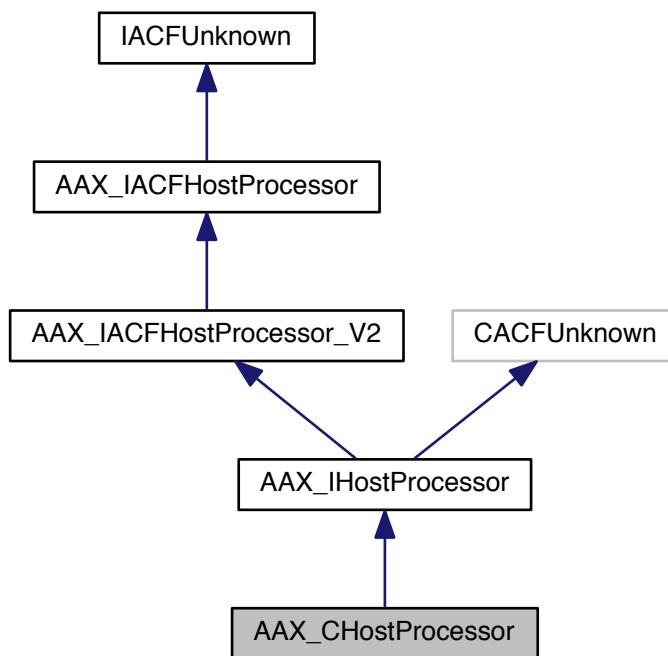
The documentation for this class was generated from the following file:

- [AAX_Exception.h](#)

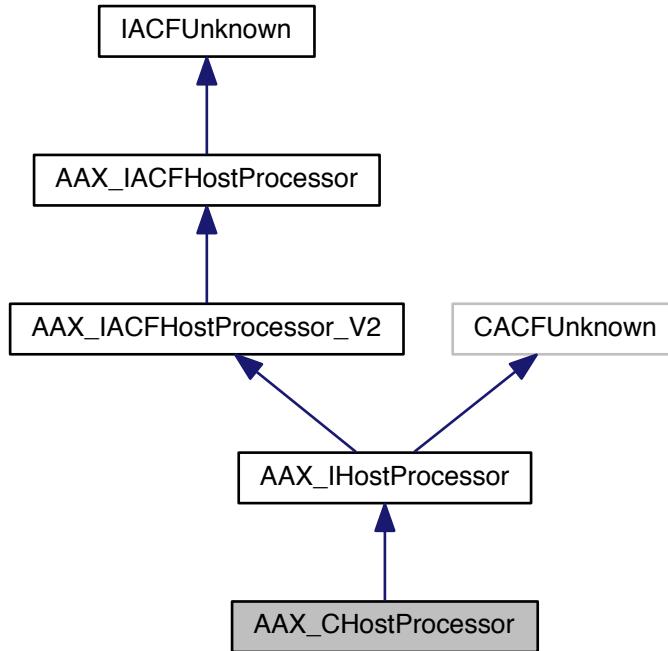
14.13 AAX_CHostProcessor Class Reference

```
#include <AAX_CHostProcessor.h>
```

Inheritance diagram for AAX_CHostProcessor:



Collaboration diagram for AAX_CHostProcessor:



14.13.1 Description

Concrete implementation of the [AAX_IHostProcessor](#) interface for non-real-time processing.

Host processor objects are used to process regions of audio data in a non-real-time context.

- Host processors must generate output samples linearly and incrementally, but may randomly access samples from the processing region on the timeline for input. See [GetAudio\(\)](#) for more information.
- Host processors may re-define the processing region using [AAX_CHostProcessor::TranslateOutputBounds\(\)](#).

See also

[AAX_IHostProcessorDelegate](#)

Public Member Functions

- [AAX_CHostProcessor \(void\)](#)
- virtual [~AAX_CHostProcessor \(\)](#)

Initialization and uninitialized

- virtual [AAX_Result Initialize \(IACFUnknown *iController\) AAX_OVERRIDE](#)
Host Processor initialization.
- virtual [AAX_Result Uninitialize \(\) AAX_OVERRIDE](#)
Host Processor teardown.

Host processor interface

- virtual `AAX_Result InitOutputBounds` (int64_t iSrcStart, int64_t iSrcEnd, int64_t *oDstStart, int64_t *oDstEnd) **AAX_OVERRIDE**
Sets the processing region.
- virtual `AAX_Result SetLocation` (int64_t iSample) **AAX_OVERRIDE**
Updates the relative sample location of the current processing frame.
- virtual `AAX_Result RenderAudio` (const float *const inAudioIns[], int32_t inAudioInCount, float *const i← AudioOuts[], int32_t iAudioOutCount, int32_t *ioWindowSize) **AAX_OVERRIDE**
Perform the signal processing.
- virtual `AAX_Result PreRender` (int32_t inAudioInCount, int32_t iAudioOutCount, int32_t iWindowSize) **A← AX_OVERRIDE**
Invoked right before the start of a Preview or Render pass.
- virtual `AAX_Result PostRender` () **AAX_OVERRIDE**
Invoked at the end of a Render pass.
- virtual `AAX_Result AnalyzeAudio` (const float *const inAudioIns[], int32_t inAudioInCount, int32_t *io← WindowSize) **AAX_OVERRIDE**
Override this method if the plug-in needs to analyze the audio prior to a Render pass.
- virtual `AAX_Result PreAnalyze` (int32_t inAudioInCount, int32_t iWindowSize) **AAX_OVERRIDE**
Invoked right before the start of an Analysis pass.
- virtual `AAX_Result PostAnalyze` () **AAX_OVERRIDE**
Invoked at the end of an Analysis pass.
- virtual `AAX_Result GetClipNameSuffix` (int32_t inMaxLength, `AAX_IString` *outString) const **AAX_OVE← RRIDE**
Called by host application to retrieve a custom string to be appended to the clip name.

Convenience methods

- `AAX_IEffectParameters * GetEffectParameters` ()
- const `AAX_IEffectParameters * GetEffectParameters` () const
- `AAX_IHostProcessorDelegate * GetHostProcessorDelegate` ()
- const `AAX_IHostProcessorDelegate * GetHostProcessorDelegate` () const
- int64_t `GetLocation` () const
The relative sample location of the current processing frame.
- int64_t `GetInputRange` () const
The length (in samples) of the current timeline selection.
- int64_t `GetOutputRange` () const
The length (in samples) of the clip that will be rendered to the timeline.
- int64_t `GetSrcStart` () const
The sample position of the beginning of the current timeline selection relative to the beginning of the current input selection, i.e. 0.
- int64_t `GetSrcEnd` () const
The sample position of the end of the current timeline selection relative to the beginning of the current input selection.
- int64_t `GetDstStart` () const
The sample position of the beginning of the clip that will be rendered to the timeline relative to the beginning of the current input selection.
- int64_t `GetDstEnd` () const
The sample position of the end of the clip that will be rendered to the timeline relative to the beginning of the current input selection.
- virtual `AAX_Result TranslateOutputBounds` (int64_t iSrcStart, int64_t iSrcEnd, int64_t &oDstStart, int64_t &oDstEnd)
Define the boundaries of the clip that will be rendered to the timeline.
- virtual `AAX_Result GetAudio` (const float *const inAudioIns[], int32_t inAudioInCount, int64_t inLocation, int32_t *ioNumSamples)
Randomly access audio from the timeline.

- virtual int32_t [GetSideChainInputNum\(\)](#)
CALL: Returns the index of the side chain input buffer.
- [AAX_IController * Controller\(\)](#)
- const [AAX_IController * Controller\(\)](#) const
- [AAX_IHostProcessorDelegate * HostProcessorDelegate\(\)](#)
- const [AAX_IHostProcessorDelegate * HostProcessorDelegate\(\)](#) const
- [AAX_IEffectParameters * EffectParameters\(\)](#)
- const [AAX_IEffectParameters * EffectParameters\(\)](#) const

Additional Inherited Members

14.13.2 Constructor & Destructor Documentation

14.13.2.1 [AAX_CHostProcessor::AAX_CHostProcessor\(void \)](#)

14.13.2.2 virtual [AAX_CHostProcessor::~AAX_CHostProcessor\(\)](#) [virtual]

14.13.3 Member Function Documentation

14.13.3.1 virtual [AAX_Result AAX_CHostProcessor::Initialize\(IACFUnknown * iController \)](#) [virtual]

Host Processor initialization.

Parameters

in	<i>iController</i>	A versioned reference that can be resolved to both an AAX_IController interface and an AAX_IHostProcessorDelegate
----	--------------------	---

Implements [AAX_IACFHostProcessor](#).

14.13.3.2 virtual [AAX_Result AAX_CHostProcessor::Uninitialize\(\)](#) [virtual]

Host Processor teardown.

Implements [AAX_IACFHostProcessor](#).

14.13.3.3 virtual [AAX_Result AAX_CHostProcessor::InitOutputBounds\(int64_t iSrcStart, int64_t iSrcEnd, int64_t * oDstStart, int64_t * oDstEnd \)](#) [virtual]

Sets the processing region.

This method allows offline processing plug-ins to vary the length and/or start/end points of the audio processing region.

This method is called in a few different scenarios:

- Before an analyze, process or preview of data begins.
- At the end of every preview loop.
- After the user makes a new data selection on the timeline.

Plug-ins that inherit from [AAX_CHostProcessor](#) should not override this method. Instead, use the following convenience functions:

- To retrieve the length or boundaries of the processing region, use [GetInputRange\(\)](#), [GetSrcStart\(\)](#), etc.
- To change the boundaries of the processing region before processing begins, use [AAX_CHostProcessor::TranslateOutputBounds\(\)](#)

Note

Currently, a host processor may not randomly access samples outside of the boundary defined by `oDstStart` and `oDstEnd`.

Legacy Porting Notes DAE no longer makes use of the `mStartBound` and `mEndBounds` member variables that existed in the legacy RTAS/TDM SDK. Use `oDstStart` and `oDstEnd` instead (preferably by overriding [TranslateOutputBounds\(\)](#)).

Parameters

in	<i>iSrcStart</i>	The selection start of the user selected region. This is will always return 0 for a given selection on the timeline.
in	<i>iSrcEnd</i>	The selection end of the user selected region. This will always return the value of the selection length on the timeline.
in	<i>oDstStart</i>	The starting sample location in the output audio region. By default, this is the same as <code>iSrcStart</code> .
in	<i>oDstEnd</i>	The ending sample location in the output audio region. By default, this is the same as <code>iSrcEnd</code> .

Implements [AAX_IACFHostProcessor](#).

14.13.3.4 virtual AAX_Result AAX_CHostProcessor::SetLocation(int64_t iSample) [virtual]

Updates the relative sample location of the current processing frame.

This method is called by the host to update the relative sample location of the current processing frame.

Note

Plug-ins should not override this method; instead, use [AAX_CHostProcessor::GetLocation\(\)](#) to retrieve the current relative sample location.

Parameters

in	<i>iSample</i>	The sample location of the first sample in the current processing frame relative to the beginning of the full processing buffer
----	----------------	---

Implements [AAX_IACFHostProcessor](#).

14.13.3.5 virtual AAX_Result AAX_CHostProcessor::RenderAudio(const float *const *inAudioIns*[], int32_t *inAudioInCount*, float *const *iAudioOuts*[], int32_t *iAudioOutCount*, int32_t * *ioWindowSize*) [virtual]

Perform the signal processing.

This method is called by the host to invoke the plug-in's signal processing.

Legacy Porting Notes This method is a replacement for the AudioSuite `ProcessAudio` method

Parameters

in	<i>inAudioIns</i>	Input audio buffer
in	<i>inAudioInCount</i>	The number if input channels
in	<i>iAudioOuts</i>	The number of output channels

in	<i>iAudioOutCount</i>	A user defined destination end of the ingested audio
in	<i>ioWindowSize</i>	Window buffer length of the received audio

Implements [AAX_IACFHostProcessor](#).

14.13.3.6 virtual **AAX_Result** **AAX_CHostProcessor::PreRender** (**int32_t** *inAudioInCount*, **int32_t** *iAudioOutCount*, **int32_t** *iWindowSize*) [virtual]

Invoked right before the start of a Preview or Render pass.

This method is called by the host to allow a plug-in to make any initializations before processing actually begins. Upon a Preview pass, PreRender will also be called at the beginning of every "loop".

See also

[AAX_eProcessingState_StartPass](#), [AAX_eProcessingState_BeginPassGroup](#)

Parameters

in	<i>inAudioInCount</i>	The number if input channels
in	<i>iAudioOutCount</i>	The number of output channels
in	<i>iWindowSize</i>	Window buffer length of the ingested audio

Implements [AAX_IACFHostProcessor](#).

14.13.3.7 virtual **AAX_Result** **AAX_CHostProcessor::PostRender** () [virtual]

Invoked at the end of a Render pass.

Note

Upon a Preview pass, PostRender will not be called until Preview has stopped.

See also

[AAX_eProcessingState_StopPass](#), [AAX_eProcessingState_EndPassGroup](#)

Implements [AAX_IACFHostProcessor](#).

14.13.3.8 virtual **AAX_Result** **AAX_CHostProcessor::AnalyzeAudio** (**const float ***const *inAudios*[], **int32_t** *inAudioInCount*, **int32_t** * *ioWindowSize*) [virtual]

Override this method if the plug-in needs to analyze the audio prior to a Render pass.

Use this after declaring the appropriate properties in **Describe**. See [AAX_eProperty_RequiresAnalysis](#) and [AAX_eProperty_OptionalAnalysis](#)

To request an analysis pass from within a plug-in, use [AAX_IHostProcessorDelegate::ForceAnalyze\(\)](#)

Legacy Porting Notes Ported from AudioSuite's **AnalyzeAudio(bool isMasterBypassed)** method

Parameters

in	<i>inAudioIns</i>	Input audio buffer
in	<i>inAudioInCount</i>	The number of input channels
in	<i>iWindowSize</i>	Window buffer length of the ingested audio

Implements [AAX_IACFHostProcessor](#).

14.13.3.9 virtual **AAX_Result** AAX_CHostProcessor::PreAnalyze (*int32_t inAudioInCount, int32_t iWindowSize*)
[virtual]

Invoked right before the start of an Analysis pass.

This method is called by the host to allow a plug-in to make any initializations before an Analysis pass actually begins.

See also

[AAX_eProcessingState_StartPass](#), [AAX_eProcessingState_BeginPassGroup](#)

Parameters

in	<i>inAudioInCount</i>	The number if input channels
in	<i>iWindowSize</i>	Window buffer length of the ingested audio

Implements [AAX_IACFHostProcessor](#).

14.13.3.10 virtual **AAX_Result** AAX_CHostProcessor::PostAnalyze () [virtual]

Invoked at the end of an Analysis pass.

See also

[AAX_eProcessingState_StopPass](#), [AAX_eProcessingState_EndPassGroup](#)

Implements [AAX_IACFHostProcessor](#).

14.13.3.11 virtual **AAX_Result** AAX_CHostProcessor::GetClipNameSuffix (*int32_t inMaxLength, AAX_IString * outString*) const [virtual]

Called by host application to retrieve a custom string to be appended to the clip name.

If no string is provided then the host's default will be used.

Parameters

in	<i>inMaxLength</i>	The maximum allowed string length, not including the NULL terminating char
out	<i>outString</i>	Add a value to this string to provide a custom clip suffix

Implements [AAX_IACFHostProcessor_V2](#).

14.13.3.12 **AAX_IEffectParameters*** AAX_CHostProcessor::GetEffectParameters (*void*) [inline]

14.13.3.13 **const AAX_IEffectParameters*** AAX_CHostProcessor::GetEffectParameters (*void*) const [inline]

14.13.3.14 **AAX_IHostProcessorDelegate*** AAX_CHostProcessor::GetHostProcessorDelegate () [inline]

14.13.3.15 **const AAX_IHostProcessorDelegate*** AAX_CHostProcessor::GetHostProcessorDelegate () const [inline]

14.13.3.16 int64_t AAX_CHostProcessor::GetLocation() const [inline]

The relative sample location of the current processing frame.

This method returns the relative sample location for the current [RenderAudio\(\)](#) processing frame. For example, if a value of 10 is provided for the [RenderAudio\(\)](#) `ioWindow` parameter, then calls to this method from within each execution of [RenderAudio\(\)](#) will return 0, 10, 20,...

14.13.3.17 int64_t AAX_CHostProcessor::GetInputRange() const [inline]

The length (in samples) of the current timeline selection.

14.13.3.18 int64_t AAX_CHostProcessor::GetOutputRange() const [inline]

The length (in samples) of the clip that will be rendered to the timeline.

14.13.3.19 int64_t AAX_CHostProcessor::GetSrcStart() const [inline]

The sample position of the beginning of the current timeline selection relative to the beginning of the current input selection, i.e. 0.

14.13.3.20 int64_t AAX_CHostProcessor::GetSrcEnd() const [inline]

The sample position of the end of the current timeline selection relative to the beginning of the current input selection.

14.13.3.21 int64_t AAX_CHostProcessor::GetDstStart() const [inline]

The sample position of the beginning of the of the clip that will be rendered to the timeline relative to the beginning of the current input selection.

This value will be equal to the value returned by [GetSrcStart\(\)](#) unless the selection boundaries have been modified by overriding [TranslateOutputBounds\(\)](#)

14.13.3.22 int64_t AAX_CHostProcessor::GetDstEnd() const [inline]

The sample position of the end of the of the clip that will be rendered to the timeline relative to the beginning of the current input selection.

This value will be equal to the value returned by [GetSrcStart\(\)](#) unless the selection boundaries have been modified by overriding [TranslateOutputBounds\(\)](#)

14.13.3.23 virtual AAX_Result AAX_CHostProcessor::TranslateOutputBounds(int64_t *iSrcStart*, int64_t *iSrcEnd*, int64_t & *oDstStart*, int64_t & *oDstEnd*) [protected], [virtual]

Define the boundaries of the clip that will be rendered to the timeline.

This method is called from [AAX_CHostProcessor::InitOutputBounds\(\)](#), providing a convenient hook for re-defining the processing region boundaries. See [InitOutputBounds\(\)](#) for more information.

Parameters

in	<i>iSrcStart</i>	The selection start of the user selected region. This is will always return 0 for a given selection on the timeline.
----	------------------	--

in	<i>iSrcEnd</i>	The selection end of the user selected region. This will always return the value of the selection length on the timeline.
in	<i>oDstStart</i>	The starting sample location in the output audio region. By default, this is the same as <i>iSrcStart</i> .
in	<i>oDstEnd</i>	The ending sample location in the output audio region. By default, this is the same as <i>iSrcEnd</i> .

14.13.3.24 virtual AAX_Result AAX_CHostProcessor::GetAudio (const float *const *inAudioIns*[], int32_t *inAudioInCount*, int64_t *inLocation*, int32_t * *ioNumSamples*) [protected], [virtual]

Randomly access audio from the timeline.

This is a convenience wrapper around [AAX_IHostProcessorDelegate::GetAudio\(\)](#).

Parameters

in	<i>inAudioIns</i>	Timeline audio buffer(s). This must be set to <i>inAudioIns</i> from AAX_IHostProcessor::RenderAudio()
----	-------------------	--

Parameters

in	<i>inAudioInCount</i>	Number of buffers in <i>inAudioIns</i> . This must be set to <i>inAudioInCount</i> from AAX_IHostProcessor::RenderAudio()
----	-----------------------	---

Parameters

in	<i>inLocation</i>	A sample location relative to the beginning of the currently processed region, e.g. a value of 0 corresponds to the timeline location returned by AAX_CHostProcessor::GetSrcStart()
in, out	<i>ioNumSamples</i>	<ul style="list-style-type: none"> • Input: The maximum number of samples to read. • Output: The actual number of samples that were read from the timeline

14.13.3.25 virtual int32_t AAX_CHostProcessor::GetSideChainInputNum () [protected], [virtual]

CALL: Returns the index of the side chain input buffer.

This is a convenience wrapper around [AAX_IHostProcessorDelegate::GetSideChainInputNum\(\)](#)

14.13.3.26 AAX_IController* AAX_CHostProcessor::Controller (void) [inline], [protected]

14.13.3.27 const AAX_IController* AAX_CHostProcessor::Controller (void) const [inline], [protected]

14.13.3.28 AAX_IHostProcessorDelegate* AAX_CHostProcessor::HostProcessorDelegate () [inline], [protected]

14.13.3.29 const AAX_IHostProcessorDelegate* AAX_CHostProcessor::HostProcessorDelegate () const [inline], [protected]

14.13.3.30 AAX_IEffectParameters* AAX_CHostProcessor::EffectParameters (void) [inline], [protected]

14.13.3.31 const AAX_IEffectParameters* AAX_CHostProcessor::EffectParameters (void) const [inline], [protected]

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

- [AAX_CHostProcessor.h](#)

14.14 AAX_CHostServices Class Reference

```
#include <AAX_CHostServices.h>
```

14.14.1 Description

Method access to a singleton implementation of the [AAX_IHostServices](#) interface.

Static Public Member Functions

- static void [Set \(IACFUnknown *pUnkHost \)](#)
- static [AAX_Result HandleAssertFailure \(const char *iFile, int32_t iLine, const char *iNote, int32_t iFlags = AAX_eAssertFlags_Default \)](#)
Handle an assertion failure.
- static [AAX_Result Trace \(AAX_ETracePriorityHost iPriority, const char *iMessage, ... \)](#)
Log a trace message.
- static [AAX_Result StackTrace \(AAX_ETracePriorityHost iTracePriority, AAX_ETracePriorityHost iStackTracePriority, const char *iMessage, ... \)](#)
Log a trace message or a stack trace.

14.14.2 Member Function Documentation

14.14.2.1 static void [AAx_CHostServices::Set \(IACFUnknown * pUnkHost \) \[static\]](#)

14.14.2.2 static [AAX_Result AAx_CHostServices::HandleAssertFailure \(const char * iFile, int32_t iLine, const char * iNote, int32_t iFlags = AAX_eAssertFlags_Default \) \[static\]](#)

Handle an assertion failure.

Use this method to delegate assertion failure handling to the host

Use *iFlags* to request that specific behavior be included when handling the failure. This request may not be fulfilled by the host, and absence of a flag does not preclude the host from using that behavior when handling the failure.

Parameters

in	<i>iFile</i>	The name of the file containing the assert check. Usually <code>__FILE__</code>
in	<i>iLine</i>	The line number of the assert check. Usually <code>__LINE__</code>
in	<i>iNote</i>	Text to display related to the assert. Usually the condition which failed
in	<i>iFlags</i>	Bitfield of AAX_EAssertFlags to request specific handling behavior

14.14.2.3 static [AAX_Result AAx_CHostServices::Trace \(AAX_ETracePriorityHost iPriority, const char * iMessage, ... \) \[static\]](#)

Log a trace message.

Parameters

in	<i>iPriority</i>	Priority of the trace, used for log filtering. One of kAAX_Trace_Priority_Low , kAAX_Trace_Priority_Normal , kAAX_Trace_Priority_High
in	<i>iMessage</i>	Message string to log

```
14.14.2.4 static AAX_Result AAX_CHostServices::StackTrace ( AAX_ETracePriorityHost iTracePriority,
    AAX_ETracePriorityHost iStackTracePriority, const char * iMessage, ... ) [static]
```

Log a trace message or a stack trace.

If the logging output filtering is set to include logs with *iStackTracePriority* then both the logging message and a stack trace will be emitted, regardless of *iTracePriority*.

If the logging output filtering is set to include logs with *iTracePriority* but to exclude logs with *iStackTracePriority* then this will emit a normal log with no stack trace.

Parameters

in	<i>iTracePriority</i>	Priority of the trace, used for log filtering. One of kAAX_Trace_Priority_Low , kAAX_Trace_Priority_Normal , kAAX_Trace_Priority_High
in	<i>iStackTracePriority</i>	Priority of the stack trace, used for log filtering. One of kAAX_Trace_Priority_Low , kAAX_Trace_Priority_Normal , kAAX_Trace_Priority_High
in	<i>iMessage</i>	Message string to log

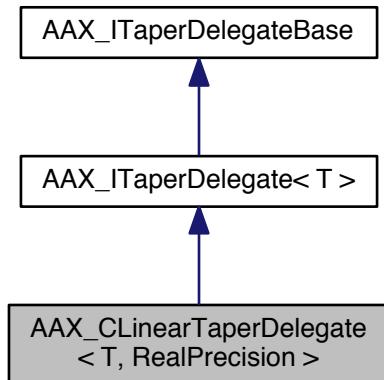
The documentation for this class was generated from the following file:

- [AAX_CHostServices.h](#)

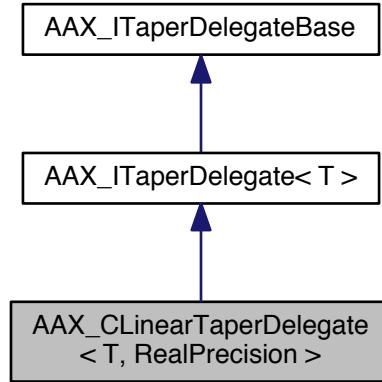
14.15 AAX_CLinearTaperDelegate< T, RealPrecision > Class Template Reference

```
#include <AAX_CLinearTaperDelegate.h>
```

Inheritance diagram for AAX_CLinearTaperDelegate< T, RealPrecision >:



Collaboration diagram for AAX_CLinearTaperDelegate< T, RealPrecision >:



14.15.1 Description

```
template<typename T, int32_t RealPrecision = 0>class AAX_CLinearTaperDelegate< T, RealPrecision >
```

A linear taper conforming to [AAX_ITaperDelegate](#).

This taper spaces a parameter's real values evenly between its minimum and maximum, with a linear mapping between the parameter's real and normalized values.

RealPrecision

In addition to its type templatization, this taper includes a precision template parameter. RealPrecision is a multiplier that works in conjunction with the round() function to limit the precision of the real values provided by this taper. For example, if RealPrecision is 1000, it will round to the closest 0.001 when doing any sort of value conversion. If RealPrecision is 1, it will round to the nearest integer. If RealPrecision is 1000000, it will round to the nearest 0.000001. This is particularly useful for preventing things like 1.9999999 truncating down to 1 instead of rounding up to 2.

To accomplish this behavior, the taper multiplies its unrounded parameter values by RealPrecision, rounds the result to the nearest valid value, then divides RealPrecision back out.

Rounding will be disabled if RealPrecision is set to a value less than 1. This is the default.

Public Member Functions

- [AAX_CLinearTaperDelegate \(T minValue=0, T maxValue=1\)](#)
Constructs a Linear Taper with specified minimum and maximum values.
- virtual [AAX_CLinearTaperDelegate< T, RealPrecision > * Clone \(\) const AAX_OVERRIDE](#)
Constructs and returns a copy of the taper delegate.
- virtual T [GetMinimumValue \(\) const AAX_OVERRIDE](#)
Returns the taper's minimum real value.
- virtual T [GetMaximumValue \(\) const AAX_OVERRIDE](#)
Returns the taper's maximum real value.
- virtual T [ConstrainRealValue \(T value\) const AAX_OVERRIDE](#)

- virtual T [NormalizedToReal](#) (double normalizedValue) const **AAX_OVERRIDE**
Converts a normalized value to a real value.
- virtual double [RealToNormalized](#) (T realValue) const **AAX_OVERRIDE**
Normalizes a real parameter value.

Protected Member Functions

- T [Round](#) (double iValue) const

14.15.2 Constructor & Destructor Documentation

14.15.2.1 `template<typename T, int32_t RealPrecision> AAX_CLinearTaperDelegate< T, RealPrecision >::AAX_CLinearTaperDelegate (T minValue = 0, T maxValue = 1)`

Constructs a Linear Taper with specified minimum and maximum values.

Note

The parameter's default value should lie within the min to max range.

Parameters

in	<i>minValue</i>
in	<i>maxValue</i>

14.15.3 Member Function Documentation

14.15.3.1 `template<typename T, int32_t RealPrecision> AAX_CLinearTaperDelegate< T, RealPrecision > * AAX_CLinearTaperDelegate< T, RealPrecision >::Clone () const [virtual]`

Constructs and returns a copy of the taper delegate.

In general, this method's implementation can use a simple copy constructor:

```
template <typename T>
AAX_CSubclassTaperDelegate<T>* AAX_CSubclassTaperDelegate<T>::Clone() const
{
    return new AAX_CSubclassTaperDelegate(*this);
}
```

Implements [AAX_ITaperDelegate< T >](#).

14.15.3.2 `template<typename T, int32_t RealPrecision = 0> virtual T AAX_CLinearTaperDelegate< T, RealPrecision >::GetMinimumValue () const [inline], [virtual]`

Returns the taper's minimum real value.

Implements [AAX_ITaperDelegate< T >](#).

14.15.3.3 `template<typename T, int32_t RealPrecision = 0> virtual T AAX_CLinearTaperDelegate< T, RealPrecision >::GetMaximumValue () const [inline], [virtual]`

Returns the taper's maximum real value.

Implements [AAX_ITaperDelegate< T >](#).

```
14.15.3.4 template<typename T , int32_t RealPrecision> T AAX_CLinearTaperDelegate< T, RealPrecision
>::ConstrainRealValue ( T value ) const [virtual]
```

Applies a constraint to the value and returns the constrained value.

This method is useful if the taper requires a constraint beyond simple minimum and maximum real value limits.

Note

This is the function that should actually enforces the constraints in `NormalizeToReal()` and [RealToNormalized\(\)](#).

Parameters

in	<i>value</i>	The unconstrained value
----	--------------	-------------------------

Implements [AAX_ITaperDelegate< T >](#).

```
14.15.3.5 template<typename T , int32_t RealPrecision> T AAX_CLinearTaperDelegate< T, RealPrecision
>::NormalizedToReal ( double normalizedValue ) const [virtual]
```

Converts a normalized value to a real value.

This is where the actual taper algorithm is implemented.

This function should perform the exact inverse of [RealToNormalized\(\)](#), to within the roundoff precision of the individual taper implementation.

Parameters

in	<i>normalizedValue</i>	The normalized value that will be converted
----	------------------------	---

Implements [AAX_ITaperDelegate< T >](#).

```
14.15.3.6 template<typename T , int32_t RealPrecision> double AAX_CLinearTaperDelegate< T, RealPrecision
>::RealToNormalized ( T realValue ) const [virtual]
```

Normalizes a real parameter value.

This is where the actual taper algorithm is implemented.

This function should perform the exact inverse of [NormalizedToReal\(\)](#), to within the roundoff precision of the individual taper implementation.

Parameters

in	<i>realValue</i>	The real parameter value that will be normalized
----	------------------	--

Implements [AAX_ITaperDelegate< T >](#).

```
14.15.3.7 template<typename T , int32_t RealPrecision> T AAX_CLinearTaperDelegate< T, RealPrecision >::Round (
double iValue ) const [protected]
```

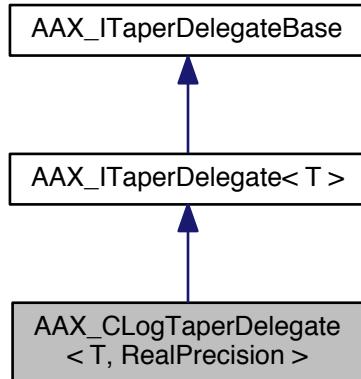
The documentation for this class was generated from the following file:

- [AAX_CLinearTaperDelegate.h](#)

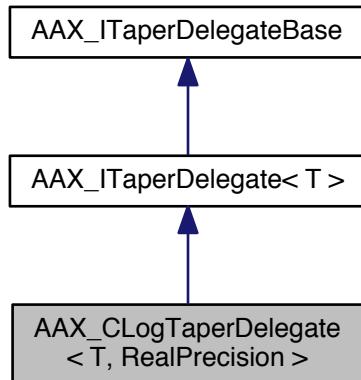
14.16 AAX_CLogTaperDelegate< T, RealPrecision > Class Template Reference

```
#include <AAX_CLogTaperDelegate.h>
```

Inheritance diagram for AAX_CLogTaperDelegate< T, RealPrecision >:



Collaboration diagram for AAX_CLogTaperDelegate< T, RealPrecision >:



14.16.1 Description

```
template<typename T, int32_t RealPrecision = 1000>class AAX_CLogTaperDelegate< T, RealPrecision >
```

A logarithmic taper conforming to [AAX_ITaperDelegate](#).

This taper spaces a parameter's real values between its minimum and maximum bounds, with a natural logarithmic mapping between the parameter's real and normalized values.

RealPrecision

In addition to its type templatization, this taper includes a precision template parameter. RealPrecision is a multiplier that works in conjunction with the round() function to limit the precision of the real values provided by

this taper. For example, if RealPrecision is 1000, it will round to the closest 0.001 when doing any sort of value conversion. If RealPrecision is 1, it will round to the nearest integer. If RealPrecision is 1000000, it will round to the nearest 0.000001. This is particularly useful for preventing things like 1.9999999 truncating down to 1 instead of rounding up to 2.

To accomplish this behavior, the taper multiplies its unrounded parameter values by RealPrecision, rounds the result to the nearest valid value, then divides RealPrecision back out.

Rounding will be disabled if RealPrecision is set to a value less than 1

Public Member Functions

- [AAX_CLogTaperDelegate \(T minValue=0, T maxValue=1\)](#)
Constructs a Log Taper with specified minimum and maximum values.
- [virtual AAX_CLogTaperDelegate< T, RealPrecision > * Clone \(\) const AAX_OVERRIDE](#)
Constructs and returns a copy of the taper delegate.
- [virtual T GetMinimumValue \(\) const AAX_OVERRIDE](#)
Returns the taper's minimum real value.
- [virtual T GetMaximumValue \(\) const AAX_OVERRIDE](#)
Returns the taper's maximum real value.
- [virtual T ConstrainRealValue \(T value\) const AAX_OVERRIDE](#)
Applies a constraint to the value and returns the constrained value.
- [virtual T NormalizedToReal \(double normalizedValue\) const AAX_OVERRIDE](#)
Converts a normalized value to a real value.
- [virtual double RealToNormalized \(T realValue\) const AAX_OVERRIDE](#)
Normalizes a real parameter value.

Protected Member Functions

- [T Round \(double iValue\) const](#)

14.16.2 Constructor & Destructor Documentation

14.16.2.1 `template<typename T , int32_t RealPrecision> AAX_CLogTaperDelegate< T, RealPrecision >::AAX_CLogTaperDelegate (T minValue = 0, T maxValue = 1)`

Constructs a Log Taper with specified minimum and maximum values.

Note

The parameter's default value should lie within the min to max range.

Parameters

in	<i>minValue</i>
in	<i>maxValue</i>

14.16.3 Member Function Documentation

14.16.3.1 `template<typename T , int32_t RealPrecision> AAX_CLogTaperDelegate< T, RealPrecision > * AAX_CLogTaperDelegate< T, RealPrecision >::Clone () const [virtual]`

Constructs and returns a copy of the taper delegate.

In general, this method's implementation can use a simple copy constructor:

```
template <typename T>
AAX_CSubclassTaperDelegate<T>* AAX_CSubclassTaperDelegate<T>::Clone() const
{
    return new AAX_CSubclassTaperDelegate(*this);
}
```

Implements [AAX_ITaperDelegate< T >](#).

14.16.3.2 template<typename T, int32_t RealPrecision = 1000> virtual T AAX_CLogTaperDelegate< T, RealPrecision >::GetMinimumValue() const [inline], [virtual]

Returns the taper's minimum real value.

Implements [AAX_ITaperDelegate< T >](#).

14.16.3.3 template<typename T, int32_t RealPrecision = 1000> virtual T AAX_CLogTaperDelegate< T, RealPrecision >::GetMaximumValue() const [inline], [virtual]

Returns the taper's maximum real value.

Implements [AAX_ITaperDelegate< T >](#).

14.16.3.4 template<typename T, int32_t RealPrecision> T AAX_CLogTaperDelegate< T, RealPrecision >::ConstrainRealValue(T value) const [virtual]

Applies a constraint to the value and returns the constrained value.

This method is useful if the taper requires a constraint beyond simple minimum and maximum real value limits.

Note

This is the function that should actually enforces the constraints in `NormalizeToReal()` and [RealToNormalized\(\)](#).

Parameters

in	value	The unconstrained value
----	-------	-------------------------

Implements [AAX_ITaperDelegate< T >](#).

14.16.3.5 template<typename T, int32_t RealPrecision> T AAX_CLogTaperDelegate< T, RealPrecision >::NormalizedToReal(double normalizedValue) const [virtual]

Converts a normalized value to a real value.

This is where the actual taper algorithm is implemented.

This function should perform the exact inverse of [RealToNormalized\(\)](#), to within the roundoff precision of the individual taper implementation.

Parameters

in	normalizedValue	The normalized value that will be converted
----	-----------------	---

Implements [AAX_ITaperDelegate< T >](#).

References AAX::SafeLog().

Here is the call graph for this function:



14.16.3.6 template<typename T , int32_t RealPrecision> double AAX_CLogTaperDelegate< T, RealPrecision >::RealToNormalized (T realValue) const [virtual]

Normalizes a real parameter value.

This is where the actual taper algorithm is implemented.

This function should perform the exact inverse of [NormalizedToReal\(\)](#), to within the roundoff precision of the individual taper implementation.

Parameters

in	realValue	The real parameter value that will be normalized
----	-----------	--

Implements [AAX_ITaperDelegate< T >](#).

References [AAX::SafeLog\(\)](#).

Here is the call graph for this function:



14.16.3.7 template<typename T , int32_t RealPrecision> T AAX_CLogTaperDelegate< T, RealPrecision >::Round (double iValue) const [protected]

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

- [AAX_CLogTaperDelegate.h](#)

14.17 AAX_CMidiPacket Struct Reference

```
#include <AAX.h>
```

14.17.1 Description

Packet structure for MIDI data.

See also

[AAX_CMidiStream](#)

Legacy Porting Notes Corresponds to DirectMidiPacket in the legacy SDK

Public Attributes

- `uint32_t mTimestamp`

This is the playback time at which the MIDI event should occur, relative to the beginning of the current audio buffer.

- `uint32_t mLength`

The length of MIDI message, in terms of bytes.

- `unsigned char mData [4]`

The MIDI message itself. Each array element is one byte of the message, with the 0th element being the first byte.

- `AAX_CBoolean mIsImmediate`

Indicates that the message is to be sent as soon as possible.

14.17.2 Member Data Documentation

14.17.2.1 `uint32_t AAX_CMidiPacket::mTimestamp`

This is the playback time at which the MIDI event should occur, relative to the beginning of the current audio buffer.

14.17.2.2 `uint32_t AAX_CMidiPacket::mLength`

The length of MIDI message, in terms of bytes.

14.17.2.3 `unsigned char AAX_CMidiPacket::mData[4]`

The MIDI message itself. Each array element is one byte of the message, with the 0th element being the first byte.

Referenced by `AAX::IsAccentedClick()`, `AAX::IsAllNotesOff()`, `AAX::IsNoteOff()`, `AAX::IsNoteOn()`, and `AAX::IsUnaccentedClick()`.

14.17.2.4 `AAX_CBoolean AAX_CMidiPacket::mIsImmediate`

Indicates that the message is to be sent as soon as possible.

Host Compatibility Notes This value is not currently set. Use `mTimestamp == 0` to detect immediate packets

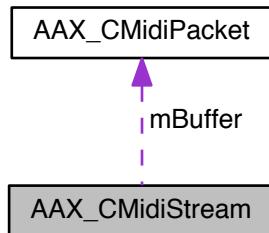
The documentation for this struct was generated from the following file:

- [AAX.h](#)

14.18 AAX_CMidiStream Struct Reference

```
#include <AAx.h>
```

Collaboration diagram for AAX_CMidiStream:



14.18.1 Description

MIDI stream data structure used by [AAX_IMIDINode](#).

For [MIDI input](#), mBufferSize is set by the AAX host when the buffer is filled.

For [MIDI output](#), the plug-in sets mBufferSize with the number of [AAX_CMidiPacket](#) objects it has filled mBuffer with. The AAX host will reset mBufferSize to 0 after it has received the buffer of MIDI.

System Exclusive (SysEx) messages that are greater than 4 bytes in length can be transmitted via a series of concurrent [AAX_CMidiPacket](#) objects in mBuffer. In accordance with the MIDI Specification, `0xF0` indicates the beginning of a SysEx message and `0xF7` indicates its end.

Legacy Porting Notes Corresponds to DirectMidiNode in the legacy SDK

Public Attributes

- `uint32_t mBufferSize`
The number of [AAX_CMidiPacket](#) objects contained in the node's buffer.
- `AAX_CMidiPacket * mBuffer`
Pointer to the first element of the node's buffer.

14.18.2 Member Data Documentation

14.18.2.1 `uint32_t AAX_CMidiStream::mBufferSize`

The number of [AAX_CMidiPacket](#) objects contained in the node's buffer.

14.18.2.2 `AAX_CMidiPacket* AAX_CMidiStream::mBuffer`

Pointer to the first element of the node's buffer.

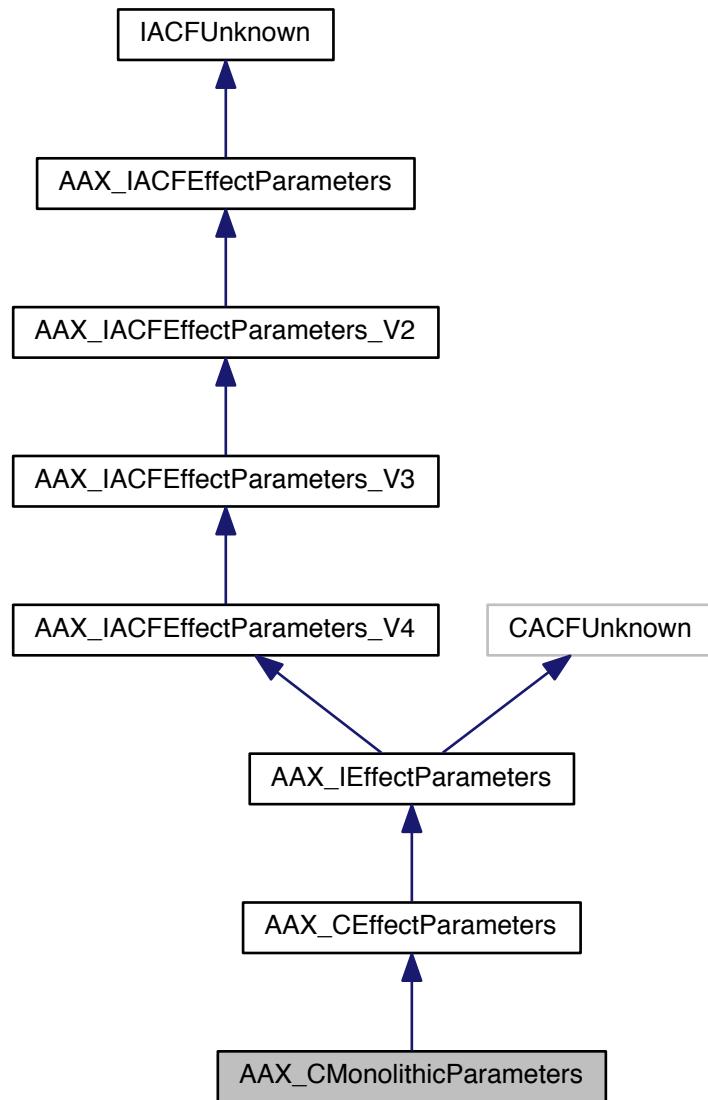
The documentation for this struct was generated from the following file:

- [AAx.h](#)

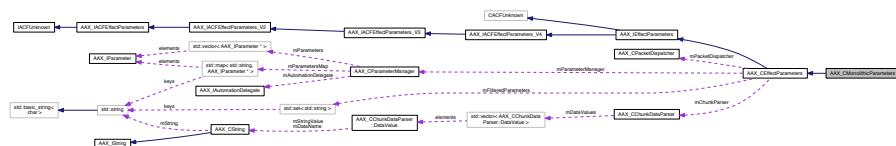
14.19 AAX_CMonolithicParameters Class Reference

```
#include <AAX_CMonolithicParameters.h>
```

Inheritance diagram for AAX_CMonolithicParameters:



Collaboration diagram for AAX_CMonolithicParameters:



14.19.1 Description

Extension of the [AAX_CEffectParameters](#) class for monolithic VIs and effects.

This extension to [AAX_CEffectParameters](#) adds some conveniences for Virtual Instrument (VI) plug-ins and for other plug-ins that use a monolithic processing object, i.e. an object that combines state data with the audio render routine in a single object.

- The [RenderAudio](#) method provides a direct audio processing callback within the data model object. Perform all audio processing in this method.
- The [StaticDescribe](#) method establishes a generic MIDI processing context for the Effect. Call this method from the plug-in's [Description callback](#) implementation.
- The [AddSynchronizedParameter](#) method provides a mechanism for synchronizing parameter updates with the real-time thread, allowing deterministic, accurate automation playback. For more information about this feature, see [Fixing timing issues due to shared data](#)

Note

This convenience class assumes a monolithic processing environment (i.e. [AAX_eConstraintLocationMask_DataModel](#).) This precludes the use of [AAX_CMonolithicParameters](#)-derived Effects in distributed-processing formats such as AAX DSP.

Public Member Functions

- [AAX_CMonolithicParameters](#) (void)
- virtual [~AAX_CMonolithicParameters](#) (void)

Protected Types

- `typedef std::pair< AAX_CParamID const, const AAX_IParameterValue * > TParamValPair`

Protected Member Functions

Real-time functions

Virtual functions called on the real-time thread

- virtual void [RenderAudio](#) ([AAX_SInstrumentRenderInfo](#) *ioRenderInfo, const [TParamValPair](#) *inSynchronizedParamValues[], int32_t inNumSynchronizedParamValues)

Configuration methods

- void [AddSynchronizedParameter](#) (const [AAX_IParameter](#) &inParameter)

Convenience Layer Methods

Note

You should not need to override these methods, but if you do, make sure to call into the base class.

- virtual [AAX_Result UpdateParameterNormalizedValue](#) ([AAX_CParamID](#) iParamID, double aValue, [AAX_EUpdateSource](#) inSource)
Updates a single parameter's state to its current value.
- virtual [AAX_Result GenerateCoefficients](#) ()

- Generates and dispatches new coefficient packets.*
- virtual `AAX_Result ResetFieldData (AAX_CFieldIndex iFieldIndex, void *oData, uint32_t iDataSize) const`
Called by the host to reset a private data field in the plug-in's algorithm.
 - virtual `AAX_Result TimerWakeUp ()`
Periodic wakeup callback for idle-time operations.
 - static `AAX_Result StaticDescribe (AAX_IEffectDescriptor *ioDescriptor, const AAX_SInstrumentSetupInfo &setupInfo)`
 - static void `AAX_CALLBACK StaticRenderAudio (AAX_SInstrumentRenderInfo *const inInstancesBegin[], const void *inInstancesEnd)`

Additional Inherited Members

14.19.2 Member Typedef Documentation

14.19.2.1 `typedef std::pair<AAX_CParamID const, const AAX_IParameterValue*>`
`AAX_CMonolithicParameters::TPairValPair [protected]`

14.19.3 Constructor & Destructor Documentation

14.19.3.1 `AAX_CMonolithicParameters::AAX_CMonolithicParameters (void)`

14.19.3.2 `AAX_CMonolithicParameters::~AAX_CMonolithicParameters (void) [virtual]`

14.19.4 Member Function Documentation

14.19.4.1 `virtual void AAX_CMonolithicParameters::RenderAudio (AAX_SInstrumentRenderInfo * ioRenderInfo, const TParamValPair * inSynchronizedParamValues[], int32_t inNumSynchronizedParamValues) [inline], [protected], [virtual]`

Perform audio render

Parameters

<code>in, out</code>	<code>ioRenderInfo</code>	State data for the current render buffer
<code>in</code>	<code>in↔ Synchronized↔ ParamValues</code>	The parameter values which should be applied for the current render buffer

See also

[AddSynchronizedParameter](#)

Parameters

<code>in</code>	<code>inNum↔ Synchronized↔ ParamValues</code>	The number of parameter values provided in <code>inSynchronizedParamValues</code>
-----------------	---	---

Referenced by `StaticRenderAudio()`.

Here is the caller graph for this function:



14.19.4.2 void AAX_CMonolithicParameters::AddSynchronizedParameter (const AAX_IParameter & *inParameter*)
 [protected]

Add a parameter for state synchronization

A parameter should be added for synchronization if:

- It is important for the parameter's automation to be applied at the correct point on the timeline
- It is possible to quickly update the plug-in's state to reflect a parameter change

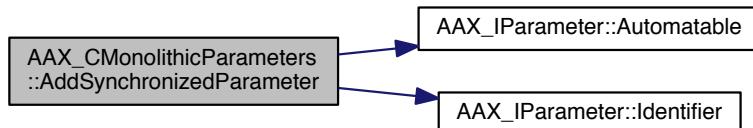
See [Parameter update timing](#) for more information

Parameters

in	<i>inParameter</i>	The parameter to be synchronized. This string will be copied internally and is not required to persist
----	--------------------	--

References AAX_ASSERT, AAX_IParameter::Automatable(), AAX_IParameter::Identifier(), and kSynchronizedParameterQueueSize.

Here is the call graph for this function:



14.19.4.3 AAX_Result AAX_CMonolithicParameters::UpdateParameterNormalizedValue (AAX_CParamID *iParameterID*, double *iValue*, AAX_EUpdateSource *iSource*) [virtual]

Updates a single parameter's state to its current value.

Note

Do *not* call this method from the plug-in. This method should be called by the host only. To set parameter values from within the plug-in, use the [AAX_IParameter](#) interface.

Todo FLAGGED FOR CONSIDERATION OF REVISION

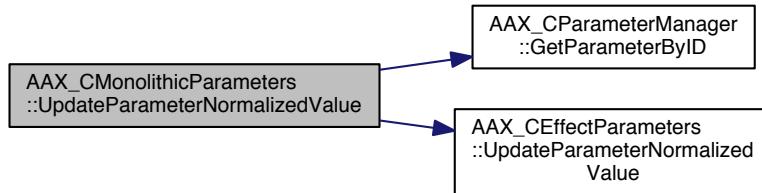
Parameters

in	<i>iParameterID</i>	The ID of the parameter that is being updated
in	<i>iValue</i>	The parameter's current value, to which its internal state must be updated
in	<i>iSource</i>	The source of the update

Reimplemented from [AAX_CEffectParameters](#).

References AAX_SUCCESS, AAX_CParameterManager::GetParameterByID(), AAX_CEffectParameters::mParameterManager, and AAX_CEffectParameters::UpdateParameterNormalizedValue().

Here is the call graph for this function:



14.19.4.4 AAX_Result AAX_CMonolithicParameters::GenerateCoefficients() [virtual]

Generates and dispatches new coefficient packets.

This method is responsible for updating the coefficient packets associated with all parameters whose states have changed since the last call to [GenerateCoefficients\(\)](#). The host may call this method once for every parameter update, or it may "batch" parameter updates such that changes for several parameters are all handled by a single call to [GenerateCoefficients\(\)](#).

For more information on tracking parameters' statuses using the [AAX_CPacketDispatcher](#), helper class, see [AA←X_CPacketDispatcher::SetDirty\(\)](#).

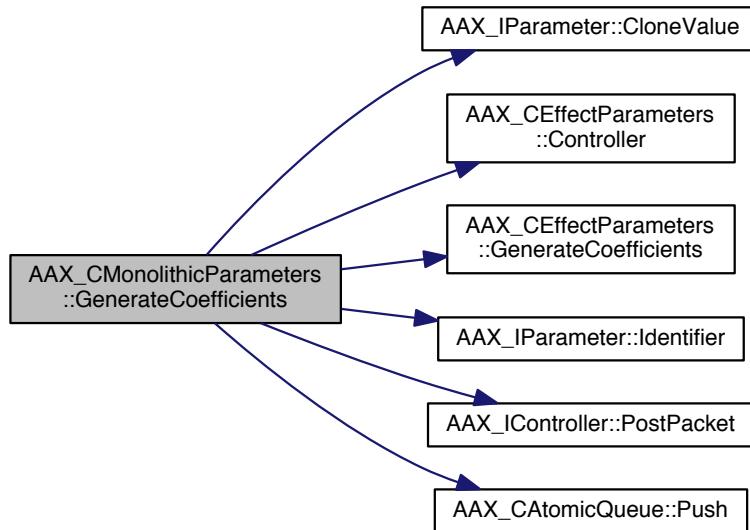
Note

Do not call this method from the plug-in. This method should be called by the host only. To set parameter values from within the plug-in, use the [AAX_IParameter](#) interface.

Reimplemented from [AAX_CEffectParameters](#).

References [AAX_ASSERT](#), [AAX_FIELD_INDEX](#), [AAX_SUCCESS](#), [AAX_IParameter::CloneValue\(\)](#), [AA←CEffectParameters::Controller\(\)](#), [AAX.IContainer::eStatus_Success](#), [AAX_CEffectParameters::Generate←Coefficients\(\)](#), [AAX_IParameter::Identifier\(\)](#), [AAX_IController::PostPacket\(\)](#), and [AAX_CAtomicQueue< T, S >::Push\(\)](#).

Here is the call graph for this function:



14.19.4.5 AAX_Result AAX_CMonolithicParameters::ResetFieldData (*AAX_CFieldIndex* *inFieldIndex*, *void* * *oData*, *uint32_t* *inDataSize*) const [virtual]

Called by the host to reset a private data field in the plug-in's algorithm.

This method is called sequentially for all private data fields on Effect initialization and during any "reset" event, such as priming for a non-real-time render. This method is called before the algorithm's optional initialization callback, and the initialized private data will be available within that callback via its context block.

See also

[Algorithm initialization](#).

Warning

Any data structures that will be passed between platforms (for example, sent to a TI DSP in an AAX DSP plug-in) must be properly data-aligned for compatibility across both platforms. See [AAX_ALIGN_FILE_ALIGNMENT](#) for more information about guaranteeing cross-platform compatibility of data structures used for algorithm processing.

Parameters

in	<i>inFieldIndex</i>	The index of the field that is being initialized
out	<i>oData</i>	The pre-allocated block of data that should be initialized
in	<i>inDataSize</i>	The size of the data block, in bytes

Reimplemented from [AAX_CEffectParameters](#).

References [AAX_ASSERT](#), [AAX_FIELD_INDEX](#), [AAX_SUCCESS](#), [AAX_SInstrumentPrivateData::mMonolithicParametersPtr](#), and [AAX_CEffectParameters::ResetFieldData\(\)](#).

Here is the call graph for this function:



14.19.4.6 **AAX_Result AAX_CMonolithicParameters::TimerWakeup() [virtual]**

Periodic wakeup callback for idle-time operations.

This method is called from the host using a non-main thread. In general, it should be driven at approximately one call per 30 ms. However, the wakeup is not guaranteed to be called at any regular interval - for example, it could be held off by a high real-time processing load - and there is no host contract regarding maximum latency between wakeup calls.

This wakeup thread runs continuously and cannot be armed/disarmed or by the plug-in.

Reimplemented from [AAX_CEffectParameters](#).

References [AAX_CEffectParameters::TimerWakeup\(\)](#).

Here is the call graph for this function:



14.19.4.7 **AAX_Result AAX_CMonolithicParameters::StaticDescribe(AAX_IEffectDescriptor * ioDescriptor, const AAX_SInstrumentSetupInfo & setupInfo) [static]**

Static description callback

This method performs all of the basic context setup and pointer passing work

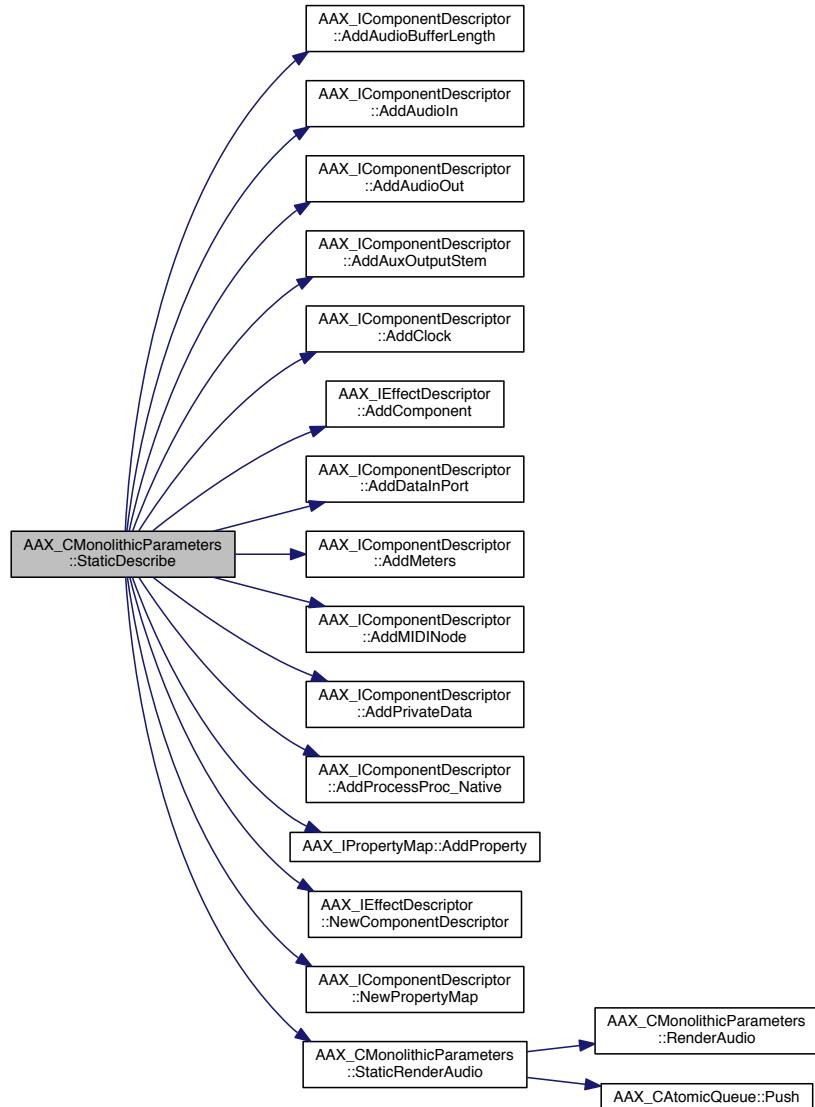
Parameters

in, out	<i>ioDescriptor</i>
in	<i>setupInfo</i>

References [AAX_ASSERT](#), [AAX_eConstraintLocationMask_DataModel](#), [AAX_eMIDINodeType_Global](#), [AAX_eMIDINodeType_LocalInput](#), [AAX_eMIDINodeType_Transport](#), [AAX_ePrivateDataOptions_DefaultOptions](#), [AAX_eProperty_CanBypass](#), [AAX_eProperty_Constraint_Location](#), [AAX_eProperty_Constraint_MultiMonoSupport](#), [AAX_eProperty_HybridInputStemFormat](#), [AAX_eProperty_HybridOutputStemFormat](#), [AAX_eProperty_InputStemFormat](#), [AAX_eProperty_ManufacturerID](#), [AAX_eProperty_OutputStemFormat](#), [AAX_eProperty_PluginID_AudioSuite](#), [AAX_eProperty_PluginID_RTAS](#), [AAX_eProperty_ProductID](#), [AAX_eProperty_UsesClientGUI](#),

AAX_eProperty_UsesTransport, AAX_ERROR_NULL_OBJECT, AAX_eStemFormat_None, AAX_FIELD_IND←
EX, AAX_IComponentDescriptor::AddAudioBufferLength(), AAX_IComponentDescriptor::AddAudioIn(), AAX_I←
ComponentDescriptor::AddAudioOut(), AAX_IComponentDescriptor::AddAuxOutputStem(), AAX_IComponent←
Descriptor::AddClock(), AAX_IEffectDescriptor::AddComponent(), AAX_IComponentDescriptor::AddDataInPort(),
AAX_IComponentDescriptor::AddMeters(), AAX_IComponentDescriptor::AddMIDINode(), AAX_IComponentDescriptor::AddPrivateData(),
AAX_IComponentDescriptor::AddProcessProc_Native(), AAX_IPropertyMap::AddProperty(), kMaxAdditionalMIDINodes,
kMaxAuxOutputStems, AAX_SInstrumentSetupInfo::mAudiosuiteID, AAX_SInstrumentSetupInfo::mAuxOutputStemFormats,
AAX_SInstrumentSetupInfo::mAuxOutputStemNames, AAX_SInstrumentSetupInfo::mCanBypass, AAX_SInstrumentSetupInfo::mGlobalMIDIEventMask,
AAX_SInstrumentSetupInfo::mGlobalMIDINodeName, AAX_SInstrumentSetupInfo::mHybridInputStemFormat,
AAX_SInstrumentSetupInfo::mHybridOutputStemFormat, AAX_SInstrumentSetupInfo::mInputMIDIChannelMask,
AAX_SInstrumentSetupInfo::mInputMIDINodeName, AAX_SInstrumentSetupInfo::mInputStemFormat, AAX_SInstrumentSetupInfo::mManufacturerID,
AAX_SInstrumentSetupInfo::mMeterIDs, AAX_SInstrumentSetupInfo::mMultiMonoSupport, AAX_SInstrumentSetupInfo::mNeedsGlobalMIDI,
AAX_SInstrumentSetupInfo::mNeedsInputMIDI, AAX_SInstrumentSetupInfo::mNeedsTransport,
AAX_SInstrumentSetupInfo::mNumAdditionalInputMIDINodes, AAX_SInstrumentSetupInfo::mNumAuxOutputStems,
AAX_SInstrumentSetupInfo::mNumMeters, AAX_SInstrumentSetupInfo::mOutputStemFormat, AAX_SInstrumentSetupInfo::mPluginID,
AAX_SInstrumentSetupInfo::mProductID, AAX_SInstrumentSetupInfo::mUseHostGeneratedGUI, AAX_IEffectDescriptor::NewComponentDescriptor(),
AAX_IComponentDescriptor::NewPropertyMap(), and StaticRenderAudio().

Here is the call graph for this function:



14.19.4.8 void AAX_CALLBACK AAX_CMonolithicParameters::StaticRenderAudio (AAX_SInstrumentRenderInfo *const *inInstancesBegin*[], const void * *inInstancesEnd*) [static]

Static RenderAudio (Called by the host)

Plug-ins should override [AAX_CMonolithicParameters::RenderAudio\(\)](#)

Parameters

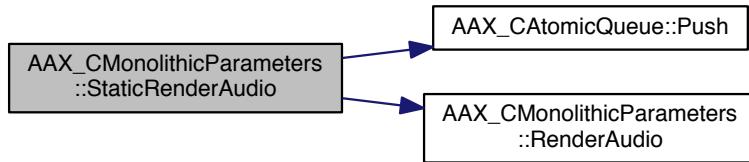
in	<i>inInstancesBegin</i>	
----	-------------------------	--

in	<i>inInstancesEnd</i>
----	-----------------------

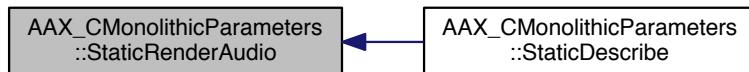
References `AAX_ASSERT`, `AAX_IContainer::eStatus_Success`, `AAX_SInstrumentPrivateData::mMonolithicParametersPtr`, `AAX_CAtomicQueue< T, S >::Push()`, and `RenderAudio()`.

Referenced by `StaticDescribe()`.

Here is the call graph for this function:



Here is the caller graph for this function:



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

- [AAX_CMonolithicParameters.h](#)
- [AAX_CMonolithicParameters.cpp](#)

14.20 AAX_CMutex Class Reference

```
#include <AAX_CMutex.h>
```

14.20.1 Description

Mutex with try lock functionality.

Public Member Functions

- [AAX_CMutex \(\)](#)
- [~AAX_CMutex \(\)](#)
- `bool Lock ()`
- `void Unlock ()`
- `bool Try_Lock ()`

14.20.2 Constructor & Destructor Documentation

14.20.2.1 `AAX_CMutex::AAX_CMutex()`

14.20.2.2 `AAX_CMutex::~AAX_CMutex()`

14.20.3 Member Function Documentation

14.20.3.1 `bool AAX_CMutex::Lock()`

Referenced by `AAX_StLock_Guard::AAX_StLock_Guard()`.

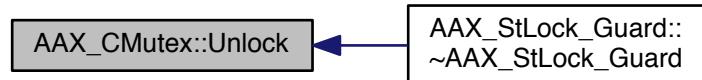
Here is the caller graph for this function:



14.20.3.2 `void AAX_CMutex::Unlock()`

Referenced by `AAX_StLock_Guard::~AAX_StLock_Guard()`.

Here is the caller graph for this function:



14.20.3.3 `bool AAX_CMutex::Try_Lock()`

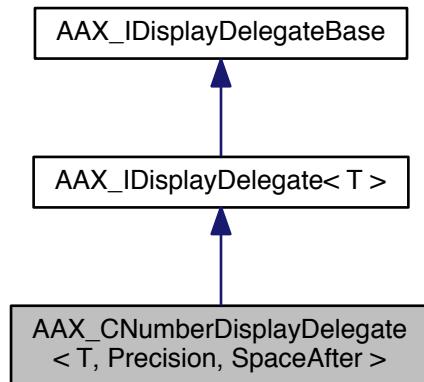
The documentation for this class was generated from the following file:

- [AAX_CMutex.h](#)

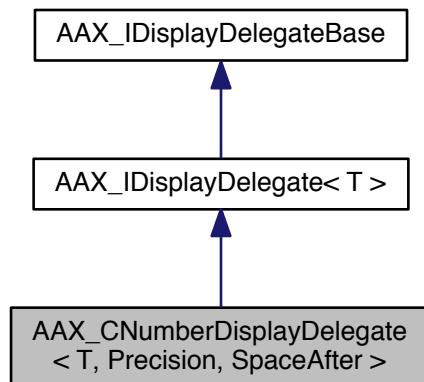
14.21 `AAX_CNumberDisplayDelegate< T, Precision, SpaceAfter >` Class Template Reference

```
#include <AAX_CNumberDisplayDelegate.h>
```

Inheritance diagram for AAX_CNumberDisplayDelegate< T, Precision, SpaceAfter >:



Collaboration diagram for AAX_CNumberDisplayDelegate< T, Precision, SpaceAfter >:



14.21.1 Description

```
template<typename T, uint32_t Precision = 2, uint32_t SpaceAfter = 0>class AAX_CNumberDisplayDelegate< T, Precision, SpaceAfter >
```

A numeric display format conforming to [AAX_IDisplayDelegate](#).

This display delegate converts a parameter value to a numeric string using a specified precision.

Public Member Functions

- virtual [AAX_CNumberDisplayDelegate * Clone \(\) const AAX_OVERRIDE](#)

Constructs and returns a copy of the display delegate.

- virtual bool `ValueToString` (T value, `AAX_CString` *valueString) const `AAX_OVERRIDE`

Converts a real parameter value to a string representation.

- virtual bool `ValueToString` (T value, int32_t maxNumChars, `AAX_CString` *valueString) const `AAX_OVERRIDE`

Converts a real parameter value to a string representation using a size hint, useful for control surfaces and other character limited displays.

- virtual bool `StringToValue` (const `AAX_CString` &valueString, T *value) const `AAX_OVERRIDE`

Converts a string to a real parameter value.

14.21.2 Member Function Documentation

14.21.2.1 template<typename T, uint32_t Precision, uint32_t SpaceAfter> `AAX_CNumberDisplayDelegate`< T, Precision, SpaceAfter > * `AAX_CNumberDisplayDelegate`< T, Precision, SpaceAfter >::Clone() const [virtual]

Constructs and returns a copy of the display delegate.

In general, this method's implementation can use a simple copy constructor:

```
template <typename T>
AAX_CSubclassDisplayDelegate<T>* AAX_CSubclassDisplayDelegate<T>::Clone() const
{
    return new AAX_CSubclassDisplayDelegate(*this);
}
```

Implements `AAX_IDisplayDelegate`< T >.

14.21.2.2 template<typename T, uint32_t Precision, uint32_t SpaceAfter> bool `AAX_CNumberDisplayDelegate`< T, Precision, SpaceAfter >::ValueToString(T value, `AAX_CString` * valueString) const [virtual]

Converts a real parameter value to a string representation.

Parameters

in	<code>value</code>	The real parameter value that will be converted
out	<code>valueString</code>	A string corresponding to value

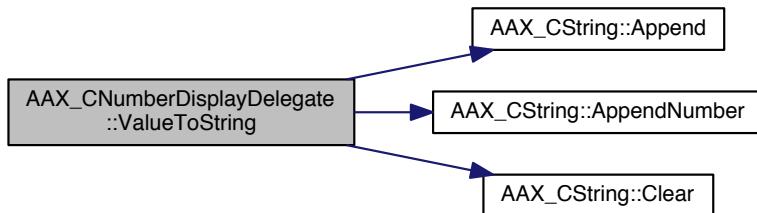
Return values

<code>true</code>	The string conversion was successful
<code>false</code>	The string conversion was unsuccessful

Implements `AAX_IDisplayDelegate`< T >.

References `AAX_CString::Append()`, `AAX_CString::AppendNumber()`, and `AAX_CString::Clear()`.

Here is the call graph for this function:



14.21.2.3 template<typename T, uint32_t Precision, uint32_t SpaceAfter> bool AAX_CNumberDisplayDelegate< T, Precision, SpaceAfter >::ValueToString (T value, int32_t maxNumChars, AAX_CString * valueString) const [virtual]

Converts a real parameter value to a string representation using a size hint, useful for control surfaces and other character limited displays.

Parameters

in	<i>value</i>	The real parameter value that will be converted
in	<i>maxNumChars</i>	Size hint for the desired maximum number of characters in the string (not including null termination)
out	<i>valueString</i>	A string corresponding to <i>value</i>

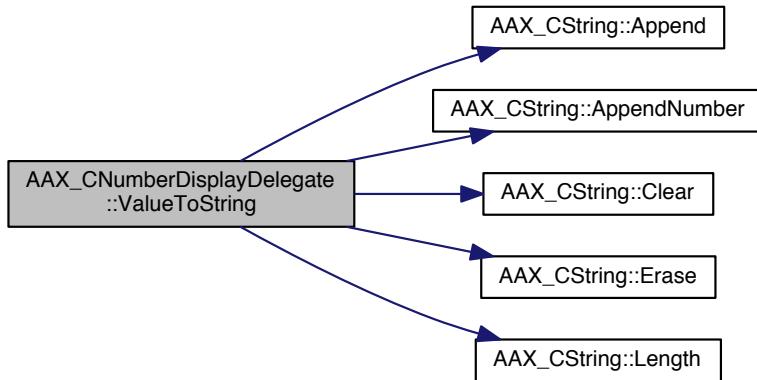
Return values

<i>true</i>	The string conversion was successful
<i>false</i>	The string conversion was unsuccessful

Implements [AAX_IDisplayDelegate< T >](#).

References [AAX_CString::Append\(\)](#), [AAX_CString::AppendNumber\(\)](#), [AAX_CString::Clear\(\)](#), [AAX_CString::Erase\(\)](#), and [AAX_CString::Length\(\)](#).

Here is the call graph for this function:



14.21.2.4 template<typename T , uint32_t Precision, uint32_t SpaceAfter> bool **AAX_CNumberDisplayDelegate< T, Precision, SpaceAfter >::StringToValue** (const **AAX_CString** & *valueString*, *T* * *value*) const [virtual]

Converts a string to a real parameter value.

Parameters

in	<i>valueString</i>	The string that will be converted
out	<i>value</i>	The real parameter value corresponding to <i>valueString</i>

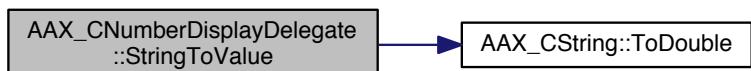
Return values

<i>true</i>	The string conversion was successful
<i>false</i>	The string conversion was unsuccessful

Implements [AAX_IDisplayDelegate< T >](#).

References [AAX_CString::.ToDouble\(\)](#).

Here is the call graph for this function:



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

- [AAX_CNumberDisplayDelegate.h](#)

14.22 AAX_Component< aContextType > Class Template Reference

```
#include <AAx_Callbacks.h>
```

14.22.1 Description

```
template<typename aContextType> class AAX_Component< aContextType >
```

Empty class containing type declarations for the AAX algorithm and associated callbacks.

Public Types

- `typedef const void * inContextPtrsEnd`
- `typedef void * AAX_CALLBACK * CPacketAllocator(const aContextType *inContextPtr, AAX_CFieldIndex inOutputPort, AAX_CTimestamp inTimestamp)`
- `typedef AAX_EComponentInstanceInitAction iAction`
- `typedef void * inNewBlock`
- `typedef void int32_t inSize`
- `typedef void int32_t IACFUnknown *const inController`

Public Member Functions

- `typedef void (AAX_CALLBACK *CProcessProc)(aContextType *const inContextPtrsBegin[])`
- `typedef int32_t (AAX_CALLBACK *CInstanceInitProc)(const aContextType *inInstanceContextPtr)`
- `typedef int32_t (AAX_CALLBACK *CBackgroundProc)(void)`
- `typedef void (AAX_CALLBACK *CInitPrivateDataProc)(AAX_CFieldIndex inFieldIndex)`

14.22.2 Member Typedef Documentation

14.22.2.1 `template<typename aContextType > typedef const void* AAX_Component< aContextType >::inContextPtrsEnd`

14.22.2.2 `template<typename aContextType > typedef void* AAX_CALLBACK* AAX_Component< aContextType >::CPacketAllocator(const aContextType *inContextPtr, AAX_CFieldIndex inOutputPort, AAX_CTimestamp inTimestamp)`

14.22.2.3 `template<typename aContextType > typedef AAX_EComponentInstanceInitAction AAX_Component< aContextType >::iAction`

14.22.2.4 `template<typename aContextType > typedef void* AAX_Component< aContextType >::inNewBlock`

14.22.2.5 `template<typename aContextType > typedef void int32_t AAX_Component< aContextType >::inSize`

14.22.2.6 `template<typename aContextType > typedef void int32_t IACFUnknown* const AAX_Component< aContextType >::inController`

14.22.3 Member Function Documentation

14.22.3.1 `template<typename aContextType > typedef void (AAX_CALLBACK * CProcessProc) const`

14.22.3.2 `template<typename aContextType > typedef AAX_Component< aContextType >::int32_t(AAX_CALLBACK * CInstanceInitProc) const`

14.22.3.3 template<typename aContextType > typedef AAX_Component< aContextType >::int32_t(AAX_CALLBACK * *CBackgroundProc*)

14.22.3.4 template<typename aContextType > typedef AAX_Component< aContextType >::void(AAX_CALLBACK * *CInitPrivateDataProc*)

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

- [AAX_Callbacks.h](#)

14.23 AAX_CPacket Class Reference

```
#include <AAX_CPacketDispatcher.h>
```

14.23.1 Description

Container for packet-related data.

This class collects a number of packet-related data into the same object and provides a facility for tracking when the parameter is "dirty", i.e. after its value has been updated and before an associated packet has not been posted.

Public Member Functions

- [AAX_CPacket \(AAX_CFieldIndex inFieldIndex\)](#)
- [~AAX_CPacket \(\)](#)
- template<typename DataType > [DataType * GetPtr \(\)](#)
- [void SetDirty \(bool iDirty\)](#)
- [bool IsDirty \(\) const](#)
- [AAX_CFieldIndex GetID \(\) const](#)
- [uint32_t GetSize \(\) const](#)
- template<> [const void * GetPtr \(\)](#)

14.23.2 Constructor & Destructor Documentation

14.23.2.1 [AAX_CPacket::AAX_CPacket \(AAX_CFieldIndex *inFieldIndex* \) \[inline\]](#)

14.23.2.2 [AAX_CPacket::~AAX_CPacket \(\) \[inline\]](#)

14.23.3 Member Function Documentation

14.23.3.1 [template<typename DataType > *DataType** AAX_CPacket::GetPtr \(\) \[inline\]](#)

14.23.3.2 [void AAX_CPacket::SetDirty \(bool *iDirty* \) \[inline\]](#)

14.23.3.3 [bool AAX_CPacket::IsDirty \(\) const \[inline\]](#)

14.23.3.4 [AAX_CFieldIndex AAX_CPacket::GetID \(\) const \[inline\]](#)

14.23.3.5 [uint32_t AAX_CPacket::GetSize \(\) const \[inline\]](#)

14.23.3.6 [template<> *const void** AAX_CPacket::GetPtr \(\) \[inline\]](#)

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

- [AAX_CPacketDispatcher.h](#)

14.24 AAX_CPacketDispatcher Class Reference

```
#include <AAX_CPacketDispatcher.h>
```

14.24.1 Description

Helper class for managing AAX packet posting.

This optional class can be used to associate individual parameters with custom update callbacks. The update callbacks for all "dirty" parameters are triggered whenever [AAX_CPacketDispatcher::Dispatch\(\)](#) is called. The resulting coefficient data is then posted to the [AAX_IController](#) automatically by the packet dispatcher.

The packet dispatcher supports many-to-one relationships between parameters and handler callbacks, so a single callback may be registered for several related parameters.

See also

[AAX_CEffectParameters::EffectInit\(\)](#)

Public Member Functions

- [AAX_CPacketDispatcher \(\)](#)
- [~AAX_CPacketDispatcher \(\)](#)
- [void Initialize \(AAX_IController *iPlugIn, AAX_IEffectParameters *iEffectParameters\)](#)
- [AAX_Result RegisterPacket \(AAX_CParamID paramID, AAX_CFieldIndex portID, const AAX_IPacketHandler *iHandler\)](#)
- [template<class TWorker , typename Func > AAX_Result RegisterPacket \(AAX_CParamID paramID, AAX_CFieldIndex portID, TWorker *iPt2Object, Func infPt\)](#)
- [AAX_Result RegisterPacket \(AAX_CParamID paramID, AAX_CFieldIndex portID\)](#)
- [AAX_Result SetDirty \(AAX_CParamID paramID, bool iDirty=true\)](#)
- [AAX_Result Dispatch \(\)](#)
- [AAX_Result GenerateSingleValuePacket \(AAX_CParamID iParam, AAX_CPacket &iOPacket\)](#)

14.24.2 Constructor & Destructor Documentation

14.24.2.1 [AAX_CPacketDispatcher::AAX_CPacketDispatcher \(\)](#)

14.24.2.2 [AAX_CPacketDispatcher::~AAX_CPacketDispatcher \(\)](#)

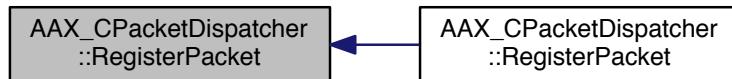
14.24.3 Member Function Documentation

14.24.3.1 [void AAX_CPacketDispatcher::Initialize \(AAX_IController * iPlugIn, AAX_IEffectParameters * iEffectParameters \)](#)

14.24.3.2 [AAX_Result AAX_CPacketDispatcher::RegisterPacket \(AAX_CParamID paramID, AAX_CFieldIndex portID, const AAX_IPacketHandler * iHandler \)](#)

Referenced by [RegisterPacket\(\)](#).

Here is the caller graph for this function:



14.24.3.3 template<class TWorker , typename Func > AAX_Result AAX_CPacketDispatcher::RegisterPacket (AAX_CParamID paramID, AAX_CFieldIndex portID, TWorker * iPt2Object, Func infPt) [inline]

References RegisterPacket().

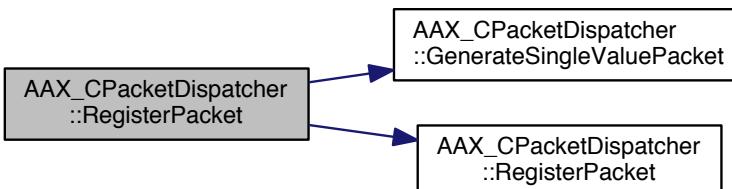
Here is the call graph for this function:



14.24.3.4 AAX_Result AAX_CPacketDispatcher::RegisterPacket (AAX_CParamID paramID, AAX_CFieldIndex portID) [inline]

References GenerateSingleValuePacket(), and RegisterPacket().

Here is the call graph for this function:



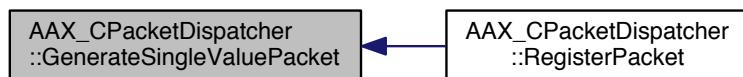
14.24.3.5 AAX_Result AAX_CPacketDispatcher::SetDirty (AAX_CParamID paramID, bool iDirty = true)

14.24.3.6 `AAX_Result AAX_CPacketDispatcher::Dispatch()`

14.24.3.7 `AAX_Result AAX_CPacketDispatcher::GenerateSingleValuePacket(AAX_CParamID iParam, AAX_CPacket & ioPacket)`

Referenced by `RegisterPacket()`.

Here is the caller graph for this function:



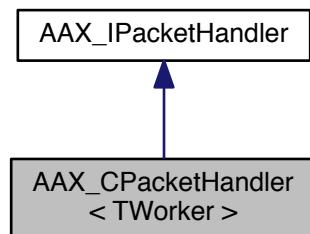
The documentation for this class was generated from the following file:

- [AAX_CPacketDispatcher.h](#)

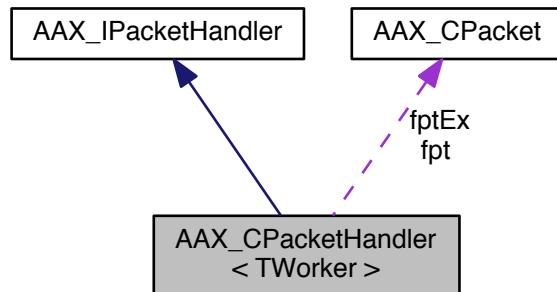
14.25 AAX_CPacketHandler< TWorker > Class Template Reference

```
#include <AAX_CPacketDispatcher.h>
```

Inheritance diagram for `AAX_CPacketHandler< TWorker >`:



Collaboration diagram for AAX_CPacketHandler< TWorker >:



14.25.1 Description

```
template<class TWorker>class AAX_CPacketHandler< TWorker >
```

Callback container used by [AAX_CPacketDispatcher](#).

Public Member Functions

- [AAX_CPacketHandler](#) (TWorker *iPt2Object, fPt2Fn infPt)
- [AAX_CPacketHandler](#) (TWorker *iPt2Object, fPt2FnEx infPt)
- [AAX_IPacketHandler * Clone \(\) const](#)
- [AAX_Result Call](#) (AAX_CParamID inParamID, [AAX_CPacket](#) &ioPacket) const

Protected Attributes

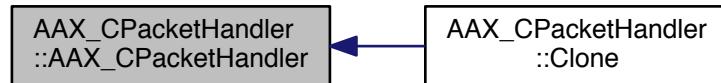
- TWorker * [pt2Object](#)
- fPt2Fn [fpt](#)
- fPt2FnEx [fptEx](#)

14.25.2 Constructor & Destructor Documentation

```
14.25.2.1 template<class TWorker> AAX_CPacketHandler< TWorker >::AAX_CPacketHandler ( TWorker *
iPt2Object, fPt2Fn infPt ) [inline]
```

Referenced by [AAX_CPacketHandler< TWorker >::Clone\(\)](#).

Here is the caller graph for this function:



14.25.2.2 template<class TWorker> AAX_CPacketHandler< TWorker >::AAX_CPacketHandler (TWorker * *iPt2Object*, fPt2FnEx *infPt*) [inline]

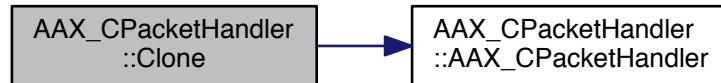
14.25.3 Member Function Documentation

14.25.3.1 template<class TWorker> AAX_IPacketHandler* AAX_CPacketHandler< TWorker >::Clone () const [inline], [virtual]

Implements [AAX_IPacketHandler](#).

References AAX_CPacketHandler< TWorker >::AAX_CPacketHandler().

Here is the call graph for this function:



14.25.3.2 template<class TWorker> AAX_Result AAX_CPacketHandler< TWorker >::Call (AAX_CParamID *inParamID*, AAX_CPacket & *ioPacket*) const [inline], [virtual]

Implements [AAX_IPacketHandler](#).

References AAX_ERROR_NULL_OBJECT, AAX_CPacketHandler< TWorker >::fpt, AAX_CPacketHandler< TWorker >::fptEx, and AAX_CPacketHandler< TWorker >::pt2Object.

14.25.4 Member Data Documentation

14.25.4.1 template<class TWorker> TWorker* AAX_CPacketHandler< TWorker >::pt2Object [protected]

Referenced by AAX_CPacketHandler< TWorker >::Call().

14.25.4.2 template<class TWorker> fPt2Fn AAX_CPacketHandler< TWorker >::fpt [protected]

Referenced by AAX_CPacketHandler< TWorker >::Call().

14.25.4.3 template<class TWorker> fPt2FnEx AAX_CPacketHandler< TWorker >::fptEx [protected]

Referenced by AAX_CPacketHandler< TWorker >::Call().

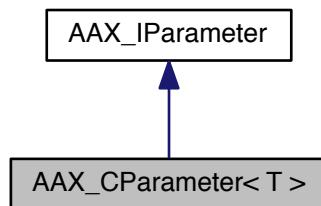
The documentation for this class was generated from the following file:

- [AAX_CPacketDispatcher.h](#)

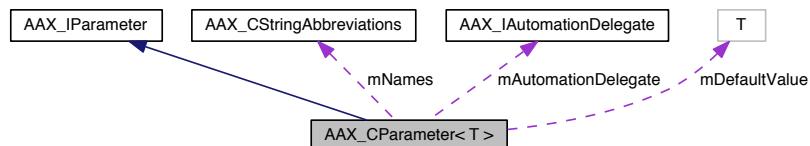
14.26 AAX_CParameter< T > Class Template Reference

#include <AAX_CParameter.h>

Inheritance diagram for AAX_CParameter< T >:



Collaboration diagram for AAX_CParameter< T >:



14.26.1 Description

template<typename T>class AAX_CParameter< T >

Generic implementation of an [AAX_IParameter](#).

This is a concrete, templatized implementation of [AAX_IParameter](#) for parameters with standard types such as float, uint32, bool, etc.

Many different behaviors can be composited into this class as delegates. [AAX_ITaperDelegate](#) and [AAX_IDisplayDelegate](#) are two examples of delegates that this class uses in order to apply custom behaviors to the [AAX_IParameter](#) interface.

Plug-in developers can subclass these delegates to create adaptable, reusable parameter behaviors, which can then be "mixed in" to individual [AAX_CParameter](#) objects without the need to modify the objects themselves.

Note

Because [AAX_CParameter](#) is a C++ template, each [AAX_CParameter](#) template parameter that is used creates a new subclass that adheres to the [AAX_IParameter](#) interface.

Public Types

- enum `Type` { `eParameterTypeUndefined` = 0, `eParameterTypeBool` = 1, `eParameterTypeInt32` = 2, `eParameterTypeFloat` = 3, `eParameterTypeCustom` = 4 }
- enum `Defaults` { `eParameterDefaultNumStepsDiscrete` = 2, `eParameterDefaultNumStepsContinuous` = 128 }

Public Member Functions

- [`AAX_CParameter \(AAX_CParamID identifier, const AAX_IString &name, T defaultValue, const AAX_ITaperDelegate< T > &taperDelegate, const AAX_IDisplayDelegate< T > &displayDelegate, bool automatable=false\)`](#)
Constructs an [AAX_CParameter](#) object using the specified taper and display delegates.
- [`AAX_CParameter \(const AAX_IString &identifier, const AAX_IString &name, T defaultValue, const AAX_ITaperDelegate< T > &taperDelegate, const AAX_IDisplayDelegate< T > &displayDelegate, bool automatable=false\)`](#)
Constructs an [AAX_CParameter](#) object using the specified taper and display delegates.
- [`AAX_CParameter \(const AAX_IString &identifier, const AAX_IString &name, T defaultValue, bool automatable=false\)`](#)
Constructs an [AAX_CParameter](#) object with no delegates.
- [`AAX_CParameter \(const AAX_IString &identifier, const AAX_IString &name, bool automatable=false\)`](#)
Constructs an [AAX_CParameter](#) object with no delegates or default value.
- [`AAX_DEFAULT_MOVECTOR \(AAX_CParameter\)`](#)
- [`AAX_DEFAULT_MOVEOPER \(AAX_CParameter\)`](#)
- [`AAX_DELETE \(AAX_CParameter\(\)\)`](#)
- [`AAX_DELETE \(AAX_CParameter\(const AAX_CParameter &other\)\)`](#)
- [`AAX_DELETE \(AAX_CParameter &operator=\(const AAX_CParameter &other\)\)`](#)
- [`virtual ~AAX_CParameter \(\)`](#)
Virtual destructor used to delete all locally allocated pointers.
- [`virtual AAX_IParameterValue * CloneValue \(\) const AAX_OVERRIDE`](#)
Clone the parameter's value to a new [AAX_IParameterValue](#) object.
- [`template<> bool GetValueAsString \(AAX_IString *\) const`](#)
Retrieves the parameter's value as a string.
- [`template<> bool SetValueWithBool \(bool value\)`](#)
Sets the parameter's value as a bool.
- [`template<> bool SetValueWithInt32 \(int32_t value\)`](#)
Sets the parameter's value as an int32_t.
- [`template<> bool SetValueWithFloat \(float value\)`](#)
Sets the parameter's value as a float.
- [`template<> bool SetValueWithDouble \(double value\)`](#)
Sets the parameter's value as a double.
- [`template<> bool SetValueWithString \(const AAX_IString &value\)`](#)
Sets the parameter's value as a string.

- template<> bool [GetNormalizedValueFromBool](#) (bool value, double *normalizedValue) const
Converts a bool to a normalized parameter value.
 - template<> bool [GetNormalizedValueFromInt32](#) (int32_t value, double *normalizedValue) const
Converts an integer to a normalized parameter value.
 - template<> bool [GetNormalizedValueFromFloat](#) (float value, double *normalizedValue) const
Converts a float to a normalized parameter value.
 - template<> bool [GetNormalizedValueFromDouble](#) (double value, double *normalizedValue) const
Converts a double to a normalized parameter value.
 - template<>
-

- `bool GetBoolFromNormalizedValue (double inNormalizedValue, bool *value) const`
Converts a normalized parameter value to a bool representing the corresponding real value.
- template<> `bool GetInt32FromNormalizedValue (double inNormalizedValue, int32_t *value) const`
Converts a normalized parameter value to an integer representing the corresponding real value.
- template<> `bool GetFloatFromNormalizedValue (double inNormalizedValue, float *value) const`
Converts a normalized parameter value to a float representing the corresponding real value.
- template<> `bool GetDoubleFromNormalizedValue (double inNormalizedValue, double *value) const`
Converts a normalized parameter value to a double representing the corresponding real value.

Identification methods

- `virtual AAX_CParamID Identifier () const AAX_OVERRIDE`
Returns the parameter's unique identifier.
- `virtual void SetName (const AAX_CString &name) AAX_OVERRIDE`
Sets the parameter's display name.
- `virtual const AAX_CString & Name () const AAX_OVERRIDE`
Returns the parameter's display name.
- `virtual void AddShortenedName (const AAX_CString &name) AAX_OVERRIDE`
Sets the parameter's shortened display name.
- `virtual const AAX_CString & ShortenedName (int32_t iNumCharacters) const AAX_OVERRIDE`
Returns the parameter's shortened display name.
- `virtual void ClearShortenedNames () AAX_OVERRIDE`
Clears the internal list of shortened display names.

Taper methods

- `virtual void SetNormalizedDefaultValue (double normalizedDefault) AAX_OVERRIDE`
Sets the parameter's default value using its normalized representation.
- `virtual double GetNormalizedDefaultValue () const AAX_OVERRIDE`
Returns the normalized representation of the parameter's real default value.
- `virtual void SetToDefaultValue () AAX_OVERRIDE`
Restores the state of this parameter to its default value.
- `virtual void SetNormalizedValue (double newNormalizedValue) AAX_OVERRIDE`
Sets a parameter value using it's normalized representation.
- `virtual double GetNormalizedValue () const AAX_OVERRIDE`
Returns the normalized representation of the parameter's current real value.
- `virtual void SetNumberOfSteps (uint32_t numSteps) AAX_OVERRIDE`
Sets the number of discrete steps for this parameter.
- `virtual uint32_t GetNumberOfSteps () const AAX_OVERRIDE`
Returns the number of discrete steps used by the parameter.
- `virtual uint32_t GetStepValue () const AAX_OVERRIDE`
Returns the current step for the current value of the parameter.
- `virtual double GetNormalizedValueFromStep (uint32_t iStep) const AAX_OVERRIDE`
Returns the normalized value for a given step.
- `virtual uint32_t GetStepValueFromNormalizedValue (double normalizedValue) const AAX_OVERRIDE`
Returns the step value for a normalized value of the parameter.
- `virtual void SetStepValue (uint32_t iStep) AAX_OVERRIDE`
Returns the current step for the current value of the parameter.
- `virtual void SetType (AAX_EParameterType iControlType) AAX_OVERRIDE`
Sets the type of this parameter.
- `virtual AAX_EParameterType GetType () const AAX_OVERRIDE`
Returns the type of this parameter as an AAX_EParameterType.
- `virtual void SetOrientation (AAX_EParameterOrientation iOrientation) AAX_OVERRIDE`
Sets the orientation of this parameter.
- `virtual AAX_EParameterOrientation GetOrientation () const AAX_OVERRIDE`
Returns the orientation of this parameter.
- `virtual void SetTaperDelegate (AAX_ITaperDelegateBase &inTaperDelegate, bool inPreserveValue=true) AAX_OVERRIDE`

Sets the parameter's taper delegate.

Display methods

- virtual void `SetDisplayDelegate (AAAX_IDisplayDelegateBase &inDisplayDelegate)` **AAX_OVERRIDE**
Sets the parameter's display delegate.
- virtual bool `GetValueString (AAAX_CString *valueString) const` **AAX_OVERRIDE**
Serializes the parameter value into a string.
- virtual bool `GetValueString (int32_t iMaxNumChars, AAAX_CString *valueString) const` **AAX_OVERRIDE**
Serializes the parameter value into a string, size hint included.
- virtual bool `GetNormalizedValueFromBool (bool value, double *normalizedValue) const` **AAX_OVERRIDE**
Converts a bool to a normalized parameter value.
- virtual bool `GetNormalizedValueFromInt32 (int32_t value, double *normalizedValue) const` **AAX_OVERRIDE**
Converts an integer to a normalized parameter value.
- virtual bool `GetNormalizedValueFromFloat (float value, double *normalizedValue) const` **AAX_OVERRIDE**
Converts a float to a normalized parameter value.
- virtual bool `GetNormalizedValueFromDouble (double value, double *normalizedValue) const` **AAX_OVERRIDE**
Converts a double to a normalized parameter value.
- virtual bool `GetNormalizedValueFromString (const AAAX_CString &valueString, double *normalizedValue) const` **AAX_OVERRIDE**
Converts a given string to a normalized parameter value.
- virtual bool `GetBoolFromNormalizedValue (double normalizedValue, bool *value) const` **AAX_OVERRIDE**
Converts a normalized parameter value to a bool representing the corresponding real value.
- virtual bool `GetInt32FromNormalizedValue (double normalizedValue, int32_t *value) const` **AAX_OVERRIDE**
Converts a normalized parameter value to an integer representing the corresponding real value.
- virtual bool `GetFloatFromNormalizedValue (double normalizedValue, float *value) const` **AAX_OVERRIDE**
Converts a normalized parameter value to a float representing the corresponding real value.
- virtual bool `GetDoubleFromNormalizedValue (double normalizedValue, double *value) const` **AAX_OVERRIDE**
Converts a normalized parameter value to a double representing the corresponding real value.
- virtual bool `GetStringFromNormalizedValue (double normalizedValue, AAAX_CString &valueString) const` **AAX_OVERRIDE**
Converts a normalized parameter value to a string representing the corresponding real value.
- virtual bool `GetStringFromNormalizedValue (double normalizedValue, int32_t iMaxNumChars, AAAX_CString &valueString) const` **AAX_OVERRIDE**
Converts a normalized parameter value to a string representing the corresponding real, size hint included. value.
- virtual bool `SetValueFromString (const AAAX_CString &newValueString)` **AAX_OVERRIDE**
Converts a string to a real parameter value and sets the parameter to this value.

Automation methods

- virtual void `SetAutomationDelegate (AAAX_IAutomationDelegate *iAutomationDelegate)` **AAX_OVERRIDE**
Sets the automation delegate (if one is required)
- virtual bool `Automatable () const` **AAX_OVERRIDE**
Returns true if the parameter is automatable, false if it is not.
- virtual void `Touch ()` **AAX_OVERRIDE**
Signals the automation system that a control has been touched.
- virtual void `Release ()` **AAX_OVERRIDE**
Signals the automation system that a control has been released.

Typed accessors

- virtual bool `GetValueAsBool (bool *value) const` **AAX_OVERRIDE**

- `virtual bool GetValueAsInt32 (int32_t *value) const AAX_OVERRIDE`
Retrieves the parameter's value as a bool.
- `virtual bool GetValueAsFloat (float *value) const AAX_OVERRIDE`
Retrieves the parameter's value as an int32_t.
- `virtual bool GetValueAsDouble (double *value) const AAX_OVERRIDE`
Retrieves the parameter's value as a float.
- `virtual bool GetValueAsString (AAX_IString *value) const AAX_OVERRIDE`
Retrieves the parameter's value as a string.
- `virtual bool SetValueWithBool (bool value) AAX_OVERRIDE`
Sets the parameter's value as a bool.
- `virtual bool SetValueWithInt32 (int32_t value) AAX_OVERRIDE`
Sets the parameter's value as an int32_t.
- `virtual bool SetValueWithFloat (float value) AAX_OVERRIDE`
Sets the parameter's value as a float.
- `virtual bool SetValueWithDouble (double value) AAX_OVERRIDE`
Sets the parameter's value as a double.
- `virtual bool SetValueWithString (const AAX_IString &value) AAX_OVERRIDE`
Sets the parameter's value as a string.

Host interface methods

- `virtual void UpdateNormalizedValue (double newNormalizedValue) AAX_OVERRIDE`
Sets the parameter's state given a normalized value.

Direct methods on AAX_CParameter

These methods can be used to access the parameter's state and properties. These methods are specific to the concrete `AAX_CParameter` class and are not part of the `AAX_IParameter` interface.

- `void SetValue (T newValue)`
Initiates a host request to set the parameter's value.
- `T GetValue () const`
Returns the parameter's value.
- `void SetDefaultValue (T newValue)`
Set the parameter's default value.
- `T GetDefaultValue () const`
Returns the parameter's default value.
- `const AAX_ITaperDelegate< T > * TaperDelegate () const`
Returns a reference to the parameter's taper delegate.
- `const AAX_IDisplayDelegate< T > * DisplayDelegate () const`
Returns a reference to the parameter's display delegate.

Protected Attributes

- `AAX_CStringAbbreviations mNames`
- `bool mAutomatable`
- `uint32_t mNumSteps`
- `AAX_EParameterType mControlType`
- `AAX_EParameterOrientation mOrientation`
- `AAX_ITaperDelegate< T > * mTaperDelegate`
- `AAX_IDisplayDelegate< T > * mDisplayDelegate`
- `AAX_IAutomationDelegate * mAutomationDelegate`
- `bool mNeedNotify`
- `AAX_CParameterValue< T > mValue`
- `T mDefaultValue`

14.26.2 Constructor & Destructor Documentation

14.26.2.1 template<typename T> AAX_CParameter<T>::AAX_CParameter(AAX_CParamID identifier, const AAX_IString & name, T defaultValue, const AAX_ITaperDelegate<T> & taperDelegate, const AAX_IDisplayDelegate<T> & displayDelegate, bool automatable = false)

Constructs an [AAX_CParameter](#) object using the specified taper and display delegates.

The delegates are passed in by reference to prevent ambiguities of object ownership. For more information about identifier and name, please consult the base [AAX_IParameter](#) interface.

Parameters

in	<i>identifier</i>	Unique ID for the parameter, these can only be 31 characters long at most. (the fixed length is a requirement for some optimizations in the host)
in	<i>name</i>	The parameter's unabbreviated display name
in	<i>defaultValue</i>	The parameter's default value
in	<i>taperDelegate</i>	A delegate representing the parameter's taper behavior
in	<i>displayDelegate</i>	A delegate representing the parameter's display conversion behavior
in	<i>automatable</i>	A flag to set whether the parameter will be visible to the host's automation system

Note

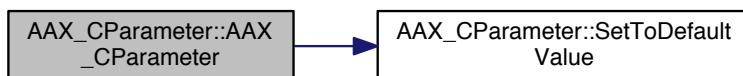
Upon construction, the state (value) of the parameter will be the default value, as established by the provided taperDelegate.

Host Compatibility Notes As of Pro Tools 10.2, DAE will check for a matching parameter NAME and not an ID when reading in automation data from a session saved with an AAX plug-ins RTAS/TDM counter part.

As of Pro Tools 11.1, AAE will first try to match ID. If that fails, AAE will fall back to matching by Name.

References [AAX_CParameter< T >::SetToDefaultValue\(\)](#).

Here is the call graph for this function:



14.26.2.2 template<typename T> AAX_CParameter<T>::AAX_CParameter(const AAX_IString & identifier, const AAX_IString & name, T defaultValue, const AAX_ITaperDelegate<T> & taperDelegate, const AAX_IDisplayDelegate<T> & displayDelegate, bool automatable = false)

Constructs an [AAX_CParameter](#) object using the specified taper and display delegates.

This constructor uses an [AAX_IString](#) for the parameter identifier, which can be a more flexible solution for some plug-ins.

References [AAX_CParameter< T >::SetToDefaultValue\(\)](#).

Here is the call graph for this function:



14.26.2.3 template<typename T> AAX_CParameter<T>::AAX_CParameter (const AAX_IString & identifier, const AAX_IString & name, T defaultValue, bool automatable = false)

Constructs an [AAX_CParameter](#) object with no delegates.

Delegates may be set on this object after construction. Most parameter operations will not work until after delegates have been set.

See also

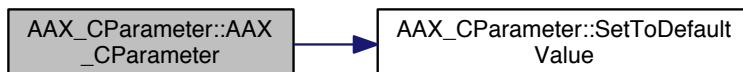
- [AAX_CParameter::SetTaperDelegate\(\)](#)

See also

- [AAX_CParameter::SetDisplayDelegate\(\)](#)

References [AAX_CParameter< T >::SetToDefaultValue\(\)](#).

Here is the call graph for this function:



14.26.2.4 template<typename T> AAX_CParameter<T>::AAX_CParameter (const AAX_IString & identifier, const AAX_IString & name, bool automatable = false)

Constructs an [AAX_CParameter](#) object with no delegates or default value.

Delegates and default value may be set on this object after construction. Most parameter operations will not work until after delegates have been set.

See also

- [AAX_CParameter::SetDefaultValue\(\)](#)

See also

- [AAX_CParameter::SetTaperDelegate\(\)](#)

See also

- [AAX_CParameter::SetDisplayDelegate\(\)](#)

References [AAX_CParameter< T >::SetToDefaultValue\(\)](#).

Here is the call graph for this function:



14.26.2.5 template<typename T> AAX_CParameter< T >::~AAX_CParameter() [virtual]

Virtual destructor used to delete all locally allocated pointers.

14.26.3 Member Function Documentation

14.26.3.1 template<typename T> AAX_CParameter< T >::AAX_DEFAULT_MOVECTOR(AAX_CParameter< T >)

Move constructor and move assignment operator are allowed

14.26.3.2 template<typename T> AAX_CParameter< T >::AAX_DEFAULT_MOVEOPER(AAX_CParameter< T >)

14.26.3.3 template<typename T> AAX_CParameter< T >::AAX_DELETE(AAX_CParameter< T >())

Default constructor not allowed, except by possible wrappering classes.

14.26.3.4 template<typename T> AAX_CParameter< T >::AAX_DELETE(AAX_CParameter< T >(const AAX_CParameter< T > &other))

14.26.3.5 template<typename T> AAX_CParameter< T >::AAX_DELETE(AAX_CParameter< T > &operator=(const AAX_CParameter< T > &other))

14.26.3.6 template<typename T> AAX_IPAttributeValue * AAX_CParameter< T >::CloneValue() const [virtual]

Clone the parameter's value to a new [AAX_IPAttributeValue](#) object.

The returned object is independent from the [AAX_IPParameter](#). For example, changing the state of the returned object will not result in a change to the original [AAX_IPParameter](#).

Implements [AAX_IPParameter](#).

14.26.3.7 template<typename T> AAX_CParamID AAX_CParameter< T >::Identifier() const [virtual]

Returns the parameter's unique identifier.

This unique ID is used by the [Parameter Manager](#) and by outside applications to uniquely identify and target control messages. This value may not be changed after the parameter has been constructed.

Implements [AAX_IParameter](#).

14.26.3.8 template<typename T> void AAX_CParameter< T >::SetName (const AAX_CString & name) [virtual]

Sets the parameter's display name.

This name is used for display only, it is not used for indexing or identifying the parameter This name may be changed after the parameter has been created, but display name changes may not be recognized by all AAX hosts.

Parameters

in	name	Display name that will be assigned to the parameter
----	------	---

Implements [AAX_IParameter](#).

14.26.3.9 template<typename T> const AAX_CString & AAX_CParameter< T >::Name () const [virtual]

Returns the parameter's display name.

Note

This method returns a const reference in order to prevent a string copy. Do not cast away the const to change this value.

Implements [AAX_IParameter](#).

14.26.3.10 template<typename T> void AAX_CParameter< T >::AddShortenedName (const AAX_CString & name) [virtual]

Sets the parameter's shortened display name.

This name is used for display only, it is not used for indexing or identifying the parameter These names show up when the host asks for shorter length parameter names for display on Control Surfaces or other string length constrained situations.

Parameters

in	name	Shortened display names that will be assigned to the parameter
----	------	--

Implements [AAX_IParameter](#).

14.26.3.11 template<typename T> const AAX_CString & AAX_CParameter< T >::ShortenedName (int32_t iNumCharacters) const [virtual]

Returns the parameter's shortened display name.

Note

This method returns a const reference in order to prevent a string copy. Do not cast away the const to change this value.

Implements [AAX_IParameter](#).

References AAX_CString::Get().

Here is the call graph for this function:



14.26.3.12 template<typename T> void AAX_CParameter< T >::ClearShortenedNames() [virtual]

Clears the internal list of shortened display names.

Implements [AAX_IParameter](#).

14.26.3.13 template<typename T> void AAX_CParameter< T >::SetNormalizedDefaultValue(double *normalizedDefault*) [virtual]

Sets the parameter's default value using its normalized representation.

Implements [AAX_IParameter](#).

14.26.3.14 template<typename T> double AAX_CParameter< T >::GetNormalizedDefaultValue() const [virtual]

Returns the normalized representation of the parameter's real default value.

Implements [AAX_IParameter](#).

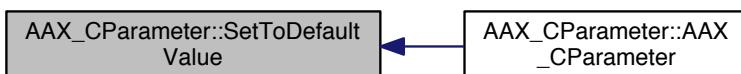
14.26.3.15 template<typename T> void AAX_CParameter< T >::SetToDefaultValue() [virtual]

Restores the state of this parameter to its default value.

Implements [AAX_IParameter](#).

Referenced by [AAX_CParameter< T >::AAX_CParameter\(\)](#).

Here is the caller graph for this function:



14.26.3.16 template<typename T> void AAX_CParameter< T >::SetNormalizedValue(double *newNormalizedValue*) [virtual]

Sets a parameter value using it's normalized representation.

For more information regarding normalized values, see [Parameter Manager](#)

Parameters

in	<i>new← NormalizedValue</i>	New value (normalized) to which the parameter will be set
----	---------------------------------	---

Implements [AAX_IParameter](#).

14.26.3.17 template<typename T> double AAX_CParameter< T >::GetNormalizedValue() const [virtual]

Returns the normalized representation of the parameter's current real value.

Implements [AAX_IParameter](#).

14.26.3.18 template<typename T> void AAX_CParameter< T >::SetNumberOfSteps(uint32_t numSteps) [virtual]

Sets the number of discrete steps for this parameter.

Stepped parameter values are useful for discrete parameters and for "jumping" events such as mouse wheels, page up/down, etc. The parameter's step size is used to specify the coarseness of those changes.

Note

numSteps MUST be greater than zero. All other values may be considered an error by the host.

Parameters

in	<i>numSteps</i>	The number of steps that the parameter will use
----	-----------------	---

Implements [AAX_IParameter](#).

References [AAX_ASSERT](#).

14.26.3.19 template<typename T> uint32_t AAX_CParameter< T >::GetNumberOfSteps() const [virtual]

Returns the number of discrete steps used by the parameter.

See [SetNumberOfSteps\(\)](#) for more information about parameter steps.

Implements [AAX_IParameter](#).

14.26.3.20 template<typename T> uint32_t AAX_CParameter< T >::GetStepValue() const [virtual]

Returns the current step for the current value of the parameter.

See [SetNumberOfSteps\(\)](#) for more information about parameter steps.

Implements [AAX_IParameter](#).

14.26.3.21 template<typename T> double AAX_CParameter< T >::GetNormalizedValueFromStep(uint32_t iStep) const [virtual]

Returns the normalized value for a given step.

See [SetNumberOfSteps\(\)](#) for more information about parameter steps.

Implements [AAX_IParameter](#).

14.26.3.22 template<typename T> uint32_t AAX_CParameter< T >::GetStepValueFromNormalizedValue (double *normalizedValue*) const [virtual]

Returns the step value for a normalized value of the parameter.

See [SetNumberOfSteps\(\)](#) for more information about parameter steps.

Implements [AAX_IParameter](#).

14.26.3.23 template<typename T> void AAX_CParameter< T >::SetStepValue (uint32_t *iStep*) [virtual]

Returns the current step for the current value of the parameter.

See [SetNumberOfSteps\(\)](#) for more information about parameter steps.

Implements [AAX_IParameter](#).

14.26.3.24 template<typename T> void AAX_CParameter< T >::SetType (AAX_EParameterType *iControlType*) [virtual]

Sets the type of this parameter.

See [GetType](#) for use cases

Parameters

in	<i>iControlType</i>	The parameter's new type as an AAX_EParameterType
----	---------------------	---

Implements [AAX_IParameter](#).

14.26.3.25 template<typename T> AAX_EParameterType AAX_CParameter< T >::GetType () const [virtual]

Returns the type of this parameter as an AAX_EParameterType.

Todo Document use cases for control type

Implements [AAX_IParameter](#).

14.26.3.26 template<typename T> void AAX_CParameter< T >::SetOrientation (AAX_EParameterOrientation *iOrientation*) [virtual]

Sets the orientation of this parameter.

Parameters

in	<i>iOrientation</i>	The parameter's new orientation
----	---------------------	---------------------------------

Implements [AAX_IParameter](#).

14.26.3.27 template<typename T> AAX_EParameterOrientation AAX_CParameter< T >::GetOrientation () const [virtual]

Returns the orientation of this parameter.

Implements [AAX_IParameter](#).

14.26.3.28 template<typename T> void AAX_CParameter<T>::SetTaperDelegate(AAX_ITaperDelegateBase &
inTaperDelegate, bool *inPreserveValue* = true) [virtual]

Sets the parameter's taper delegate.

Parameters

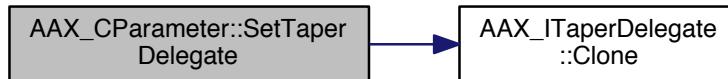
in	<i>inTaperDelegate</i>	A reference to the parameter's new taper delegate
in	<i>inPreserveValue</i>	

Todo Document this parameter

Implements [AAX_IParameter](#).

References [AAX_ITaperDelegate< T >::Clone\(\)](#).

Here is the call graph for this function:



14.26.3.29 template<typename T> void AAX_CParameter< T >::SetDisplayDelegate (AAX_IDisplayDelegateBase & *inDisplayDelegate*) [virtual]

Sets the parameter's display delegate.

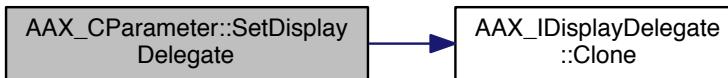
Parameters

in	<i>inDisplayDelegate</i>	A reference to the parameter's new display delegate

Implements [AAX_IParameter](#).

References [AAX_IDisplayDelegate< T >::Clone\(\)](#).

Here is the call graph for this function:



14.26.3.30 template<typename T> bool AAX_CParameter< T >::GetValueString (AAX_CString * *valueString*) const [virtual]

Serializes the parameter value into a string.

Parameters

<i>out</i>	<i>valueString</i>	A string representing the parameter's real value
------------	--------------------	--

Return values

<i>true</i>	The string conversion was successful
<i>false</i>	The string conversion was unsuccessful

Implements [AAX_IParameter](#).

14.26.3.31 template<typename T> bool AAX_CParameter<T>::GetValueString (int32_t *iMaxNumChars*, AAX_CString * *valueString*) const [virtual]

Serializes the parameter value into a string, size hint included.

Parameters

<i>in</i>	<i>iMaxNumChars</i>	A size hint for the size of the string being requested. Useful for control surfaces and other limited area text fields. (make sure that size of desired string also has room for null termination)
<i>out</i>	<i>valueString</i>	A string representing the parameter's real value

Return values

<i>true</i>	The string conversion was successful
<i>false</i>	The string conversion was unsuccessful

Implements [AAX_IParameter](#).

14.26.3.32 template<typename T> bool AAX_CParameter<T>::GetNormalizedValueFromBool (bool *value*, double * *normalizedValue*) const [virtual]

Converts a bool to a normalized parameter value.

Parameters

<i>in</i>	<i>value</i>	A value for the parameter
<i>out</i>	<i>normalizedValue</i>	The normalized parameter value associated with value

Return values

<i>true</i>	The value conversion was successful
<i>false</i>	The value conversion was unsuccessful

Implements [AAX_IParameter](#).

14.26.3.33 template<typename T> bool AAX_CParameter<T>::GetNormalizedValueFromInt32 (int32_t *value*, double * *normalizedValue*) const [virtual]

Converts an integer to a normalized parameter value.

Parameters

<i>in</i>	<i>value</i>	A value for the parameter
<i>out</i>	<i>normalizedValue</i>	The normalized parameter value associated with value

Return values

<i>true</i>	The value conversion was successful
<i>false</i>	The value conversion was unsuccessful

Implements [AAX_IParameter](#).

14.26.3.34 template<typename T> bool AAX_CParameter< T >::GetNormalizedValueFromFloat (float *value*, double * *normalizedValue*) const [virtual]

Converts a float to a normalized parameter value.

Parameters

<i>in</i>	<i>value</i>	A value for the parameter
<i>out</i>	<i>normalizedValue</i>	The normalized parameter value associated with value

Return values

<i>true</i>	The value conversion was successful
<i>false</i>	The value conversion was unsuccessful

Implements [AAX_IParameter](#).

14.26.3.35 template<typename T> bool AAX_CParameter< T >::GetNormalizedValueFromDouble (double *value*, double * *normalizedValue*) const [virtual]

Converts a double to a normalized parameter value.

Parameters

<i>in</i>	<i>value</i>	A value for the parameter
<i>out</i>	<i>normalizedValue</i>	The normalized parameter value associated with value

Return values

<i>true</i>	The value conversion was successful
<i>false</i>	The value conversion was unsuccessful

Implements [AAX_IParameter](#).

14.26.3.36 template<typename T> bool AAX_CParameter< T >::GetNormalizedValueFromString (const AAX_CString & *valueString*, double * *normalizedValue*) const [virtual]

Converts a given string to a normalized parameter value.

Parameters

<i>in</i>	<i>valueString</i>	A string representing a possible real value for the parameter
<i>out</i>	<i>normalizedValue</i>	The normalized parameter value associated with valueString

Return values

<i>true</i>	The string conversion was successful
<i>false</i>	The string conversion was unsuccessful

Implements [AAX_IParameter](#).

```
14.26.3.37 template<typename T> bool AAX_CParameter<T>::GetBoolFromNormalizedValue( double normalizedValue,  
bool * value ) const [virtual]
```

Converts a normalized parameter value to a bool representing the corresponding real value.

Parameters

<i>in</i>	<i>normalizedValue</i>	The normalized value to convert
<i>out</i>	<i>value</i>	The converted value. Set only if conversion is successful.

Return values

<i>true</i>	The conversion to bool was successful
<i>false</i>	The conversion to bool was unsuccessful

Implements [AAX_IParameter](#).

14.26.3.38 template<typename T > bool AAX_CParameter< T >::GetInt32FromNormalizedValue (double *normalizedValue*, int32_t * *value*) const [virtual]

Converts a normalized parameter value to an integer representing the corresponding real value.

Parameters

<i>in</i>	<i>normalizedValue</i>	The normalized value to convert
<i>out</i>	<i>value</i>	The converted value. Set only if conversion is successful.

Return values

<i>true</i>	The conversion to int32_t was successful
<i>false</i>	The conversion to int32_t was unsuccessful

Implements [AAX_IParameter](#).

14.26.3.39 template<typename T > bool AAX_CParameter< T >::GetFloatFromNormalizedValue (double *normalizedValue*, float * *value*) const [virtual]

Converts a normalized parameter value to a float representing the corresponding real value.

Parameters

<i>in</i>	<i>normalizedValue</i>	The normalized value to convert
<i>out</i>	<i>value</i>	The converted value. Set only if conversion is successful.

Return values

<i>true</i>	The conversion to float was successful
<i>false</i>	The conversion to float was unsuccessful

Implements [AAX_IParameter](#).

14.26.3.40 template<typename T > bool AAX_CParameter< T >::GetDoubleFromNormalizedValue (double *normalizedValue*, double * *value*) const [virtual]

Converts a normalized parameter value to a double representing the corresponding real value.

Parameters

<i>in</i>	<i>normalizedValue</i>	The normalized value to convert
<i>out</i>	<i>value</i>	The converted value. Set only if conversion is successful.

Return values

<i>true</i>	The conversion to double was successful
<i>false</i>	The conversion to double was unsuccessful

Implements [AAX_IParameter](#).

14.26.3.41 template<typename T> bool AAX_CParameter< T >::GetStringFromNormalizedValue (double *normalizedValue*, AAX_CString & *valueString*) const [virtual]

Converts a normalized parameter value to a string representing the corresponding real value.

Parameters

<i>in</i>	<i>normalizedValue</i>	A normalized parameter value
<i>out</i>	<i>valueString</i>	A string representing the parameter value associated with <i>normalizedValue</i>

Return values

<i>true</i>	The string conversion was successful
<i>false</i>	The string conversion was unsuccessful

Implements [AAX_IParameter](#).

14.26.3.42 template<typename T> bool AAX_CParameter< T >::GetStringFromNormalizedValue (double *normalizedValue*, int32_t *iMaxNumChars*, AAX_CString & *valueString*) const [virtual]

Converts a normalized parameter value to a string representing the corresponding real, size hint included. value.

Parameters

<i>in</i>	<i>normalizedValue</i>	A normalized parameter value
<i>in</i>	<i>iMaxNumChars</i>	A size hint for the size of the string being requested. Useful for control surfaces and other limited area text fields. (make sure that size of desired string also has room for null termination)
<i>out</i>	<i>valueString</i>	A string representing the parameter value associated with <i>normalizedValue</i>

Return values

<i>true</i>	The string conversion was successful
<i>false</i>	The string conversion was unsuccessful

Implements [AAX_IParameter](#).

14.26.3.43 template<typename T> bool AAX_CParameter< T >::SetValueFromString (const AAX_CString & *newValueString*) [virtual]

Converts a string to a real parameter value and sets the parameter to this value.

Parameters

<i>in</i>	<i>newValueString</i>	A string representing the parameter's new real value
-----------	-----------------------	--

Return values

<i>true</i>	The string conversion was successful
<i>false</i>	The string conversion was unsuccessful

Implements [AAX_IParameter](#).

```
14.26.3.44 template<typename T> void AAX_CParameter< T >::SetAutomationDelegate (   
    AAX_IAutomationDelegate * iAutomationDelegate ) [virtual]
```

Sets the automation delegate (if one is required)

Parameters

in	<i>iAutomation</i> \leftarrow Delegate	A reference to the parameter manager's automation delegate interface
----	--	--

Implements [AAX_IParameter](#).

References [AAX_IAutomationDelegate::RegisterParameter\(\)](#).

Here is the call graph for this function:



14.26.3.45 template<typename T> bool AAX_CParameter< T >::Automatable() const [virtual]

Returns true if the parameter is automatable, false if it is not.

Note

Subclasses that return true in this method must support host-based automation.

Implements [AAX_IParameter](#).

14.26.3.46 template<typename T> void AAX_CParameter< T >::Touch() [virtual]

Signals the automation system that a control has been touched.

Call this method in response to GUI events that begin editing, such as a mouse down. After this method has been called you are free to call [SetNormalizedValue\(\)](#) as much as you need, e.g. in order to respond to subsequent mouse moved events. Call [Release\(\)](#) to free the parameter for updates from other controls.

Implements [AAX_IParameter](#).

14.26.3.47 template<typename T> void AAX_CParameter< T >::Release(void) [virtual]

Signals the automation system that a control has been released.

Call this method in response to GUI events that complete editing, such as a mouse up. Once this method has been called you should not call [SetNormalizedValue\(\)](#) again until after the next call to [Touch\(\)](#).

Implements [AAX_IParameter](#).

14.26.3.48 template<typename T> bool AAX_CParameter< T >::GetValueAsBool(bool * value) const [virtual]

Retrieves the parameter's value as a bool.

Parameters

<i>out</i>	<i>value</i>	The parameter's real value. Set only if conversion is successful.
------------	--------------	---

Return values

<i>true</i>	The conversion to bool was successful
<i>false</i>	The conversion to bool was unsuccessful

Implements [AAX_IParameter](#).

14.26.3.49 template<typename T > bool AAX_CParameter< T >::GetValueAsInt32 (int32_t * *value*) const [virtual]

Retrieves the parameter's value as an int32_t.

Parameters

<i>out</i>	<i>value</i>	The parameter's real value. Set only if conversion is successful.
------------	--------------	---

Return values

<i>true</i>	The conversion to int32_t was successful
<i>false</i>	The conversion to int32_t was unsuccessful

Implements [AAX_IParameter](#).

14.26.3.50 template<typename T > bool AAX_CParameter< T >::GetValueAsFloat (float * *value*) const [virtual]

Retrieves the parameter's value as a float.

Parameters

<i>out</i>	<i>value</i>	The parameter's real value. Set only if conversion is successful.
------------	--------------	---

Return values

<i>true</i>	The conversion to float was successful
<i>false</i>	The conversion to float was unsuccessful

Implements [AAX_IParameter](#).

14.26.3.51 template<typename T > bool AAX_CParameter< T >::GetValueAsDouble (double * *value*) const [virtual]

Retrieves the parameter's value as a double.

Parameters

<i>out</i>	<i>value</i>	The parameter's real value. Set only if conversion is successful.
------------	--------------	---

Return values

<i>true</i>	The conversion to double was successful
<i>false</i>	The conversion to double was unsuccessful

Implements [AAX_IParameter](#).

14.26.3.52 template<typename T> bool AAX_CParameter<T>::GetValueAsString (AAX_IString * value) const
[virtual]

Retrieves the parameter's value as a string.

Parameters

<code>out</code>	<code>value</code>	The parameter's real value. Set only if conversion is successful.
------------------	--------------------	---

Return values

<code>true</code>	The conversion to string was successful
<code>false</code>	The conversion to string was unsuccessful

Implements [AAX_IParameter](#).

14.26.3.53 template<typename T> bool AAX_CParameter< T >::SetValueWithBool(bool value) [virtual]

Sets the parameter's value as a bool.

Parameters

<code>out</code>	<code>value</code>	The parameter's real value. Set only if conversion is successful.
------------------	--------------------	---

Return values

<code>true</code>	The conversion from bool was successful
<code>false</code>	The conversion from bool was unsuccessful

Implements [AAX_IParameter](#).

14.26.3.54 template<typename T> bool AAX_CParameter< T >::SetValueWithInt32(int32_t value) [virtual]

Sets the parameter's value as an int32_t.

Parameters

<code>out</code>	<code>value</code>	The parameter's real value. Set only if conversion is successful.
------------------	--------------------	---

Return values

<code>true</code>	The conversion from int32_t was successful
<code>false</code>	The conversion from int32_t was unsuccessful

Implements [AAX_IParameter](#).

14.26.3.55 template<typename T> bool AAX_CParameter< T >::SetValueWithFloat(float value) [virtual]

Sets the parameter's value as a float.

Parameters

<code>out</code>	<code>value</code>	The parameter's real value. Set only if conversion is successful.
------------------	--------------------	---

Return values

<code>true</code>	The conversion from float was successful
<code>false</code>	The conversion from float was unsuccessful

Implements [AAX_IParameter](#).

14.26.3.56 template<typename T> bool AAX_CParameter< T >::SetValueWithDouble(double value) [virtual]

Sets the parameter's value as a double.

Parameters

out	<i>value</i>	The parameter's real value. Set only if conversion is successful.
-----	--------------	---

Return values

<i>true</i>	The conversion from double was successful
<i>false</i>	The conversion from double was unsuccessful

Implements [AAX_IParameter](#).

14.26.3.57 `template<typename T> bool AAX_CParameter< T >::SetValueWithString (const AAX_IString & value) [virtual]`

Sets the parameter's value as a string.

Parameters

out	<i>value</i>	The parameter's real value. Set only if conversion is successful.
-----	--------------	---

Return values

<i>true</i>	The conversion from string was successful
<i>false</i>	The conversion from string was unsuccessful

Implements [AAX_IParameter](#).

14.26.3.58 `template<typename T> void AAX_CParameter< T >::UpdateNormalizedValue (double newNormalizedValue) [virtual]`

Sets the parameter's state given a normalized value.

This is the second half of the parameter setting operation that is initiated with a call to [SetValue\(\)](#). Parameters should not be set directly using this method; instead, use [SetValue\(\)](#).

Parameters

in	<i>newNormalizedValue</i>	Normalized value that will be used to set the parameter's new state
----	---------------------------	---

Implements [AAX_IParameter](#).

14.26.3.59 `template<typename T> void AAX_CParameter< T >::SetValue (T newValue)`

Initiates a host request to set the parameter's value.

This method normalizes the provided value and sends a request for the value change to the AAX host. The host responds with a call to [AAX_IParameter::UpdateNormalizedValue\(\)](#) to complete the set operation.

Parameters

in	<i>newValue</i>	The parameter's new value
----	-----------------	---------------------------

14.26.3.60 `template<typename T> T AAX_CParameter< T >::GetValue () const`

Returns the parameter's value.

This is the parameter's real, logical value and should not be normalized

14.26.3.61 template<typename T> void AAX_CParameter< T >::SetDefaultValue (T newDefaultValue)

Set the parameter's default value.

This is the parameter's real, logical value and should not be normalized

Parameters

in	<i>newDefaultValue</i>	The parameter's new default value
----	------------------------	-----------------------------------

14.26.3.62 template<typename T> T AAX_CParameter< T >::GetDefaultValue () const

Returns the parameter's default value.

This is the parameter's real, logical value and should not be normalized

14.26.3.63 template<typename T> const AAX_ITaperDelegate< T > * AAX_CParameter< T >::TaperDelegate () const

Returns a reference to the parameter's taper delegate.

14.26.3.64 template<typename T> const AAX_IDisplayDelegate< T > * AAX_CParameter< T >::DisplayDelegate () const

Returns a reference to the parameter's display delegate.

14.26.3.65 template<> bool AAX_CParameter< AAX_CString >::GetValueAsString (AAX_IString * value) const [virtual]

Retrieves the parameter's value as a string.

Parameters

out	<i>value</i>	The parameter's real value. Set only if conversion is successful.
-----	--------------	---

Return values

<i>true</i>	The conversion to string was successful
<i>false</i>	The conversion to string was unsuccessful

Implements [AAX_IParameter](#).

14.26.3.66 template<> bool AAX_CParameter< bool >::SetValueWithBool (bool value) [virtual]

Sets the parameter's value as a bool.

Parameters

out	<i>value</i>	The parameter's real value. Set only if conversion is successful.
-----	--------------	---

Return values

<i>true</i>	The conversion from bool was successful
<i>false</i>	The conversion from bool was unsuccessful

Implements [AAX_IParameter](#).

14.26.3.67 template<> bool AAX_CParameter< int32_t >::SetValueWithInt32 (int32_t value) [virtual]

Sets the parameter's value as an int32_t.

Parameters

out	value	The parameter's real value. Set only if conversion is successful.
-----	-------	---

Return values

true	The conversion from int32_t was successful
false	The conversion from int32_t was unsuccessful

Implements [AAX_IParameter](#).

14.26.3.68 template<> bool AAX_CParameter< float >::SetValueWithFloat (float value) [virtual]

Sets the parameter's value as a float.

Parameters

out	value	The parameter's real value. Set only if conversion is successful.
-----	-------	---

Return values

true	The conversion from float was successful
false	The conversion from float was unsuccessful

Implements [AAX_IParameter](#).

14.26.3.69 template<> bool AAX_CParameter< double >::SetValueWithDouble (double value) [virtual]

Sets the parameter's value as a double.

Parameters

out	value	The parameter's real value. Set only if conversion is successful.
-----	-------	---

Return values

true	The conversion from double was successful
false	The conversion from double was unsuccessful

Implements [AAX_IParameter](#).

14.26.3.70 template<> bool AAX_CParameter< AAX_CString >::SetValueWithString (const AAX_IString & value) [virtual]

Sets the parameter's value as a string.

Parameters

out	value	The parameter's real value. Set only if conversion is successful.
-----	-------	---

Return values

<i>true</i>	The conversion from string was successful
<i>false</i>	The conversion from string was unsuccessful

Implements [AAX_IParameter](#).

14.26.3.71 template<> bool AAX_CParameter< bool >::GetNormalizedValueFromBool (bool *value*, double * *normalizedValue*) const [virtual]

Converts a bool to a normalized parameter value.

Parameters

<i>in</i>	<i>value</i>	A value for the parameter
<i>out</i>	<i>normalizedValue</i>	The normalized parameter value associated with value

Return values

<i>true</i>	The value conversion was successful
<i>false</i>	The value conversion was unsuccessful

Implements [AAX_IParameter](#).

14.26.3.72 template<> bool AAX_CParameter< int32_t >::GetNormalizedValueFromInt32 (int32_t *value*, double * *normalizedValue*) const [virtual]

Converts an integer to a normalized parameter value.

Parameters

<i>in</i>	<i>value</i>	A value for the parameter
<i>out</i>	<i>normalizedValue</i>	The normalized parameter value associated with value

Return values

<i>true</i>	The value conversion was successful
<i>false</i>	The value conversion was unsuccessful

Implements [AAX_IParameter](#).

14.26.3.73 template<> bool AAX_CParameter< float >::GetNormalizedValueFromFloat (float *value*, double * *normalizedValue*) const [virtual]

Converts a float to a normalized parameter value.

Parameters

<i>in</i>	<i>value</i>	A value for the parameter
<i>out</i>	<i>normalizedValue</i>	The normalized parameter value associated with value

Return values

<i>true</i>	The value conversion was successful
<i>false</i>	The value conversion was unsuccessful

Implements [AAX_IParameter](#).

14.26.3.74 template<> bool AAX_CParameter< double >::GetNormalizedValueFromDouble (double *value*, double * *normalizedValue*) const [virtual]

Converts a double to a normalized parameter value.

Parameters

in	<i>value</i>	A value for the parameter
out	<i>normalizedValue</i>	The normalized parameter value associated with value

Return values

<i>true</i>	The value conversion was successful
<i>false</i>	The value conversion was unsuccessful

Implements [AAX_IParameter](#).

14.26.3.75 template<> bool AAX_CParameter< bool >::GetBoolFromNormalizedValue (double *normalizedValue*, bool * *value*) const [virtual]

Converts a normalized parameter value to a bool representing the corresponding real value.

Parameters

in	<i>normalizedValue</i>	The normalized value to convert
out	<i>value</i>	The converted value. Set only if conversion is successful.

Return values

<i>true</i>	The conversion to bool was successful
<i>false</i>	The conversion to bool was unsuccessful

Implements [AAX_IParameter](#).

14.26.3.76 template<> bool AAX_CParameter< int32_t >::GetInt32FromNormalizedValue (double *normalizedValue*, int32_t * *value*) const [virtual]

Converts a normalized parameter value to an integer representing the corresponding real value.

Parameters

in	<i>normalizedValue</i>	The normalized value to convert
out	<i>value</i>	The converted value. Set only if conversion is successful.

Return values

<i>true</i>	The conversion to int32_t was successful
<i>false</i>	The conversion to int32_t was unsuccessful

Implements [AAX_IParameter](#).

14.26.3.77 template<> bool AAX_CParameter< float >::GetFloatFromNormalizedValue (double *normalizedValue*, float * *value*) const [virtual]

Converts a normalized parameter value to a float representing the corresponding real value.

Parameters

in	<i>normalizedValue</i>	The normalized value to convert
out	<i>value</i>	The converted value. Set only if conversion is successful.

Return values

<i>true</i>	The conversion to float was successful
<i>false</i>	The conversion to float was unsuccessful

Implements [AAX_IParameter](#).

14.26.3.78 template<> bool AAX_CParameter< double >::GetDoubleFromNormalizedValue (double *normalizedValue*, double * *value*) const [virtual]

Converts a normalized parameter value to a double representing the corresponding real value.

Parameters

in	<i>normalizedValue</i>	The normalized value to convert
out	<i>value</i>	The converted value. Set only if conversion is successful.

Return values

<i>true</i>	The conversion to double was successful
<i>false</i>	The conversion to double was unsuccessful

Implements [AAX_IParameter](#).

14.26.4 Member Data Documentation

14.26.4.1 template<typename T > AAX_CStringAbbreviations AAX_CParameter< T >::mNames [protected]

14.26.4.2 template<typename T > bool AAX_CParameter< T >::mAutomatable [protected]

14.26.4.3 template<typename T > uint32_t AAX_CParameter< T >::mNumSteps [protected]

14.26.4.4 template<typename T > AAX_EParameterType AAX_CParameter< T >::mControlType [protected]

14.26.4.5 template<typename T > AAX_EParameterOrientation AAX_CParameter< T >::mOrientation [protected]

14.26.4.6 template<typename T > AAX_ITaperDelegate<T>* AAX_CParameter< T >::mTaperDelegate [protected]

14.26.4.7 template<typename T > AAX_IDisplayDelegate<T>* AAX_CParameter< T >::mDisplayDelegate [protected]

14.26.4.8 template<typename T > AAX_IAutomationDelegate* AAX_CParameter< T >::mAutomationDelegate [protected]

14.26.4.9 template<typename T > bool AAX_CParameter< T >::mNeedNotify [protected]

14.26.4.10 template<typename T > AAX_CParameterValue<T> AAX_CParameter< T >::mValue [protected]

14.26.4.11 template<typename T > T AAX_CParameter< T >::mDefaultValue [protected]

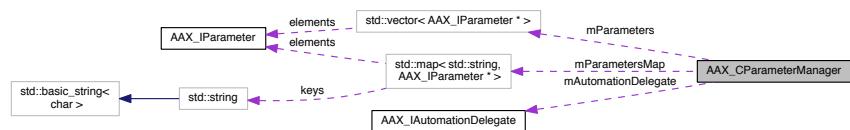
The documentation for this class was generated from the following file:

- [AAX_CParameter.h](#)

14.27 AAX_CParameterManager Class Reference

```
#include <AAX_CParameterManager.h>
```

Collaboration diagram for AAX_CParameterManager:



14.27.1 Description

A container object for plug-in parameters.

This implementation uses a STL vector to store a plug-in's set of parameters. This class contains a real implementation of the [Parameter Manager](#) (as opposed to a proxy.)

For more information, see [Parameter Manager](#).

Todo Should the Parameter Manager return error codes?

Public Member Functions

- [AAX_CParameterManager \(\)](#)
- [~AAX_CParameterManager \(\)](#)
- void [Initialize \(AAX_IAutomationDelegate *iAutomationDelegateUnknown\)](#)
Initialize the parameter manager.
- int32_t [NumParameters \(\) const](#)
Returns the number of parameters in this instance of the parameter manager.
- void [RemoveParameterByID \(AAX_CParamID identifier\)](#)
Removes a parameter from the manager.
- void [RemoveAllParameters \(\)](#)
Removes all parameters from the manager.
- AAX_IParameter * [GetParameterByID \(AAX_CParamID identifier\)](#)
Given a parameter ID, retrieves a reference to the requested parameter.
- const AAX_IParameter * [GetParameterByID \(AAX_CParamID identifier\) const](#)
Given a parameter ID, retrieves a const reference to the requested parameter.
- AAX_IParameter * [GetParameterByName \(const char *name\)](#)
Given a parameter name, retrieves a reference to the requested parameter.
- const AAX_IParameter * [GetParameterByName \(const char *name\) const](#)
Given a parameter name, retrieves a const reference to the requested parameter.
- AAX_IParameter * [GetParameter \(int32_t index\)](#)

Given a parameter index, retrieves a reference to the requested parameter.

- const [AAX_IParameter * GetParameter](#) (int32_t index) const
Given a parameter index, retrieves a const reference to the requested parameter.
- int32_t [GetParameterIndex](#) (AAX_CParamID identifier) const
- void [AddParameter](#) (AAX_IParameter *param)
- void [RemoveParameter](#) (AAX_IParameter *param)

Protected Attributes

- [AAX_IAutomationDelegate * mAutomationDelegate](#)
- std::vector< [AAX_IParameter * > mParameters](#)
- std::map< std::string, [AAX_IParameter * > mParametersMap](#)

14.27.2 Constructor & Destructor Documentation

14.27.2.1 [AAX_CParameterManager::AAX_CParameterManager\(\)](#)

14.27.2.2 [AAX_CParameterManager::~AAX_CParameterManager\(\)](#)

14.27.3 Member Function Documentation

14.27.3.1 [void AAX_CParameterManager::Initialize\(AAX_IAutomationDelegate * iAutomationDelegateUnknown \)](#)

Initialize the parameter manager.

Called when plug-in instance is first instantiated. This method will initialize the plug-in's automation delegate, among other set-up tasks.

Parameters

in	<i>iAutomation</i> ↪ Delegate ↪ Unknown	A reference to the plug-in's AAX_IAutomationDelegate interface
----	---	--

14.27.3.2 [int32_t AAX_CParameterManager::NumParameters\(\)](#) const

Returns the number of parameters in this instance of the parameter manager.

14.27.3.3 [void AAX_CParameterManager::RemoveParameterByID\(AAX_CParamID identifier \)](#)

Removes a parameter from the manager.

Todo Should this method return success/failure code?

Parameters

in	<i>identifier</i>	ID of the parameter that will be removed
----	-------------------	--

14.27.3.4 [void AAX_CParameterManager::RemoveAllParameters\(\)](#)

Removes all parameters from the manager.

Todo Should this method return success/failure code?

14.27.3.5 AAX_IParameter* AAX_CParameterManager::GetParameterByID (AAX_CParamID *identifier*)

Given a parameter ID, retrieves a reference to the requested parameter.

Parameters

in	<i>identifier</i>	ID of the parameter that will be retrieved
----	-------------------	--

Referenced by AAX_CMonolithicParameters::UpdateParameterNormalizedValue().

Here is the caller graph for this function:



14.27.3.6 const AAX_IParameter* AAX_CParameterManager::GetParameterByID (AAX_CParamID *identifier*) const

Given a parameter ID, retrieves a const reference to the requested parameter.

Parameters

in	<i>identifier</i>	ID of the parameter that will be retrieved
----	-------------------	--

14.27.3.7 AAX_IParameter* AAX_CParameterManager::GetParameterByName (const char * *name*)

Given a parameter name, retrieves a reference to the requested parameter.

Note

Parameter names may be ambiguous

Parameters

in	<i>name</i>	Name of the parameter that will be retrieved
----	-------------	--

14.27.3.8 const AAX_IParameter* AAX_CParameterManager::GetParameterByName (const char * *name*) const

Given a parameter name, retrieves a const reference to the requested parameter.

Note

Parameter names may be ambiguous

Parameters

in	<i>name</i>	ID of the parameter that will be retrieved
----	-------------	--

14.27.3.9 AAX_IParameter* AAX_CParameterManager::GetParameter (int32_t *index*)

Given a parameter index, retrieves a reference to the requested parameter.

Parameter indices are incremented in the order that parameters are added to the manager. See [AddParameter\(\)](#).

Parameters

in	<i>index</i>	Index of the parameter that will be retrieved
----	--------------	---

14.27.3.10 const AAX_IParameter* AAX_CParameterManager::GetParameter (int32_t *index*) const

Given a parameter index, retrieves a const reference to the requested parameter.

Parameter indices are incremented in the order that parameters are added to the manager. See [AddParameter\(\)](#).

Parameters

in	<i>index</i>	Index of the parameter that will be retrieved
----	--------------	---

14.27.3.11 int32_t AAX_CParameterManager::GetParameterIndex (AAX_CParamID *identifier*) const

Given a parameter ID, retrieves the index for the specified parameter

Parameters

in	<i>identifier</i>	ID of the parameter that will be retrieved
----	-------------------	--

14.27.3.12 void AAX_CParameterManager::AddParameter (AAX_IParameter * *param*)

Adds a parameter to the manager

Todo Should this method return success/failure code?

Parameters

in	<i>param</i>	Reference to the parameter that will be added
----	--------------	---

14.27.3.13 void AAX_CParameterManager::RemoveParameter (AAX_IParameter * *param*)

Removes a parameter to the manager

Todo Should this method return success/failure code?

Parameters

in	<i>param</i>	Reference to the parameter that will be removed
----	--------------	---

14.27.4 Member Data Documentation

14.27.4.1 AAX_IAutomationDelegate* AAX_CParameterManager::mAutomationDelegate [protected]

14.27.4.2 std::vector<AAX_IParameter*> AAX_CParameterManager::mParameters [protected]

14.27.4.3 std::map<std::string, AAX_IParameter*> AAX_CParameterManager::mParametersMap [protected]

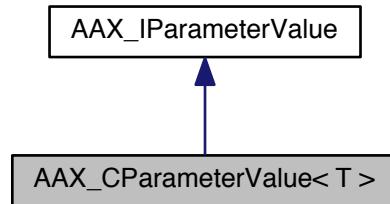
The documentation for this class was generated from the following file:

- [AAX_CParameterManager.h](#)

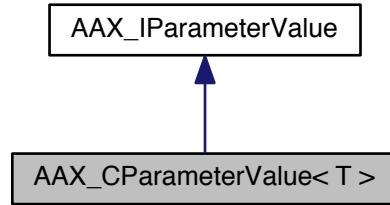
14.28 AAX_CParameterValue< T > Class Template Reference

```
#include <AAx_CParameter.h>
```

Inheritance diagram for AAX_CParameterValue< T >:



Collaboration diagram for AAX_CParameterValue< T >:



14.28.1 Description

```
template<typename T> class AAX_CParameterValue< T >
```

Concrete implementation of [AAX_IParameterValue](#).

Used by [AAX_CParameter](#)

Public Types

- enum [Defaults](#) { [eParameterDefaultMaxIdentifierSize](#) = 32, [eParameterDefaultMaxIdentifierLength](#) = [eParameterDefaultMaxIdentifierSize](#) - 1 }

Public Member Functions

- virtual [~AAX_CParameterValue](#) ()
Virtual destructor.
- [AAX_DEFAULT_MOVECTOR\(AAX_CParameterValue\)](#)

- `AAX_DEFAULT_MOVE_OPER (AAX_CParameterValue)`
- `AAX_DELETE (AAX_CParameterValue &operator=(const AAX_CParameterValue &))`
- `AAX_CParameterValue (AAX_CParamID identifier)`
Constructs an `AAX_CParameterValue` object.
- `AAX_CParameterValue (AAX_CParamID identifier, const T &value)`
Constructs an `AAX_CParameterValue` object with a defined initial state.
- `AAX_CParameterValue (const AAX_CParameterValue< T > &other)`
Copy constructor for `AAX_CParameterValue`.
- `const T & Get () const`
Direct access to the template instance's value.
- `void Set (const T &inValue)`
Direct access to the template instance's value.
- `virtual AAX_IParameterValue * Clone () const AAX_OVERRIDE`
Clones the parameter object.
- `virtual AAX_CParamID Identifier () const AAX_OVERRIDE`
Returns the parameter's unique identifier.
- `template<> bool GetValueAsBool (bool *value) const`
Retrieves the parameter's value as a bool.
- `template<> bool GetValueAsInt32 (int32_t *value) const`
Retrieves the parameter's value as an int32_t.
- `template<> bool GetValueAsFloat (float *value) const`
Retrieves the parameter's value as a float.
- `template<> bool GetValueAsDouble (double *value) const`
Retrieves the parameter's value as a double.
- `template<> bool GetValueAsString (AAX_IString *value) const`
Retrieves the parameter's value as a string.

Typed accessors

- `virtual bool GetValueAsBool (bool *value) const AAX_OVERRIDE`
Retrieves the parameter's value as a bool.
- `virtual bool GetValueAsInt32 (int32_t *value) const AAX_OVERRIDE`
Retrieves the parameter's value as an int32_t.
- `virtual bool GetValueAsFloat (float *value) const AAX_OVERRIDE`
Retrieves the parameter's value as a float.
- `virtual bool GetValueAsDouble (double *value) const AAX_OVERRIDE`
Retrieves the parameter's value as a double.
- `virtual bool GetValueAsString (AAX_IString *value) const AAX_OVERRIDE`
Retrieves the parameter's value as a string.

14.28.2 Member Enumeration Documentation

14.28.2.1 template<typename T> enum AAX_CParameterValue::Defaults

Enumerator

`eParameterDefaultMaxIdentifierSize`
`eParameterDefaultMaxIdentifierLength`

14.28.3 Constructor & Destructor Documentation

14.28.3.1 template<typename T> virtual AAX_CParameterValue< T >::~AAX_CParameterValue () [inline], [virtual]

Virtual destructor.

14.28.3.2 template<typename T> AAX_CParameterValue<T>::AAX_CParameterValue(AAX_CParamID
identifier) [explicit]

Constructs an [AAX_CParameterValue](#) object.

Parameters

in	<i>identifier</i>	Unique ID for the parameter value, these can only be 31 characters long at most. (the fixed length is a requirement for some optimizations in the host)
----	-------------------	---

Note

The initial state of the parameter value is undefined

14.28.3.3 template<typename T> AAX_CParameterValue<T>::AAX_CParameterValue (AAX_CParamID *identifier*, const T & *value*) [explicit]

Constructs an [AAX_CParameterValue](#) object with a defined initial state.

Parameters

in	<i>identifier</i>	Unique ID for the parameter value, these can only be 31 characters long at most. (the fixed length is a requirement for some optimizations in the host)
in	<i>value</i>	Initial state of the parameter value

14.28.3.4 template<typename T> AAX_CParameterValue<T>::AAX_CParameterValue (const AAX_CParameterValue<T> & *other*) [explicit]

Copy constructor for [AAX_CParameterValue](#).

14.28.4 Member Function Documentation

14.28.4.1 template<typename T> AAX_CParameterValue<T>::AAX_DEFAULT_MOVECTOR (AAX_CParameterValue<T>)

14.28.4.2 template<typename T> AAX_CParameterValue<T>::AAX_DEFAULTMOVEOPER (AAX_CParameterValue<T>)

14.28.4.3 template<typename T> AAX_CParameterValue<T>::AAX_DELETE (AAX_CParameterValue<T> & *operator=* (const AAX_CParameterValue<T> &))

14.28.4.4 template<typename T> const T& AAX_CParameterValue<T>::Get () const [inline]

Direct access to the template instance's value.

14.28.4.5 template<typename T> void AAX_CParameterValue<T>::Set (const T & *inValue*) [inline]

Direct access to the template instance's value.

14.28.4.6 template<typename T> virtual AAX_IParameterValue* AAX_CParameterValue<T>::Clone () const [inline], [virtual]

Clones the parameter object.

Note

Does NOT set the automation delegate on the clone; ownership of the automation delegate and parameter registration/unregistration stays with the original parameter

Implements [AAX_IParameterValue](#).

14.28.4.7 template<typename T> virtual AAX_CParamID AAX_CParameterValue< T >::Identifier () const
[inline], [virtual]

Returns the parameter's unique identifier.

This unique ID is used by the [Parameter Manager](#) and by outside applications to uniquely identify and target control messages. This value may not be changed after the parameter has been constructed.

Implements [AAX_IParameterValue](#).

14.28.4.8 template<typename T > bool AAX_CParameterValue< T >::GetValueAsBool (bool * value) const
[virtual]

Retrieves the parameter's value as a bool.

Parameters

out	value	The parameter's real value. Set only if conversion is successful.
-----	-------	---

Return values

true	The conversion to bool was successful
false	The conversion to bool was unsuccessful

Implements [AAX_IParameterValue](#).

14.28.4.9 template<typename T > bool AAX_CParameterValue< T >::GetValueAsInt32 (int32_t * value) const
[virtual]

Retrieves the parameter's value as an int32_t.

Parameters

out	value	The parameter's real value. Set only if conversion is successful.
-----	-------	---

Return values

true	The conversion to int32_t was successful
false	The conversion to int32_t was unsuccessful

Implements [AAX_IParameterValue](#).

14.28.4.10 template<typename T > bool AAX_CParameterValue< T >::GetValueAsFloat (float * value) const
[virtual]

Retrieves the parameter's value as a float.

Parameters

out	value	The parameter's real value. Set only if conversion is successful.
-----	-------	---

Return values

true	The conversion to float was successful
false	The conversion to float was unsuccessful

Implements [AAX_IParameterValue](#).

```
14.28.4.11 template<typename T> bool AAX_CParameterValue<T>::GetValueAsDouble( double * value ) const  
[virtual]
```

Retrieves the parameter's value as a double.

Parameters

<code>out</code>	<code>value</code>	The parameter's real value. Set only if conversion is successful.
------------------	--------------------	---

Return values

<code>true</code>	The conversion to double was successful
<code>false</code>	The conversion to double was unsuccessful

Implements [AAX_IParameterValue](#).

14.28.4.12 template<typename T> bool AAX_CParameterValue< T >::GetValueAsString (`AAX_IString * value`)
`const [virtual]`

Retrieves the parameter's value as a string.

Parameters

<code>out</code>	<code>value</code>	The parameter's real value. Set only if conversion is successful.
------------------	--------------------	---

Return values

<code>true</code>	The conversion to string was successful
<code>false</code>	The conversion to string was unsuccessful

Implements [AAX_IParameterValue](#).

14.28.4.13 template<> bool AAX_CParameterValue< bool >::GetValueAsBool (`bool * value`) const [virtual]

Retrieves the parameter's value as a bool.

Parameters

<code>out</code>	<code>value</code>	The parameter's real value. Set only if conversion is successful.
------------------	--------------------	---

Return values

<code>true</code>	The conversion to bool was successful
<code>false</code>	The conversion to bool was unsuccessful

Implements [AAX_IParameterValue](#).

14.28.4.14 template<> bool AAX_CParameterValue< int32_t >::GetValueAsInt32 (`int32_t * value`) const [virtual]

Retrieves the parameter's value as an int32_t.

Parameters

<code>out</code>	<code>value</code>	The parameter's real value. Set only if conversion is successful.
------------------	--------------------	---

Return values

<code>true</code>	The conversion to int32_t was successful
<code>false</code>	The conversion to int32_t was unsuccessful

Implements [AAX_IParameterValue](#).

14.28.4.15 template<> bool AAX_CParameterValue< float >::GetValueAsFloat (`float * value`) const [virtual]

Retrieves the parameter's value as a float.

Parameters

<code>out</code>	<code>value</code>	The parameter's real value. Set only if conversion is successful.
------------------	--------------------	---

Return values

<code>true</code>	The conversion to float was successful
<code>false</code>	The conversion to float was unsuccessful

Implements [AAX_IParameterValue](#).

14.28.4.16 `template<> bool AAX_CParameterValue< double >::GetValueAsDouble (double * value) const [virtual]`

Retrieves the parameter's value as a double.

Parameters

<code>out</code>	<code>value</code>	The parameter's real value. Set only if conversion is successful.
------------------	--------------------	---

Return values

<code>true</code>	The conversion to double was successful
<code>false</code>	The conversion to double was unsuccessful

Implements [AAX_IParameterValue](#).

14.28.4.17 `template<> bool AAX_CParameterValue< AAX_CString >::GetValueAsString (AAX_IString * value) const [virtual]`

Retrieves the parameter's value as a string.

Parameters

<code>out</code>	<code>value</code>	The parameter's real value. Set only if conversion is successful.
------------------	--------------------	---

Return values

<code>true</code>	The conversion to string was successful
<code>false</code>	The conversion to string was unsuccessful

Implements [AAX_IParameterValue](#).

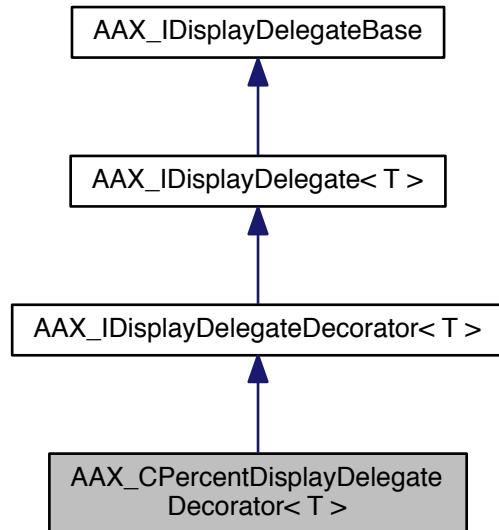
The documentation for this class was generated from the following file:

- [AAX_CParameter.h](#)

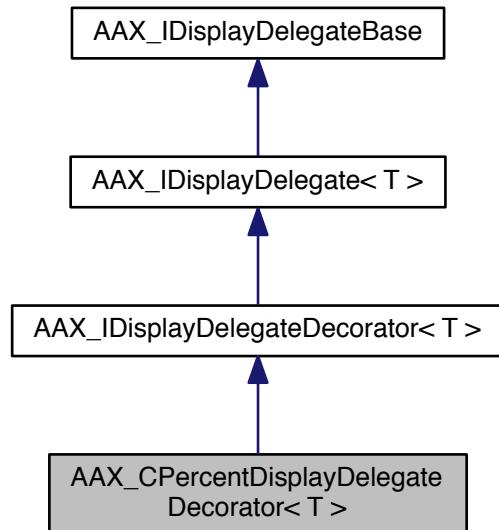
14.29 AAX_CPercentDisplayDelegateDecorator< T > Class Template Reference

```
#include <AAX_CPercentDisplayDelegateDecorator.h>
```

Inheritance diagram for AAX_CPercentDisplayDelegateDecorator< T >:



Collaboration diagram for AAX_CPercentDisplayDelegateDecorator< T >:



14.29.1 Description

```
template<typename T> class AAX_CPercentDisplayDelegateDecorator< T >
```

A percent decorator conforming to [AAX_IDisplayDelegateDecorator](#).

This class is an [AAX_IDisplayDelegateDecorator](#), meaning that it acts as a wrapper for other display delegates or concrete display types. For more information about display delegate decorators in [AAX](#), see [Display delegate decorators](#)

The behavior of this class is to provide string conversion to and from percentage (%) values. When converting a parameter value to a string, it takes the real value and performs a % conversion before passing the value on to a concrete implementation to get a value string. It then adds on the "%" string at the end to signify that the value was converted. This allows something like a gain value to remain internally linear at all times even though its display is converted to a percentage.

The inverse operation is also supported; this class can convert a percentage-formatted string into its associated real value. The string will first be converted to a number, then that number will have the inverse % calculation applied to it to retrieve the parameter's actual value.

Public Member Functions

- [AAX_CPercentDisplayDelegateDecorator](#) (const [AAX_IDisplayDelegate](#)< T > &displayDelegate)
- virtual [AAX_CPercentDisplayDelegateDecorator](#)< T > * [Clone](#) () const [AAX_OVERRIDE](#)
Constructs and returns a copy of the display delegate decorator.
- virtual bool [ValueToString](#) (T value, [AAX_CString](#) *valueString) const [AAX_OVERRIDE](#)
Converts a string to a real parameter value.
- virtual bool [ValueToString](#) (T value, int32_t maxNumChars, [AAX_CString](#) *valueString) const [AAX_OVERRIDE](#)
Converts a string to a real parameter value with a size constraint.
- virtual bool [StringToValue](#) (const [AAX_CString](#) &valueString, T *value) const [AAX_OVERRIDE](#)
Converts a string to a real parameter value.

14.29.2 Constructor & Destructor Documentation

14.29.2.1 template<typename T > [AAX_CPercentDisplayDelegateDecorator](#)< T >::[AAX_CPercentDisplayDelegateDecorator](#) (const [AAX_IDisplayDelegate](#)< T > & displayDelegate)

14.29.3 Member Function Documentation

14.29.3.1 template<typename T > [AAX_CPercentDisplayDelegateDecorator](#)< T > * [AAX_CPercentDisplayDelegateDecorator](#)< T >::[Clone](#) () const [virtual]

Constructs and returns a copy of the display delegate decorator.

In general, this method's implementation can use a simple copy constructor:

```
template <typename T>
AAX_CSubclassDisplayDelegate<T>* AAX_CSubclassDisplayDelegate<T>::Clone() const
{
    return new AAX_CSubclassDisplayDelegate(*this);
}
```

Note

This is an idiomatic method in the decorator pattern, so watch for potential problems if this method is ever changed or removed.

Reimplemented from [AAX_IDisplayDelegateDecorator](#)< T >.

14.29.3.2 template<typename T> bool AAX_CPercentDisplayDelegateDecorator< T >::ValueToString (T value, AAX_CString * valueString) const [virtual]

Converts a string to a real parameter value.

Override of the [AAX_IDisplayDelegate](#) implementation to call into the wrapped object. Display delegate decorators should call into this implementation to pass [ValueToString\(\)](#) calls on to the wrapped object after applying their own value-to-string decoration.

Parameters

in	valueString	The string that will be converted
out	value	The real parameter value corresponding to valueString

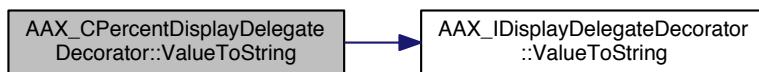
Return values

true	The string conversion was successful
false	The string conversion was unsuccessful

Reimplemented from [AAX_IDisplayDelegateDecorator< T >](#).

References [AAX_IDisplayDelegateDecorator< T >::ValueToString\(\)](#).

Here is the call graph for this function:



14.29.3.3 template<typename T> bool AAX_CPercentDisplayDelegateDecorator< T >::ValueToString (T value, int32_t maxNumChars, AAX_CString * valueString) const [virtual]

Converts a string to a real parameter value with a size constraint.

Override of the [AAX_IDisplayDelegate](#) implementation to call into the wrapped object. Display delegate decorators should call into this implementation to pass [ValueToString\(\)](#) calls on to the wrapped object after applying their own value-to-string decoration.

Parameters

in	valueString	The string that will be converted
in	maxNumChars	Size hint for the desired maximum number of characters in the string (not including null termination)
out	value	The real parameter value corresponding to valueString

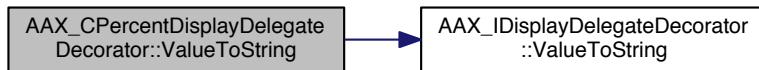
Return values

true	The string conversion was successful
false	The string conversion was unsuccessful

Reimplemented from [AAX_IDisplayDelegateDecorator< T >](#).

References [AAX_IDisplayDelegateDecorator< T >::ValueToString\(\)](#).

Here is the call graph for this function:



14.29.3.4 template<typename T > bool AAX_CPercentDisplayDelegateDecorator< T >::StringToValue (const AAX_CString & valueString, T * value) const [virtual]

Converts a string to a real parameter value.

Override of the DisplayDecorator implementation to call into the wrapped object. Display delegate decorators should call into this implementation to pass [StringToValue\(\)](#) calls on to the wrapped object after applying their own string-to-value decoding.

Parameters

in	<i>valueString</i>	The string that will be converted
out	<i>value</i>	The real parameter value corresponding to valueString

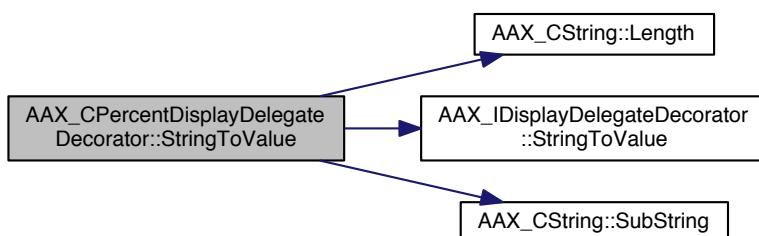
Return values

<i>true</i>	The string conversion was successful
<i>false</i>	The string conversion was unsuccessful

Reimplemented from [AAX_IDisplayDelegateDecorator< T >](#).

References [AAX_CString::Length\(\)](#), [AAX_IDisplayDelegateDecorator< T >::StringToValue\(\)](#), and [AAX_CString::SubString\(\)](#).

Here is the call graph for this function:



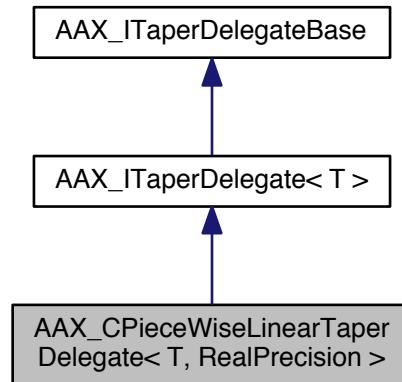
The documentation for this class was generated from the following file:

- [AAX_CPercentDisplayDelegateDecorator.h](#)

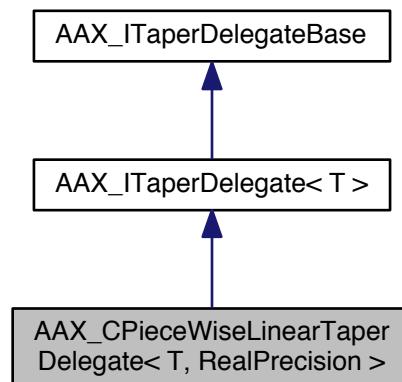
14.30 AAX_CPieceWiseLinearTaperDelegate< T, RealPrecision > Class Template Reference

```
#include <AAx_CPieceWiseLinearTaperDelegate.h>
```

Inheritance diagram for AAX_CPieceWiseLinearTaperDelegate< T, RealPrecision >:



Collaboration diagram for AAX_CPieceWiseLinearTaperDelegate< T, RealPrecision >:



14.30.1 Description

```
template<typename T, int32_t RealPrecision = 100>class AAX_CPieceWiseLinearTaperDelegate< T, RealPrecision >
```

A piece-wise linear taper conforming to [AAx_ITaperDelegate](#).

This taper spaces a parameter's real values in a piecewise linear fashion.

RealPrecision

In addition to its type templatization, this taper includes a precision template parameter. RealPrecision is a multiplier that works in conjunction with the round() function to limit the precision of the real values provided by this taper. For example, if RealPrecision is 1000, it will round to the closest 0.001 when doing any sort of value conversion. If RealPrecision is 1, it will round to the nearest integer. If RealPrecision is 1000000, it will round to the nearest 0.000001. This is particularly useful for preventing things like 1.9999999 truncating down to 1 instead of rounding up to 2.

To accomplish this behavior, the taper multiplies its unrounded parameter values by RealPrecision, rounds the result to the nearest valid value, then divides RealPrecision back out.

Rounding will be disabled if RealPrecision is set to a value less than 1

Public Member Functions

- [AAX_CPieceWiseLinearTaperDelegate](#) (const double *normalizedValues, const T *realValues, int32_t numValues)

Constructs a Piece-wise Linear Taper with paired normalized and real values.
- [AAX_CPieceWiseLinearTaperDelegate](#) (const [AAX_CPieceWiseLinearTaperDelegate](#) &other)

A copy constructor.
- [~AAX_CPieceWiseLinearTaperDelegate](#) ()

Destructor.
- virtual [AAX_CPieceWiseLinearTaperDelegate](#)< T, RealPrecision > * [Clone](#) () const [AAX_OVERRIDE](#)

Constructs and returns a copy of the taper delegate.
- virtual T [GetMinimumValue](#) () const [AAX_OVERRIDE](#)

Returns the taper's minimum real value.
- virtual T [GetMaximumValue](#) () const [AAX_OVERRIDE](#)

Returns the taper's maximum real value.
- virtual T [ConstrainRealValue](#) (T value) const [AAX_OVERRIDE](#)

Applies a constraint to the value and returns the constrained value.
- virtual T [NormalizedToReal](#) (double normalizedValue) const [AAX_OVERRIDE](#)

Converts a normalized value to a real value.
- virtual double [RealToNormalized](#) (T realValue) const [AAX_OVERRIDE](#)

Normalizes a real parameter value.

Protected Member Functions

- T [Round](#) (double iValue) const

14.30.2 Constructor & Destructor Documentation

14.30.2.1 template<typename T , int32_t RealPrecision> [AAX_CPieceWiseLinearTaperDelegate](#)< T, RealPrecision >::[AAX_CPieceWiseLinearTaperDelegate](#) (const double * *normalizedValues*, const T * *realValues*, int32_t *numValues*)

Constructs a Piece-wise Linear Taper with paired normalized and real values.

Note

The parameter's default value should lie within the min to max range.

Parameters

in	<i>normalizedValues</i>	is an array of the normalized values in sorted order. (make sure to include the full normalized range, 0.0-1.0 inclusive)
in	<i>realValues</i>	is an array of the corresponding real values to the normalized values passed in.
in	<i>numValues</i>	is the number of values that have been passed in (i.e. the element length of the other input arrays)

14.30.2.2 template<typename T , int32_t RealPrecision> AAX_CPieceWiseLinearTaperDelegate< T, RealPrecision >::AAX_CPieceWiseLinearTaperDelegate (const AAX_CPieceWiseLinearTaperDelegate< T, RealPrecision > & other)

14.30.2.3 template<typename T , int32_t RealPrecision> AAX_CPieceWiseLinearTaperDelegate< T, RealPrecision >::~AAX_CPieceWiseLinearTaperDelegate ()

14.30.3 Member Function Documentation

14.30.3.1 template<typename T , int32_t RealPrecision> AAX_CPieceWiseLinearTaperDelegate< T, RealPrecision > * AAX_CPieceWiseLinearTaperDelegate< T, RealPrecision >::Clone () const [virtual]

Constructs and returns a copy of the taper delegate.

In general, this method's implementation can use a simple copy constructor:

```
template <typename T>
AAX_CSubclassTaperDelegate<T>* AAX_CSubclassTaperDelegate<T>::Clone() const
{
    return new AAX_CSubclassTaperDelegate(*this);
}
```

Implements [AAX_ITaperDelegate< T >](#).

14.30.3.2 template<typename T, int32_t RealPrecision = 100> virtual T AAX_CPieceWiseLinearTaperDelegate< T, RealPrecision >::GetMinimumValue () const [inline], [virtual]

Returns the taper's minimum real value.

Implements [AAX_ITaperDelegate< T >](#).

14.30.3.3 template<typename T, int32_t RealPrecision = 100> virtual T AAX_CPieceWiseLinearTaperDelegate< T, RealPrecision >::GetMaximumValue () const [inline], [virtual]

Returns the taper's maximum real value.

Implements [AAX_ITaperDelegate< T >](#).

14.30.3.4 template<typename T , int32_t RealPrecision> T AAX_CPieceWiseLinearTaperDelegate< T, RealPrecision >::ConstrainRealValue (T value) const [virtual]

Applies a constraint to the value and returns the constrained value.

This method is useful if the taper requires a constraint beyond simple minimum and maximum real value limits.

Note

This is the function that should actually enforces the constraints in `NormalizeToReal()` and `RealToNormalized()`.

Parameters

in	value	The unconstrained value
----	-------	-------------------------

Implements [AAX_ITaperDelegate< T >](#).

```
14.30.3.5 template<typename T, int32_t RealPrecision> T AAX_CPieceWiseLinearTaperDelegate< T, RealPrecision
>::NormalizedToReal ( double normalizedValue ) const [virtual]
```

Converts a normalized value to a real value.

This is where the actual taper algorithm is implemented.

This function should perform the exact inverse of [RealToNormalized\(\)](#), to within the roundoff precision of the individual taper implementation.

Parameters

in	<i>normalizedValue</i>	The normalized value that will be converted
----	------------------------	---

Implements [AAX_ITaperDelegate< T >](#).

```
14.30.3.6 template<typename T, int32_t RealPrecision> double AAX_CPieceWiseLinearTaperDelegate< T,
RealPrecision >::RealToNormalized ( T realValue ) const [virtual]
```

Normalizes a real parameter value.

This is where the actual taper algorithm is implemented.

This function should perform the exact inverse of [NormalizedToReal\(\)](#), to within the roundoff precision of the individual taper implementation.

Parameters

in	<i>realValue</i>	The real parameter value that will be normalized
----	------------------	--

Implements [AAX_ITaperDelegate< T >](#).

```
14.30.3.7 template<typename T, int32_t RealPrecision> T AAX_CPieceWiseLinearTaperDelegate< T, RealPrecision
>::Round ( double iValue ) const [protected]
```

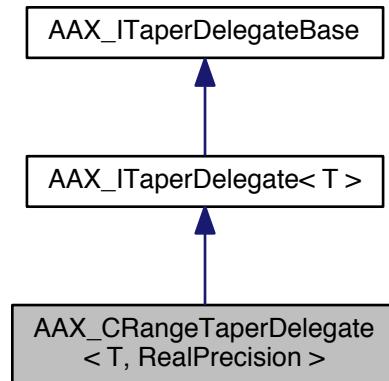
The documentation for this class was generated from the following file:

- [AAX_CPieceWiseLinearTaperDelegate.h](#)

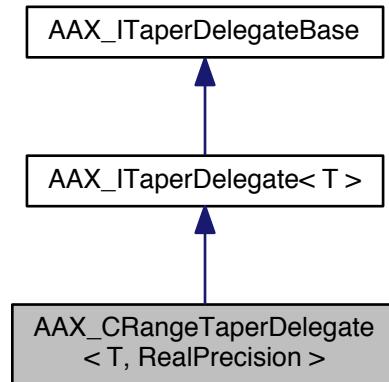
14.31 AAX_CRangeTaperDelegate< T, RealPrecision > Class Template Reference

```
#include <AAX_CRangeTaperDelegate.h>
```

Inheritance diagram for AAX_CRangeTaperDelegate< T, RealPrecision >:



Collaboration diagram for AAX_CRangeTaperDelegate< T, RealPrecision >:



14.31.1 Description

```
template<typename T, int32_t RealPrecision = 1000>class AAX_CRangeTaperDelegate< T, RealPrecision >
```

A piecewise-linear taper conforming to [AAX_ITaperDelegate](#).

This taper spaces a parameter's real values between its minimum and maximum using a series of linear regions to create the full mapping between the parameter's real and normalized values.

Here is an example of how this taper can be used:

```
float rangePoints[] = { 0.0, 1.0, 100.0, 1000.0, 2000.0 };
double rangeSteps[] = { 0.1, 1.0, 10.0, 25.0 }; // number of steps per range: 10, 99, 90, 40
```

```

const long cNumRanges = sizeof(rangeSteps)/sizeof(rangeSteps[0]);

long numSteps = 0;
for (int i = 0; i < cNumRanges; i++)
{
    numSteps += (rangePoints[i+1] - rangePoints[i]) / rangeSteps[i];
}

AAX_CRangeTaperDelegate<float> nonLinearTaper(rangePoints, rangeSteps,
                                               cNumRanges);

float controlValue = 1.5;

double normalized = nonLinearTaper.RealToNormalized(controlValue);
float real = nonLinearTaper.NormalizedToReal(normalized);

```

RealPrecision

In addition to its type templatization, this taper includes a precision template parameter. RealPrecision is a multiplier that works in conjunction with the round() function to limit the precision of the real values provided by this taper. For example, if RealPrecision is 1000, it will round to the closest 0.001 when doing any sort of value conversion. If RealPrecision is 1, it will round to the nearest integer. If RealPrecision is 1000000, it will round to the nearest 0.000001. This is particularly useful for preventing things like 1.9999999 truncating down to 1 instead of rounding up to 2.

To accomplish this behavior, the taper multiplies its unrounded parameter values by RealPrecision, rounds the result to the nearest valid value, then divides RealPrecision back out.

Rounding will be disabled if RealPrecision is set to a value less than 1

Public Member Functions

- **AAX_CRangeTaperDelegate** (T *range, double *rangesSteps, long numRanges, bool useSmartRounding=true)

Constructs a Range Taper with specified minimum and maximum values.
- **AAX_CRangeTaperDelegate** (const AAX_CRangeTaperDelegate &rhs)
- **AAX_CRangeTaperDelegate & operator=** (AAX_CRangeTaperDelegate &rhs)
- virtual **AAX_CRangeTaperDelegate**< T, RealPrecision > * **Clone** () const **AAX_OVERRIDE**

Constructs and returns a copy of the taper delegate.
- virtual T **GetMinimumValue** () const **AAX_OVERRIDE**

Returns the taper's minimum real value.
- virtual T **GetMaximumValue** () const **AAX_OVERRIDE**

Returns the taper's maximum real value.
- virtual T **ConstrainRealValue** (T value) const **AAX_OVERRIDE**

Applies a constraint to the value and returns the constrained value.
- virtual T **NormalizedToReal** (double normalizedValue) const **AAX_OVERRIDE**

Converts a normalized value to a real value.
- virtual double **RealToNormalized** (T realValue) const **AAX_OVERRIDE**

Normalizes a real parameter value.

Protected Member Functions

- T **Round** (double iValue) const
- T **SmartRound** (double value) const

14.31.2 Constructor & Destructor Documentation

14.31.2.1 template<typename T , int32_t RealPrecision> AAX_CRangeTaperDelegate< T, RealPrecision >::AAX_CRangeTaperDelegate (T * *range*, double * *rangesSteps*, long *numRanges*, bool *useSmartRounding* = true)

Constructs a Range Taper with specified minimum and maximum values.

Note

The parameter's default value should lie within the min to max range.

Parameters

in	<i>range</i>	An array of range endpoints along the taper's mapping range
in	<i>rangesSteps</i>	Step values for each region in the taper's stepwise-linear map. No values in this array may be zero.
in	<i>numRanges</i>	The total number of linear regions in the taper's map
in	<i>useSmartRounding</i>	

Todo Document *useSmartRounding* parameter

14.31.2.2 template<typename T , int32_t RealPrecision> AAX_CRangeTaperDelegate< T, RealPrecision >::AAX_CRangeTaperDelegate (const AAX_CRangeTaperDelegate< T, RealPrecision > & *rhs*)

14.31.3 Member Function Documentation

14.31.3.1 template<typename T , int32_t RealPrecision> AAX_CRangeTaperDelegate< T, RealPrecision > & AAX_CRangeTaperDelegate< T, RealPrecision >::operator= (AAX_CRangeTaperDelegate< T, RealPrecision > & *rhs*)

14.31.3.2 template<typename T , int32_t RealPrecision> AAX_CRangeTaperDelegate< T, RealPrecision > * AAX_CRangeTaperDelegate< T, RealPrecision >::Clone () const [virtual]

Constructs and returns a copy of the taper delegate.

In general, this method's implementation can use a simple copy constructor:

```
template <typename T>
AAX_CSubclassTaperDelegate<T>* AAX_CSubclassTaperDelegate<T>::Clone() const
{
    return new AAX_CSubclassTaperDelegate(*this);
}
```

Implements [AAX_ITaperDelegate< T >](#).

14.31.3.3 template<typename T, int32_t RealPrecision = 1000> virtual T AAX_CRangeTaperDelegate< T, RealPrecision >::GetMinimumValue () const [inline], [virtual]

Returns the taper's minimum real value.

Implements [AAX_ITaperDelegate< T >](#).

14.31.3.4 template<typename T, int32_t RealPrecision = 1000> virtual T AAX_CRangeTaperDelegate< T, RealPrecision >::GetMaximumValue () const [inline], [virtual]

Returns the taper's maximum real value.

Implements [AAX_ITaperDelegate< T >](#).

```
14.31.3.5 template<typename T , int32_t RealPrecision> T AAX_CRangeTaperDelegate< T, RealPrecision
>::ConstrainRealValue ( T value ) const [virtual]
```

Applies a constraint to the value and returns the constrained value.

This method is useful if the taper requires a constraint beyond simple minimum and maximum real value limits.

Note

This is the function that should actually enforces the constraints in NormalizeToReal() and [RealToNormalized\(\)](#).

Parameters

in	<i>value</i>	The unconstrained value
----	--------------	-------------------------

Implements [AAX_ITaperDelegate< T >](#).

```
14.31.3.6 template<typename T , int32_t RealPrecision> T AAX_CRangeTaperDelegate< T, RealPrecision
>::NormalizedToReal ( double normalizedValue ) const [virtual]
```

Converts a normalized value to a real value.

This is where the actual taper algorithm is implemented.

This function should perform the exact inverse of [RealToNormalized\(\)](#), to within the roundoff precision of the individual taper implementation.

Parameters

in	<i>normalizedValue</i>	The normalized value that will be converted
----	------------------------	---

Implements [AAX_ITaperDelegate< T >](#).

```
14.31.3.7 template<typename T , int32_t RealPrecision> double AAX_CRangeTaperDelegate< T, RealPrecision
>::RealToNormalized ( T realValue ) const [virtual]
```

Normalizes a real parameter value.

This is where the actual taper algorithm is implemented.

This function should perform the exact inverse of [NormalizedToReal\(\)](#), to within the roundoff precision of the individual taper implementation.

Parameters

in	<i>realValue</i>	The real parameter value that will be normalized
----	------------------	--

Implements [AAX_ITaperDelegate< T >](#).

```
14.31.3.8 template<typename T , int32_t RealPrecision> T AAX_CRangeTaperDelegate< T, RealPrecision >::Round (
double iValue ) const [protected]
```

```
14.31.3.9 template<typename T , int32_t RealPrecision> T AAX_CRangeTaperDelegate< T, RealPrecision
>::SmartRound ( double value ) const [protected]
```

[Todo](#) Document

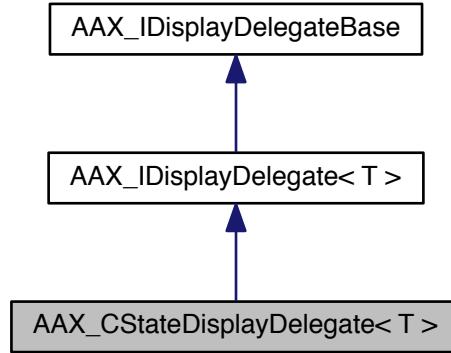
The documentation for this class was generated from the following file:

- [AAX_CRangeTaperDelegate.h](#)

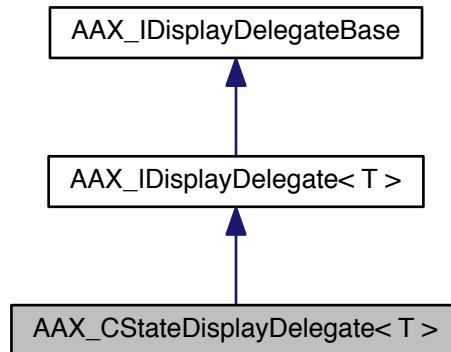
14.32 AAX_CStateDisplayDelegate< T > Class Template Reference

```
#include <AAX_CStateDisplayDelegate.h>
```

Inheritance diagram for AAX_CStateDisplayDelegate< T >:



Collaboration diagram for AAX_CStateDisplayDelegate< T >:



14.32.1 Description

```
template<typename T> class AAX_CStateDisplayDelegate< T >
```

A generic display format conforming to [AAX_IDisplayDelegate](#).

This display delegate is similar to [AAX_CNumberDisplayDelegate](#), but does not include precision or spacing templatizations.

Public Member Functions

- [AAX_CStateDisplayDelegate](#) (const char *iStateStrings[], T iMinState=0)

Constructor taking a vector of C strings.
- [AAX_CStateDisplayDelegate](#) (int32_t inNumStates, const char *iStateStrings[], T iMinState=0)

Constructor taking a vector of C strings.
- [AAX_CStateDisplayDelegate](#) (const std::vector< [AAX_IString](#) * > &iStateStrings, T iMinState=0)

Constructor taking a vector of [AAX_IString](#) objects.
- [AAX_CStateDisplayDelegate](#) (const [AAX_CStateDisplayDelegate](#) &other)
- virtual [AAX_IDisplayDelegate](#)< T > * [Clone](#) () const [AAX_OVERRIDE](#)

Constructs and returns a copy of the display delegate.
- virtual bool [ValueToString](#) (T value, [AAX_CString](#) *valueString) const [AAX_OVERRIDE](#)

Converts a real parameter value to a string representation.
- virtual bool [ValueToString](#) (T value, int32_t maxNumChars, [AAX_CString](#) *valueString) const [AAX_OVERRIDE](#)

Converts a real parameter value to a string representation using a size hint, useful for control surfaces and other character limited displays.
- virtual bool [StringToValue](#) (const [AAX_CString](#) &valueString, T *value) const [AAX_OVERRIDE](#)

Converts a string to a real parameter value.
- void [AddShortenedStrings](#) (const char *iStateStrings[], int iLength)
- bool [Compare](#) (const [AAX_CString](#) &valueString, const [AAX_CString](#) &stateString) const

14.32.2 Constructor & Destructor Documentation

14.32.2.1 template<typename T> [AAX_CStateDisplayDelegate](#)< T >::[AAX_CStateDisplayDelegate](#) (const char * iStateStrings[], T iMinState = 0) [explicit]

Constructor taking a vector of C strings.

Each state name will be copied into the display delegate; the C strings may be disposed after construction.

Note

iStateStrings must be NULL-terminated

14.32.2.2 template<typename T> [AAX_CStateDisplayDelegate](#)< T >::[AAX_CStateDisplayDelegate](#) (int32_t inNumStates, const char * iStateStrings[], T iMinState = 0) [explicit]

Constructor taking a vector of C strings.

Each state name will be copied into the display delegate; the C strings may be disposed after construction.

State strings will be copied into the display delegate until either a NULL pointer is encountered or inNumStates strings have been copied

14.32.2.3 template<typename T> [AAX_CStateDisplayDelegate](#)< T >::[AAX_CStateDisplayDelegate](#) (const std::vector< [AAX_IString](#) * > &iStateStrings, T iMinState = 0) [explicit]

Constructor taking a vector of [AAX_IString](#) objects.

Each [AAX_IString](#) will be copied into the display delegate and may be disposed after construction. The [AAX_IString](#) will not be mutated.

14.32.2.4 template<typename T> AAX_CStateDisplayDelegate<T>::AAX_CStateDisplayDelegate (const AAX_CStateDisplayDelegate<T> & other)

14.32.3 Member Function Documentation

14.32.3.1 template<typename T> AAX_IDisplayDelegate<T> * AAX_CStateDisplayDelegate<T>::Clone () const [virtual]

Constructs and returns a copy of the display delegate.

In general, this method's implementation can use a simple copy constructor:

```
template <typename T>
AAX_CSubclassDisplayDelegate<T>* AAX_CSubclassDisplayDelegate<T>::Clone () const
{
    return new AAX_CSubclassDisplayDelegate(*this);
}
```

Implements [AAX_IDisplayDelegate< T >](#).

14.32.3.2 template<typename T> bool AAX_CStateDisplayDelegate<T>::ValueToString (T value, AAX_CString * valueString) const [virtual]

Converts a real parameter value to a string representation.

Parameters

in	value	The real parameter value that will be converted
out	valueString	A string corresponding to value

Return values

true	The string conversion was successful
false	The string conversion was unsuccessful

Implements [AAX_IDisplayDelegate< T >](#).

14.32.3.3 template<typename T> bool AAX_CStateDisplayDelegate<T>::ValueToString (T value, int32_t maxNumChars, AAX_CString * valueString) const [virtual]

Converts a real parameter value to a string representation using a size hint, useful for control surfaces and other character limited displays.

Parameters

in	value	The real parameter value that will be converted
in	maxNumChars	Size hint for the desired maximum number of characters in the string (not including null termination)
out	valueString	A string corresponding to value

Return values

true	The string conversion was successful
false	The string conversion was unsuccessful

Implements [AAX_IDisplayDelegate< T >](#).

14.32.3.4 template<typename T> bool AAX_CStateDisplayDelegate<T>::StringToValue (const AAX_CString & valueString, T * value) const [virtual]

Converts a string to a real parameter value.

Parameters

in	<i>valueString</i>	The string that will be converted
out	<i>value</i>	The real parameter value corresponding to valueString

Return values

<i>true</i>	The string conversion was successful
<i>false</i>	The string conversion was unsuccessful

Implements [AAX_IDisplayDelegate< T >](#).

14.32.3.5 template<typename T> void AAX_CStateDisplayDelegate< T >::AddShortenedStrings (const char * *iStateStrings[]*, int *iLength*)

14.32.3.6 template<typename T> bool AAX_CStateDisplayDelegate< T >::Compare (const AAX_CString & *valueString*, const AAX_CString & *stateString*) const

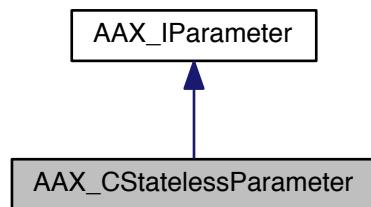
The documentation for this class was generated from the following file:

- [AAX_CStateDisplayDelegate.h](#)

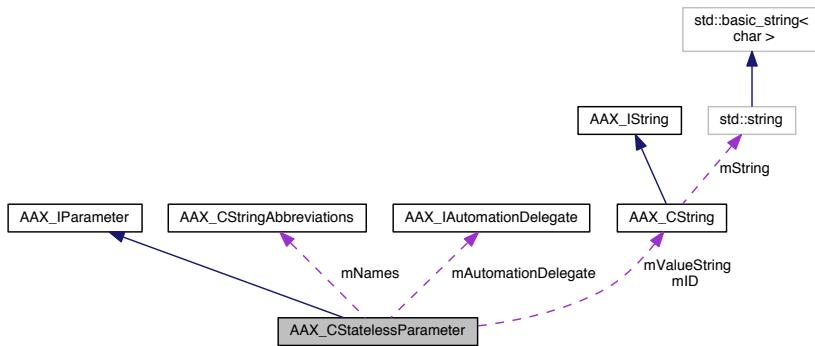
14.33 AAX_CStatelessParameter Class Reference

```
#include <AAX_CParameter.h>
```

Inheritance diagram for AAX_CStatelessParameter:



Collaboration diagram for AAX_CStatelessParameter:



14.33.1 Description

A stateless parameter implementation.

This can be useful for mapping event triggers to control surface buttons or to GUI switches.

Public Member Functions

- `AAX_CStatelessParameter (AAX_CParamID identifier, const AAX_IString &name, const AAX_IString &inValueString)`
- `AAX_CStatelessParameter (const AAX_IString &identifier, const AAX_IString &name, const AAX_IString &inValueString)`
- virtual `~AAX_CStatelessParameter ()`
- virtual `AAX_IParParameterValue * CloneValue () const`
 - Clone the parameter's value to a new AAX_IParParameterValue object.*
- virtual void `SetType (AAX_EParameterType)`
 - Sets the type of this parameter.*
- virtual `AAX_EParameterType GetType () const`
 - Returns the type of this parameter as an AAX_EParameterType.*
- virtual void `SetOrientation (AAX_EParameterOrientation)`
 - Sets the orientation of this parameter.*
- virtual `AAX_EParameterOrientation GetOrientation () const`
 - Returns the orientation of this parameter.*
- virtual void `SetTaperDelegate (AAX_ITaperDelegateBase &, bool)`
 - Sets the parameter's taper delegate.*
- virtual void `SetDisplayDelegate (AAX_IDisplayDelegateBase &)`
 - Sets the parameter's display delegate.*

Identification methods

- virtual `AAX_CParamID Identifier () const`
 - Returns the parameter's unique identifier.*
- virtual void `SetName (const AAX_CString &name)`
 - Sets the parameter's display name.*
- virtual const `AAX_CString & Name () const`
 - Returns the parameter's display name.*

- virtual void `AddShortenedName` (const `AAX_CString` &name)
Sets the parameter's shortened display name.
- virtual const `AAX_CString & ShortenedName` (int32_t iNumCharacters) const
Returns the parameter's shortened display name.
- virtual void `ClearShortenedNames` ()
Clears the internal list of shortened display names.

Automation methods

- virtual bool `Automatable` () const
Returns true if the parameter is automatable, false if it is not.
- virtual void `SetAutomationDelegate` (`AAX_IAutomationDelegate` *iAutomationDelegate)
Sets the automation delegate (if one is required)
- virtual void `Touch` ()
Signals the automation system that a control has been touched.
- virtual void `Release` ()
Signals the automation system that a control has been released.

Taper methods

- virtual void `SetNormalizedValue` (double)
Sets a parameter value using its normalized representation.
- virtual double `GetNormalizedValue` () const
Returns the normalized representation of the parameter's current real value.
- virtual void `SetNormalizedDefaultValue` (double)
Sets the parameter's default value using its normalized representation.
- virtual double `GetNormalizedDefaultValue` () const
Returns the normalized representation of the parameter's real default value.
- virtual void `SetToDefaultValue` ()
Restores the state of this parameter to its default value.
- virtual void `SetNumberOfSteps` (uint32_t)
Sets the number of discrete steps for this parameter.
- virtual uint32_t `GetNumberOfSteps` () const
Returns the number of discrete steps used by the parameter.
- virtual uint32_t `GetStepValue` () const
Returns the current step for the current value of the parameter.
- virtual double `GetNormalizedValueFromStep` (uint32_t) const
Returns the normalized value for a given step.
- virtual uint32_t `GetStepValueFromNormalizedValue` (double) const
Returns the step value for a normalized value of the parameter.
- virtual void `SetStepValue` (uint32_t)
Returns the current step for the current value of the parameter.

Display methods

This functionality is most often used by GUIs, but can also be useful for state serialization.

- virtual bool `GetValueString` (`AAX_CString` *valueString) const
Serializes the parameter value into a string.
- virtual bool `GetValueString` (int32_t, `AAX_CString` *valueString) const
Serializes the parameter value into a string, size hint included.
- virtual bool `GetNormalizedValueFromBool` (bool, double *normalizedValue) const
Converts a bool to a normalized parameter value.
- virtual bool `GetNormalizedValueFromInt32` (int32_t, double *normalizedValue) const
Converts an integer to a normalized parameter value.
- virtual bool `GetNormalizedValueFromFloat` (float, double *normalizedValue) const
Converts a float to a normalized parameter value.
- virtual bool `GetNormalizedValueFromDouble` (double, double *normalizedValue) const
Converts a double to a normalized parameter value.

- virtual bool [GetNormalizedValueFromString](#) (const [AAX_CString](#) &, double *normalizedValue) const
Converts a given string to a normalized parameter value.
- virtual bool [GetBoolFromNormalizedValue](#) (double, bool *value) const
Converts a normalized parameter value to a bool representing the corresponding real value.
- virtual bool [GetInt32FromNormalizedValue](#) (double, int32_t *) const
Converts a normalized parameter value to an integer representing the corresponding real value.
- virtual bool [GetFloatFromNormalizedValue](#) (double, float *) const
Converts a normalized parameter value to a float representing the corresponding real value.
- virtual bool [GetDoubleFromNormalizedValue](#) (double, double *) const
Converts a normalized parameter value to a double representing the corresponding real value.
- virtual bool [GetStringFromNormalizedValue](#) (double, [AAX_CString](#) &valueString) const
Converts a normalized parameter value to a string representing the corresponding real value.
- virtual bool [GetStringFromNormalizedValue](#) (double normalizedValue, int32_t, [AAX_CString](#) &valueString) const
Converts a normalized parameter value to a string representing the corresponding real, size hint included. value.
- virtual bool [SetValueFromString](#) (const [AAX_CString](#) &newValueString)
Converts a string to a real parameter value and sets the parameter to this value.

Typed accessors

- virtual bool [GetValueAsBool](#) (bool *value) const
Retrieves the parameter's value as a bool.
- virtual bool [GetValueAsInt32](#) (int32_t *) const
Retrieves the parameter's value as an int32_t.
- virtual bool [GetValueAsFloat](#) (float *) const
Retrieves the parameter's value as a float.
- virtual bool [GetValueAsDouble](#) (double *) const
Retrieves the parameter's value as a double.
- virtual bool [GetValueAsString](#) ([AAX_IString](#) *) const
Retrieves the parameter's value as a string.
- virtual bool [SetValueWithBool](#) (bool)
Sets the parameter's value as a bool.
- virtual bool [SetValueWithInt32](#) (int32_t)
Sets the parameter's value as an int32_t.
- virtual bool [SetValueWithFloat](#) (float)
Sets the parameter's value as a float.
- virtual bool [SetValueWithDouble](#) (double)
Sets the parameter's value as a double.
- virtual bool [SetValueWithString](#) (const [AAX_IString](#) &value)
Sets the parameter's value as a string.

Host interface methods

- virtual void [UpdateNormalizedValue](#) (double)
Sets the parameter's state given a normalized value.

Protected Attributes

- [AAX_CStringAbbreviations](#) mNames
- [AAX_CString](#) mID
- [AAX_IAutomationDelegate](#) * mAutomationDelegate
- [AAX_CString](#) mValueString

14.33.2 Constructor & Destructor Documentation

14.33.2.1 `AAX_CStatelessParameter::AAX_CStatelessParameter(AAX_CParamID identifier, const AAX_IString & name, const AAX_IString & inValueString) [inline]`

14.33.2.2 `AAX_CStatelessParameter::AAX_CStatelessParameter(const AAX_IString & identifier, const AAX_IString & name, const AAX_IString & inValueString) [inline]`

14.33.2.3 `virtual AAX_CStatelessParameter::~AAX_CStatelessParameter() [inline], [virtual]`

14.33.3 Member Function Documentation

14.33.3.1 `virtual AAX_IPParameterValue* AAX_CStatelessParameter::CloneValue() const [inline], [virtual]`

Clone the parameter's value to a new [AAX_IPParameterValue](#) object.

The returned object is independent from the [AAX_IPParameter](#). For example, changing the state of the returned object will not result in a change to the original [AAX_IPParameter](#).

Implements [AAX_IPParameter](#).

14.33.3.2 `virtual AAX_CParamID AAX_CStatelessParameter::Identifier() const [inline], [virtual]`

Returns the parameter's unique identifier.

This unique ID is used by the [Parameter Manager](#) and by outside applications to uniquely identify and target control messages. This value may not be changed after the parameter has been constructed.

Implements [AAX_IPParameter](#).

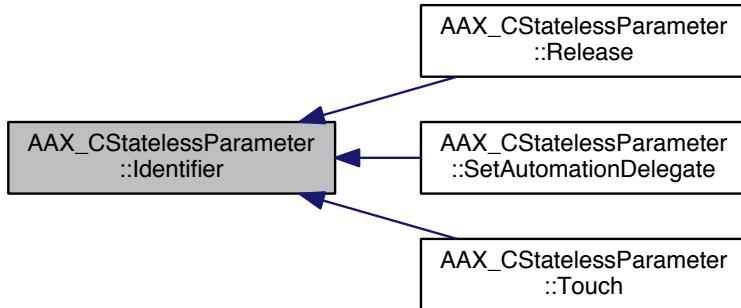
References `AAX_CString::CString()`, and `mID`.

Referenced by `Release()`, `SetAutomationDelegate()`, and `Touch()`.

Here is the call graph for this function:



Here is the caller graph for this function:



14.33.3.3 virtual void AAX_CStatelessParameter::SetName (const AAX_CString & name) [inline], [virtual]

Sets the parameter's display name.

This name is used for display only, it is not used for indexing or identifying the parameter. This name may be changed after the parameter has been created, but display name changes may not be recognized by all AAX hosts.

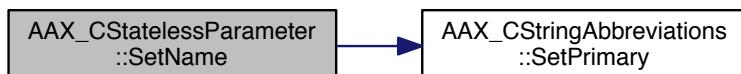
Parameters

in	name	Display name that will be assigned to the parameter
----	------	---

Implements [AAX_IParameter](#).

References mNames, and [AAX_CStringAbbreviations::SetPrimary\(\)](#).

Here is the call graph for this function:



14.33.3.4 virtual const AAX_CString& AAX_CStatelessParameter::Name () const [inline], [virtual]

Returns the parameter's display name.

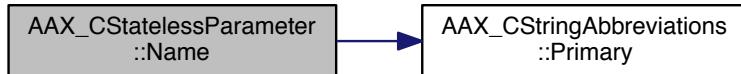
Note

This method returns a const reference in order to prevent a string copy. Do not cast away the const to change this value.

Implements [AAX_IParameter](#).

References mNames, and [AAX_CStringAbbreviations::Primary\(\)](#).

Here is the call graph for this function:



14.33.3.5 virtual void AAX_CStatelessParameter::AddShortenedName (const AAX_CString & name) [inline], [virtual]

Sets the parameter's shortened display name.

This name is used for display only, it is not used for indexing or identifying the parameter. These names show up when the host asks for shorter length parameter names for display on Control Surfaces or other string length constrained situations.

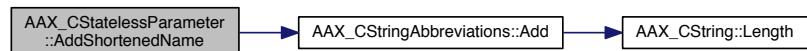
Parameters

in	name	Shortened display names that will be assigned to the parameter
----	------	--

Implements [AAX_IParameter](#).

References [AAX_CStringAbbreviations::Add\(\)](#), and [mNames](#).

Here is the call graph for this function:



14.33.3.6 virtual const AAX_CString& AAX_CStatelessParameter::ShortenedName (int32_t iNumCharacters) const [inline], [virtual]

Returns the parameter's shortened display name.

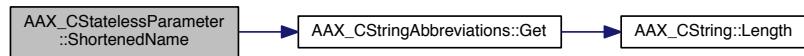
Note

This method returns a const reference in order to prevent a string copy. Do not cast away the const to change this value.

Implements [AAX_IParameter](#).

References [AAX_CStringAbbreviations::Get\(\)](#), and [mNames](#).

Here is the call graph for this function:



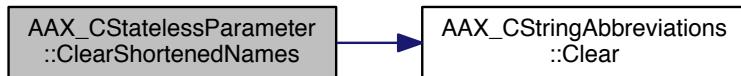
14.33.3.7 virtual void AAX_CStatelessParameter::ClearShortenedNames() [inline], [virtual]

Clears the internal list of shortened display names.

Implements [AAX_IParameter](#).

References [AAX_CStringAbbreviations::Clear\(\)](#), and [mNames](#).

Here is the call graph for this function:



14.33.3.8 virtual bool AAX_CStatelessParameter::Automatable() const [inline], [virtual]

Returns true if the parameter is automatable, false if it is not.

Note

Subclasses that return true in this method must support host-based automation.

Implements [AAX_IParameter](#).

14.33.3.9 virtual void AAX_CStatelessParameter::SetAutomationDelegate(AAX_IAutomationDelegate * iAutomationDelegate) [inline], [virtual]

Sets the automation delegate (if one is required)

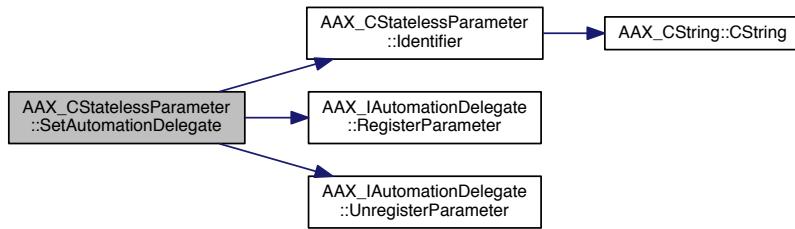
Parameters

in	<i>iAutomationDelegate</i>	A reference to the parameter manager's automation delegate interface
----	----------------------------	--

Implements [AAX_IParameter](#).

References Identifier(), mAutomationDelegate, [AAX_IAutomationDelegate::RegisterParameter\(\)](#), and [AAX_IAutomationDelegate::UnregisterParameter\(\)](#).

Here is the call graph for this function:



14.33.3.10 virtual void AAX_CStatelessParameter::Touch() [inline], [virtual]

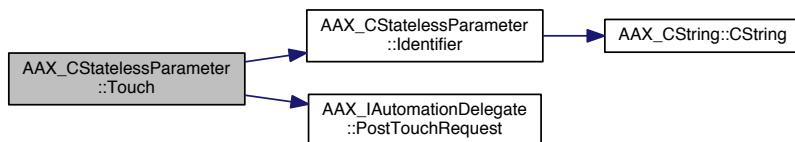
Signals the automation system that a control has been touched.

Call this method in response to GUI events that begin editing, such as a mouse down. After this method has been called you are free to call [SetNormalizedValue\(\)](#) as much as you need, e.g. in order to respond to subsequent mouse moved events. Call [Release\(\)](#) to free the parameter for updates from other controls.

Implements [AAX_IParameter](#).

References Identifier(), mAutomationDelegate, and [AAX_IAutomationDelegate::PostTouchRequest\(\)](#).

Here is the call graph for this function:



14.33.3.11 virtual void AAX_CStatelessParameter::Release() [inline], [virtual]

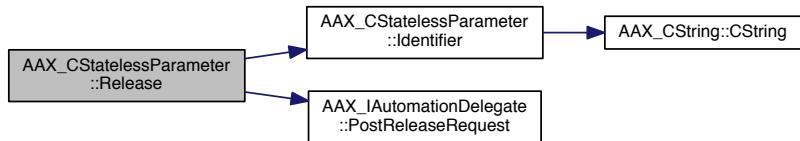
Signals the automation system that a control has been released.

Call this method in response to GUI events that complete editing, such as a mouse up. Once this method has been called you should not call [SetNormalizedValue\(\)](#) again until after the next call to [Touch\(\)](#).

Implements [AAX_IParameter](#).

References Identifier(), mAutomationDelegate, and [AAX_IAutomationDelegate::PostReleaseRequest\(\)](#).

Here is the call graph for this function:



14.33.3.12 `virtual void AAX_CStatelessParameter::SetNormalizedValue(double newNormalizedValue) [inline], [virtual]`

Sets a parameter value using it's normalized representation.

For more information regarding normalized values, see [Parameter Manager](#)

Parameters

in	<i>newNormalizedValue</i>	New value (normalized) to which the parameter will be set
----	---------------------------	---

Implements [AAX_IParameter](#).

14.33.3.13 `virtual double AAX_CStatelessParameter::GetNormalizedValue() const [inline], [virtual]`

Returns the normalized representation of the parameter's current real value.

Implements [AAX_IParameter](#).

14.33.3.14 `virtual void AAX_CStatelessParameter::SetNormalizedDefaultValue(double normalizedDefault) [inline], [virtual]`

Sets the parameter's default value using its normalized representation.

Implements [AAX_IParameter](#).

14.33.3.15 `virtual double AAX_CStatelessParameter::GetNormalizedDefaultValue() const [inline], [virtual]`

Returns the normalized representation of the parameter's real default value.

Implements [AAX_IParameter](#).

14.33.3.16 `virtual void AAX_CStatelessParameter::SetToDefaultValue() [inline], [virtual]`

Restores the state of this parameter to its default value.

Implements [AAX_IParameter](#).

14.33.3.17 `virtual void AAX_CStatelessParameter::SetNumberOfSteps(uint32_t numSteps) [inline], [virtual]`

Sets the number of discrete steps for this parameter.

Stepped parameter values are useful for discrete parameters and for "jumping" events such as mouse wheels, page up/down, etc. The parameter's step size is used to specify the coarseness of those changes.

Note

numSteps MUST be greater than zero. All other values may be considered an error by the host.

Parameters

in	<i>numSteps</i>	The number of steps that the parameter will use
----	-----------------	---

Implements [AAX_IParameter](#).

14.33.3.18 virtual uint32_t AAX_CStatelessParameter::GetNumberOfSteps() const [inline], [virtual]

Returns the number of discrete steps used by the parameter.

See [SetNumberOfSteps\(\)](#) for more information about parameter steps.

Implements [AAX_IParameter](#).

14.33.3.19 virtual uint32_t AAX_CStatelessParameter::GetStepValue() const [inline], [virtual]

Returns the current step for the current value of the parameter.

See [SetNumberOfSteps\(\)](#) for more information about parameter steps.

Implements [AAX_IParameter](#).

14.33.3.20 virtual double AAX_CStatelessParameter::GetNormalizedValueFromStep(uint32_t *iStep*) const [inline], [virtual]

Returns the normalized value for a given step.

See [SetNumberOfSteps\(\)](#) for more information about parameter steps.

Implements [AAX_IParameter](#).

14.33.3.21 virtual uint32_t AAX_CStatelessParameter::GetStepValueFromNormalizedValue(double *normalizedValue*) const [inline], [virtual]

Returns the step value for a normalized value of the parameter.

See [SetNumberOfSteps\(\)](#) for more information about parameter steps.

Implements [AAX_IParameter](#).

14.33.3.22 virtual void AAX_CStatelessParameter::SetStepValue(uint32_t *iStep*) [inline], [virtual]

Returns the current step for the current value of the parameter.

See [SetNumberOfSteps\(\)](#) for more information about parameter steps.

Implements [AAX_IParameter](#).

14.33.3.23 virtual bool AAX_CStatelessParameter::GetValueString(AAX_CString * *valueString*) const [inline], [virtual]

Serializes the parameter value into a string.

Parameters

<code>out</code>	<code>valueString</code>	A string representing the parameter's real value
------------------	--------------------------	--

Return values

<code>true</code>	The string conversion was successful
<code>false</code>	The string conversion was unsuccessful

Implements [AAX_IParameter](#).

References `mValueString`.

```
14.33.3.24 virtual bool AAX_CStatelessParameter::GetValueString( int32_t iMaxNumChars, AAX_CString * valueString )
const [inline], [virtual]
```

Serializes the parameter value into a string, size hint included.

Parameters

<code>in</code>	<code>iMaxNumChars</code>	A size hint for the size of the string being requested. Useful for control surfaces and other limited area text fields. (make sure that size of desired string also has room for null termination)
<code>out</code>	<code>valueString</code>	A string representing the parameter's real value

Return values

<code>true</code>	The string conversion was successful
<code>false</code>	The string conversion was unsuccessful

Implements [AAX_IParameter](#).

References `GetValueString()`.

Referenced by `GetValueString()`.

Here is the call graph for this function:



Here is the caller graph for this function:



14.33.3.25 `virtual bool AAX_CStatelessParameter::GetNormalizedValueFromBool (bool value, double * normalizedValue) const [inline], [virtual]`

Converts a bool to a normalized parameter value.

Parameters

in	<i>value</i>	A value for the parameter
out	<i>normalizedValue</i>	The normalized parameter value associated with value

Return values

<i>true</i>	The value conversion was successful
<i>false</i>	The value conversion was unsuccessful

Implements [AAX_IParameter](#).

14.33.3.26 `virtual bool AAX_CStatelessParameter::GetNormalizedValueFromInt32 (int32_t value, double * normalizedValue) const [inline], [virtual]`

Converts an integer to a normalized parameter value.

Parameters

in	<i>value</i>	A value for the parameter
out	<i>normalizedValue</i>	The normalized parameter value associated with value

Return values

<i>true</i>	The value conversion was successful
<i>false</i>	The value conversion was unsuccessful

Implements [AAX_IParameter](#).

14.33.3.27 `virtual bool AAX_CStatelessParameter::GetNormalizedValueFromFloat (float value, double * normalizedValue) const [inline], [virtual]`

Converts a float to a normalized parameter value.

Parameters

in	<i>value</i>	A value for the parameter
out	<i>normalizedValue</i>	The normalized parameter value associated with value

Return values

<i>true</i>	The value conversion was successful
<i>false</i>	The value conversion was unsuccessful

Implements [AAX_IParameter](#).

14.33.3.28 `virtual bool AAX_CStatelessParameter::GetNormalizedValueFromDouble (double value, double * normalizedValue) const [inline], [virtual]`

Converts a double to a normalized parameter value.

Parameters

in	<i>value</i>	A value for the parameter
out	<i>normalizedValue</i>	The normalized parameter value associated with value

Return values

<i>true</i>	The value conversion was successful
<i>false</i>	The value conversion was unsuccessful

Implements [AAX_IParameter](#).

14.33.3.29 `virtual bool AAX_CStatelessParameter::GetNormalizedValueFromString (const AAX_CString & valueString, double * normalizedValue) const [inline], [virtual]`

Converts a given string to a normalized parameter value.

Parameters

in	<i>valueString</i>	A string representing a possible real value for the parameter
out	<i>normalizedValue</i>	The normalized parameter value associated with valueString

Return values

<i>true</i>	The string conversion was successful
<i>false</i>	The string conversion was unsuccessful

Implements [AAX_IParameter](#).

14.33.3.30 `virtual bool AAX_CStatelessParameter::GetBoolFromNormalizedValue (double normalizedValue, bool * value) const [inline], [virtual]`

Converts a normalized parameter value to a bool representing the corresponding real value.

Parameters

in	<i>normalizedValue</i>	The normalized value to convert
out	<i>value</i>	The converted value. Set only if conversion is successful.

Return values

<i>true</i>	The conversion to bool was successful
<i>false</i>	The conversion to bool was unsuccessful

Implements [AAX_IParameter](#).

14.33.3.31 virtual bool AAX_CStatelessParameter::GetInt32FromNormalizedValue (double *normalizedValue*, int32_t * *value*)
const [inline], [virtual]

Converts a normalized parameter value to an integer representing the corresponding real value.

Parameters

in	<i>normalizedValue</i>	The normalized value to convert
out	<i>value</i>	The converted value. Set only if conversion is successful.

Return values

<i>true</i>	The conversion to int32_t was successful
<i>false</i>	The conversion to int32_t was unsuccessful

Implements [AAX_IParameter](#).

14.33.3.32 virtual bool AAX_CStatelessParameter::GetFloatFromNormalizedValue (double *normalizedValue*, float * *value*)
const [inline], [virtual]

Converts a normalized parameter value to a float representing the corresponding real value.

Parameters

in	<i>normalizedValue</i>	The normalized value to convert
out	<i>value</i>	The converted value. Set only if conversion is successful.

Return values

<i>true</i>	The conversion to float was successful
<i>false</i>	The conversion to float was unsuccessful

Implements [AAX_IParameter](#).

14.33.3.33 virtual bool AAX_CStatelessParameter::GetDoubleFromNormalizedValue (double *normalizedValue*, double * *value*)
const [inline], [virtual]

Converts a normalized parameter value to a double representing the corresponding real value.

Parameters

in	<i>normalizedValue</i>	The normalized value to convert
out	<i>value</i>	The converted value. Set only if conversion is successful.

Return values

<i>true</i>	The conversion to double was successful
<i>false</i>	The conversion to double was unsuccessful

Implements [AAX_IParameter](#).

14.33.3.34 virtual bool AAX_CStatelessParameter::GetStringFromNormalizedValue (double *normalizedValue*, AAX_CString & *valueString*)
const [inline], [virtual]

Converts a normalized parameter value to a string representing the corresponding real value.

Parameters

in	<i>normalizedValue</i>	A normalized parameter value
out	<i>valueString</i>	A string representing the parameter value associated with <i>normalizedValue</i>

Return values

<i>true</i>	The string conversion was successful
<i>false</i>	The string conversion was unsuccessful

Implements [AAX_IParameter](#).

References *mValueString*.

14.33.3.35 virtual bool AAX_CStatelessParameter::GetStringFromNormalizedValue (double *normalizedValue*, int32_t *iMaxNumChars*, AAX_CString & *valueString*) const [inline], [virtual]

Converts a normalized parameter value to a string representing the corresponding real, size hint included. value.

Parameters

in	<i>normalizedValue</i>	A normalized parameter value
in	<i>iMaxNumChars</i>	A size hint for the size of the string being requested. Useful for control surfaces and other limited area text fields. (make sure that size of desired string also has room for null termination)
out	<i>valueString</i>	A string representing the parameter value associated with <i>normalizedValue</i>

Return values

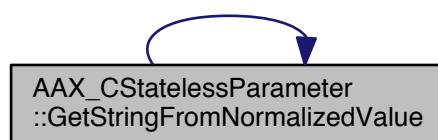
<i>true</i>	The string conversion was successful
<i>false</i>	The string conversion was unsuccessful

Implements [AAX_IParameter](#).

References *GetStringFromNormalizedValue()*.

Referenced by *GetStringFromNormalizedValue()*.

Here is the call graph for this function:



Here is the caller graph for this function:



14.33.3.36 `virtual bool AAX_CStatelessParameter::SetValueFromString (const AAX_CString & newValueString) [inline], [virtual]`

Converts a string to a real parameter value and sets the parameter to this value.

Parameters

in	<code>newValueString</code>	A string representing the parameter's new real value
----	-----------------------------	--

Return values

<code>true</code>	The string conversion was successful
<code>false</code>	The string conversion was unsuccessful

Implements [AAX_IParameter](#).

References `mValueString`.

14.33.3.37 `virtual bool AAX_CStatelessParameter::GetValueAsBool (bool * value) const [inline], [virtual]`

Retrieves the parameter's value as a bool.

Parameters

out	<code>value</code>	The parameter's real value. Set only if conversion is successful.
-----	--------------------	---

Return values

<code>true</code>	The conversion to bool was successful
<code>false</code>	The conversion to bool was unsuccessful

Implements [AAX_IParameter](#).

14.33.3.38 `virtual bool AAX_CStatelessParameter::GetValueAsInt32 (int32_t * value) const [inline], [virtual]`

Retrieves the parameter's value as an `int32_t`.

Parameters

out	<code>value</code>	The parameter's real value. Set only if conversion is successful.
-----	--------------------	---

Return values

<i>true</i>	The conversion to int32_t was successful
<i>false</i>	The conversion to int32_t was unsuccessful

Implements [AAX_IParameter](#).

14.33.3.39 virtual bool AAX_CStatelessParameter::GetValueAsFloat (float * *value*) const [inline], [virtual]

Retrieves the parameter's value as a float.

Parameters

<i>out</i>	<i>value</i>	The parameter's real value. Set only if conversion is successful.
------------	--------------	---

Return values

<i>true</i>	The conversion to float was successful
<i>false</i>	The conversion to float was unsuccessful

Implements [AAX_IParameter](#).

14.33.3.40 virtual bool AAX_CStatelessParameter::GetValueAsDouble (double * *value*) const [inline], [virtual]

Retrieves the parameter's value as a double.

Parameters

<i>out</i>	<i>value</i>	The parameter's real value. Set only if conversion is successful.
------------	--------------	---

Return values

<i>true</i>	The conversion to double was successful
<i>false</i>	The conversion to double was unsuccessful

Implements [AAX_IParameter](#).

14.33.3.41 virtual bool AAX_CStatelessParameter::GetValueAsString (AAX_IString * *value*) const [inline], [virtual]

Retrieves the parameter's value as a string.

Parameters

<i>out</i>	<i>value</i>	The parameter's real value. Set only if conversion is successful.
------------	--------------	---

Return values

<i>true</i>	The conversion to string was successful
<i>false</i>	The conversion to string was unsuccessful

Implements [AAX_IParameter](#).

14.33.3.42 virtual bool AAX_CStatelessParameter::SetValueWithBool (bool *value*) [inline], [virtual]

Sets the parameter's value as a bool.

Parameters

<code>out</code>	<code>value</code>	The parameter's real value. Set only if conversion is successful.
------------------	--------------------	---

Return values

<code>true</code>	The conversion from bool was successful
<code>false</code>	The conversion from bool was unsuccessful

Implements [AAX_IParameter](#).

14.33.3.43 virtual bool AAX_CStatelessParameter::SetValueWithInt32 (int32_t value) [inline], [virtual]

Sets the parameter's value as an int32_t.

Parameters

<code>out</code>	<code>value</code>	The parameter's real value. Set only if conversion is successful.
------------------	--------------------	---

Return values

<code>true</code>	The conversion from int32_t was successful
<code>false</code>	The conversion from int32_t was unsuccessful

Implements [AAX_IParameter](#).

14.33.3.44 virtual bool AAX_CStatelessParameter::SetValueWithFloat (float value) [inline], [virtual]

Sets the parameter's value as a float.

Parameters

<code>out</code>	<code>value</code>	The parameter's real value. Set only if conversion is successful.
------------------	--------------------	---

Return values

<code>true</code>	The conversion from float was successful
<code>false</code>	The conversion from float was unsuccessful

Implements [AAX_IParameter](#).

14.33.3.45 virtual bool AAX_CStatelessParameter::SetValueWithDouble (double value) [inline], [virtual]

Sets the parameter's value as a double.

Parameters

<code>out</code>	<code>value</code>	The parameter's real value. Set only if conversion is successful.
------------------	--------------------	---

Return values

<code>true</code>	The conversion from double was successful
<code>false</code>	The conversion from double was unsuccessful

Implements [AAX_IParameter](#).

14.33.3.46 virtual bool AAX_CStatelessParameter::SetValueWithString (const AAX_IString & value) [inline], [virtual]

Sets the parameter's value as a string.

Parameters

<code>out</code>	<code>value</code>	The parameter's real value. Set only if conversion is successful.
------------------	--------------------	---

Return values

<code>true</code>	The conversion from string was successful
<code>false</code>	The conversion from string was unsuccessful

Implements [AAX_IParameter](#).

References `mValueString`.

14.33.3.47 `virtual void AAX_CStatelessParameter::SetType(AAX_EParameterType iControlType) [inline], [virtual]`

Sets the type of this parameter.

See [GetType](#) for use cases

Parameters

<code>in</code>	<code>iControlType</code>	The parameter's new type as an <code>AAX_EParameterType</code>
-----------------	---------------------------	--

Implements [AAX_IParameter](#).

14.33.3.48 `virtual AAX_EParameterType AAX_CStatelessParameter::GetType() const [inline], [virtual]`

Returns the type of this parameter as an `AAX_EParameterType`.

Todo Document use cases for control type

Implements [AAX_IParameter](#).

References `AAX_eParameterType_Discrete`.

14.33.3.49 `virtual void AAX_CStatelessParameter::SetOrientation(AAX_EParameterOrientation iOrientation) [inline], [virtual]`

Sets the orientation of this parameter.

Parameters

<code>in</code>	<code>iOrientation</code>	The parameter's new orientation
-----------------	---------------------------	---------------------------------

Implements [AAX_IParameter](#).

14.33.3.50 `virtual AAX_EParameterOrientation AAX_CStatelessParameter::GetOrientation() const [inline], [virtual]`

Returns the orientation of this parameter.

Implements [AAX_IParameter](#).

References `AAX_eParameterOrientation_Default`.

14.33.3.51 `virtual void AAX_CStatelessParameter::SetTaperDelegate(AAX_ITaperDelegateBase & inTaperDelegate, bool inPreserveValue) [inline], [virtual]`

Sets the parameter's taper delegate.

Parameters

in	<i>inTaperDelegate</i>	A reference to the parameter's new taper delegate
in	<i>inPreserveValue</i>	

Todo Document this parameter

Implements [AAX_IParameter](#).

14.33.3.52 `virtual void AAX_CStatelessParameter::SetDisplayDelegate (AAX_IDisplayDelegateBase & inDisplayDelegate) [inline], [virtual]`

Sets the parameter's display delegate.

Parameters

in	<i>inDisplayDelegate</i>	A reference to the parameter's new display delegate
----	--------------------------	---

Implements [AAX_IParameter](#).

14.33.3.53 `virtual void AAX_CStatelessParameter::UpdateNormalizedValue (double newNormalizedValue) [inline], [virtual]`

Sets the parameter's state given a normalized value.

This is the second half of the parameter setting operation that is initiated with a call to `SetValue()`. Parameters should not be set directly using this method; instead, use `SetValue()`.

Parameters

in	<i>newNormalizedValue</i>	Normalized value that will be used to set the parameter's new state
----	---------------------------	---

Implements [AAX_IParameter](#).

14.33.4 Member Data Documentation

14.33.4.1 `AAX_CString Abbreviations AAX_CStatelessParameter::mNames [protected]`

Referenced by `AddShortenedName()`, `ClearShortenedNames()`, `Name()`, `SetName()`, and `ShortenedName()`.

14.33.4.2 `AAX_CString AAX_CStatelessParameter::mID [protected]`

Referenced by `Identifier()`.

14.33.4.3 `AAX_IAutomationDelegate* AAX_CStatelessParameter::mAutomationDelegate [protected]`

Referenced by `Release()`, `SetAutomationDelegate()`, and `Touch()`.

14.33.4.4 `AAX_CString AAX_CStatelessParameter::mValueString [protected]`

Referenced by `GetStringFromNormalizedValue()`, `GetValueString()`, `SetValueFromString()`, and `SetValueWithString()`.

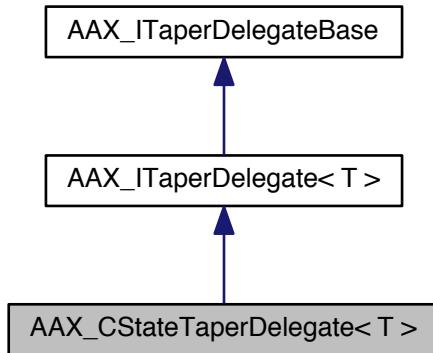
The documentation for this class was generated from the following file:

- [AAX_CParameter.h](#)

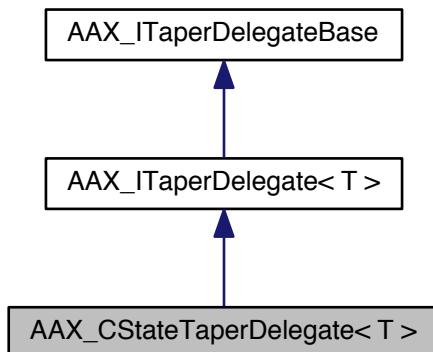
14.34 AAX_CStateTaperDelegate< T > Class Template Reference

```
#include <AAx_CStateTaperDelegate.h>
```

Inheritance diagram for AAX_CStateTaperDelegate< T >:



Collaboration diagram for AAX_CStateTaperDelegate< T >:



14.34.1 Description

```
template<typename T>class AAX_CStateTaperDelegate< T >
```

A linear taper conforming to [AAX_ITaperDelegate](#).

This taper spaces a parameter's real values evenly between its minimum and maximum, with a linear mapping between the parameter's real and normalized values. It is essentially a version of [AAx_CLinearTaperDelegate](#) without that class' additional RealPrecision templatization.

Public Member Functions

- **AAX_CStateTaperDelegate (T minValue=0, T maxValue=1)**
Constructs a State Taper with specified minimum and maximum values.
- **virtual AAX_CStateTaperDelegate< T > * Clone () const AAX_OVERRIDE**
Constructs and returns a copy of the taper delegate.
- **virtual T GetMinimumValue () const AAX_OVERRIDE**
Returns the taper's minimum real value.
- **virtual T GetMaximumValue () const AAX_OVERRIDE**
Returns the taper's maximum real value.
- **virtual T ConstrainRealValue (T value) const AAX_OVERRIDE**
Applies a constraint to the value and returns the constrained value.
- **virtual T NormalizedToReal (double normalizedValue) const AAX_OVERRIDE**
Converts a normalized value to a real value.
- **virtual double RealToNormalized (T realValue) const AAX_OVERRIDE**
Normalizes a real parameter value.

14.34.2 Constructor & Destructor Documentation

14.34.2.1 template<typename T> AAX_CStateTaperDelegate< T >::AAX_CStateTaperDelegate (T minValue = 0, T maxValue = 1)

Constructs a State Taper with specified minimum and maximum values.

Note

The parameter's default value should lie within the min to max range.

Parameters

in	<i>minValue</i>	
in	<i>maxValue</i>	

14.34.3 Member Function Documentation

14.34.3.1 template<typename T> AAX_CStateTaperDelegate< T > * AAX_CStateTaperDelegate< T >::Clone () const [virtual]

Constructs and returns a copy of the taper delegate.

In general, this method's implementation can use a simple copy constructor:

```
template <typename T>
AAX_CSubclassTaperDelegate<T>* AAX_CSubclassTaperDelegate<T>::Clone() const
{
    return new AAX_CSubclassTaperDelegate(*this);
}
```

Implements **AAX_ITaperDelegate< T >**.

14.34.3.2 template<typename T> virtual T AAX_CStateTaperDelegate< T >::GetMinimumValue () const [inline], [virtual]

Returns the taper's minimum real value.

Implements **AAX_ITaperDelegate< T >**.

14.34.3.3 template<typename T> virtual T AAX_CStateTaperDelegate< T >::GetMaximumValue () const
[inline], [virtual]

Returns the taper's maximum real value.

Implements [AAX_ITaperDelegate< T >](#).

14.34.3.4 template<typename T > T AAX_CStateTaperDelegate< T >::ConstrainRealValue (T value) const
[virtual]

Applies a constraint to the value and returns the constrained value.

This method is useful if the taper requires a constraint beyond simple minimum and maximum real value limits.

Note

This is the function that should actually enforces the constraints in `NormalizeToReal()` and `RealToNormalized()`.

Parameters

in	value	The unconstrained value
----	-------	-------------------------

Implements [AAX_ITaperDelegate< T >](#).

14.34.3.5 template<typename T > T AAX_CStateTaperDelegate< T >::NormalizedToReal (double *normalizedValue*)
const [virtual]

Converts a normalized value to a real value.

This is where the actual taper algorithm is implemented.

This function should perform the exact inverse of `RealToNormalized()`, to within the roundoff precision of the individual taper implementation.

Parameters

in	<i>normalizedValue</i>	The normalized value that will be converted
----	------------------------	---

Implements [AAX_ITaperDelegate< T >](#).

14.34.3.6 template<typename T > double AAX_CStateTaperDelegate< T >::RealToNormalized (T *realValue*) const
[virtual]

Normalizes a real parameter value.

This is where the actual taper algorithm is implemented.

This function should perform the exact inverse of `NormalizedToReal()`, to within the roundoff precision of the individual taper implementation.

Parameters

in	<i>realValue</i>	The real parameter value that will be normalized
----	------------------	--

Implements [AAX_ITaperDelegate< T >](#).

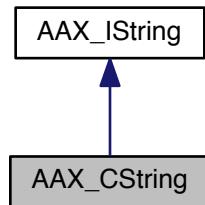
The documentation for this class was generated from the following file:

- [AAX_CStateTaperDelegate.h](#)

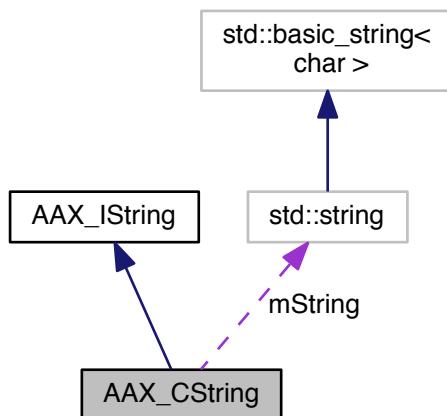
14.35 AAX_CString Class Reference

```
#include <AAX_CString.h>
```

Inheritance diagram for AAX_CString:



Collaboration diagram for AAX_CString:



14.35.1 Description

A generic AAX string class with similar functionality to `std::string`

Public Member Functions

- `virtual uint32_t Length () const`
- `virtual uint32_t MaxLength () const`
- `virtual const char * Get () const`
- `virtual void Set (const char *iString)`
- `virtual AAX_IString & operator= (const AAX_IString &iOther)`

- virtual [AAx_IString & operator=](#) (const char *iString)
- [AAx_CString \(\)](#)
- [AAx_CString \(const char *str\)](#)
- [AAx_CString \(const std::string &str\)](#)
- [AAx_CString \(const AAX_CString &other\)](#)
- [AAx_CString \(const AAX_IString &other\)](#)
- [std::string & StdString \(\)](#)
- const std::string & [StdString \(\) const](#)
- [AAx_CString & operator= \(const AAX_CString &other\)](#)
- [AAx_CString & operator= \(const std::string &other\)](#)
- void [Clear \(\)](#)
- bool [Empty \(\) const](#)
- [AAx_CString & Erase \(uint32_t pos, uint32_t n\)](#)
- [AAx_CString & Append \(const AAX_CString &str\)](#)
- [AAx_CString & Append \(const char *str\)](#)
- [AAx_CString & AppendNumber \(double number, int32_t precision\)](#)
- [AAx_CString & AppendNumber \(int32_t number\)](#)
- [AAx_CString & AppendHex \(int32_t number, int32_t width\)](#)
- [AAx_CString & Insert \(uint32_t pos, const AAX_CString &str\)](#)
- [AAx_CString & Insert \(uint32_t pos, const char *str\)](#)
- [AAx_CString & InsertNumber \(uint32_t pos, double number, int32_t precision\)](#)
- [AAx_CString & InsertNumber \(uint32_t pos, int32_t number\)](#)
- [AAx_CString & InsertHex \(uint32_t pos, int32_t number, int32_t width\)](#)
- [AAx_CString & Replace \(uint32_t pos, uint32_t n, const AAX_CString &str\)](#)
- [AAx_CString & Replace \(uint32_t pos, uint32_t n, const char *str\)](#)
- uint32_t [FindFirst \(const AAX_CString &findStr\) const](#)
- uint32_t [FindFirst \(const char *findStr\) const](#)
- uint32_t [FindFirst \(char findChar\) const](#)
- uint32_t [FindLast \(const AAX_CString &findStr\) const](#)
- uint32_t [FindLast \(const char *findStr\) const](#)
- uint32_t [FindLast \(char findChar\) const](#)
- const char * [CString \(\) const](#)
- bool [ToDouble \(double *oValue\) const](#)
- bool [ToInteger \(int32_t *oValue\) const](#)
- void [SubString \(uint32_t pos, uint32_t n, AAX_IString *outputStr\) const](#)
- bool [Equals \(const AAX_CString &other\) const](#)
- bool [Equals \(const char *other\) const](#)
- bool [Equals \(const std::string &other\) const](#)
- bool [operator== \(const AAX_CString &other\) const](#)
- bool [operator== \(const char *otherStr\) const](#)
- bool [operator== \(const std::string &otherStr\) const](#)
- bool [operator!= \(const AAX_CString &other\) const](#)
- bool [operator!= \(const char *otherStr\) const](#)
- bool [operator!= \(const std::string &otherStr\) const](#)
- bool [operator< \(const AAX_CString &other\) const](#)
- bool [operator> \(const AAX_CString &other\) const](#)
- const char & [operator\[\] \(uint32_t index\) const](#)
- char & [operator\[\] \(uint32_t index\)](#)
- [AAx_CString & operator+= \(const AAX_CString &str\)](#)
- [AAx_CString & operator+= \(const std::string &str\)](#)
- [AAx_CString & operator+= \(const char *str\)](#)

Static Public Attributes

- static const uint32_t `kInvalidIndex` = static_cast<uint32_t>(-1)
- static const uint32_t `kMaxStringLength` = static_cast<uint32_t>(-2)

Protected Attributes

- std::string `mString`

Friends

- std::ostream & `operator<<` (std::ostream &os, const `AAX_CString` &str)
- std::istream & `operator>>` (std::istream &os, `AAX_CString` &str)

14.35.2 Constructor & Destructor Documentation

14.35.2.1 `AAX_CString::AAX_CString()`

Constructs an empty string.

14.35.2.2 `AAX_CString::AAX_CString(const char * str)`

Implicit conversion constructor: Constructs a string with a const char* pointer to copy.

14.35.2.3 `AAX_CString::AAX_CString(const std::string & str) [explicit]`

Copy constructor: Constructs a string from a std::string. Beware of STL variations across various binaries.

14.35.2.4 `AAX_CString::AAX_CString(const AAX_CString & other)`

Copy constructor: Constructs a string with another concrete `AAX_CString`.

14.35.2.5 `AAX_CString::AAX_CString(const AAX_IString & other)`

Copy constructor: Constructs a string from another string that meets the `AAX_IString` interface.

14.35.3 Member Function Documentation

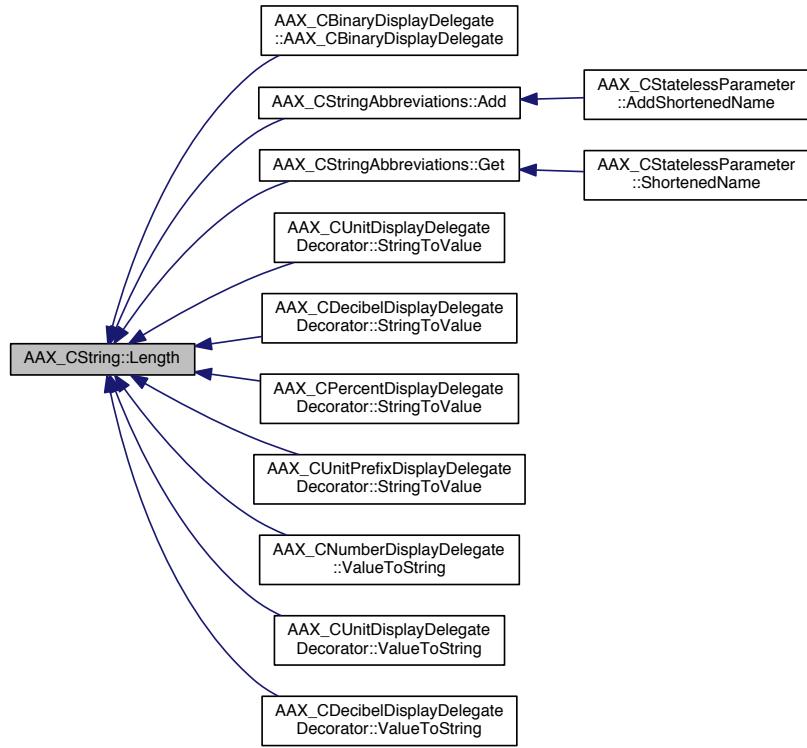
14.35.3.1 `virtual uint32_t AAX_CString::Length() const [virtual]`

Length methods

Implements `AAX_IString`.

Referenced by `AAX_CBinaryDisplayDelegate< T >::AAX_CBinaryDisplayDelegate()`, `AAX_CStringAbbreviations::Add()`, `AAX_CStringAbbreviations::Get()`, `AAX_CUnitDisplayDelegateDecorator< T >::StringToValue()`, `AAX_CDecibelDisplayDelegateDecorator< T >::StringToValue()`, `AAX_CPercentDisplayDelegateDecorator< T >::StringToValue()`, `AAX_CUnitPrefixDisplayDelegateDecorator< T >::StringToValue()`, `AAX_CNumberDisplayDelegate< T, Precision, SpaceAfter >::ValueToString()`, `AAX_CUnitDisplayDelegateDecorator< T >::ValueToString()`, and `AAX_CDecibelDisplayDelegateDecorator< T >::ValueToString()`.

Here is the caller graph for this function:



14.35.3.2 virtual uint32_t AAX_CString::MaxLength() const [virtual]

Implements [AAX_IString](#).

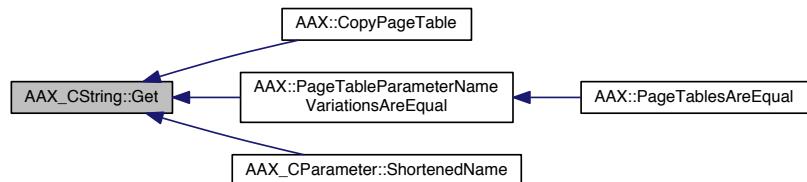
14.35.3.3 virtual const char* AAX_CString::Get() const [virtual]

C string methods

Implements [AAX_IString](#).

Referenced by `AAX::CopyPageTable()`, `AAX::PageTableNameVariationsAreEqual()`, and `AAX_CParameter< T >::ShortenedName()`.

Here is the caller graph for this function:



14.35.3.4 `virtual void AAX_CString::Set(const char * iString) [virtual]`

Implements [AAX_IString](#).

14.35.3.5 `virtual AAX_IString& AAX_CString::operator=(const AAX_IString & iOther) [virtual]`

Assignment operators

Implements [AAX_IString](#).

14.35.3.6 `virtual AAX_IString& AAX_CString::operator=(const char * iString) [virtual]`

Implements [AAX_IString](#).

14.35.3.7 `std::string& AAX_CString::StdString()`

Direct access to a std::string.

14.35.3.8 `const std::string& AAX_CString::StdString() const`

Direct access to a const std::string.

14.35.3.9 `AAX_CString& AAX_CString::operator=(const AAX_CString & other)`

Assignment operator from another [AAX_CString](#). This is kind of required or the compiler will make one for us.

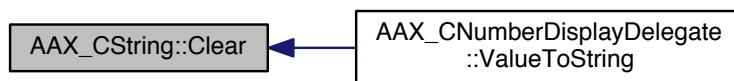
14.35.3.10 `AAX_CString& AAX_CString::operator=(const std::string & other)`

Assignment operator from a std::string. Beware of STL variations across various binaries.

14.35.3.11 `void AAX_CString::Clear()`

Referenced by `AAX_CNumberDisplayDelegate< T, Precision, SpaceAfter >::ValueToString()`.

Here is the caller graph for this function:

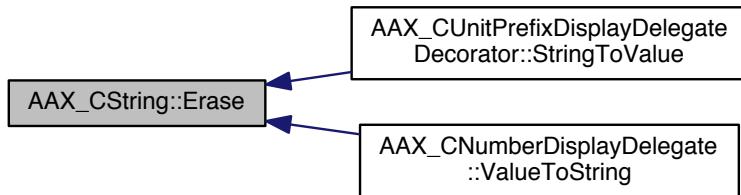


14.35.3.12 `bool AAX_CString::Empty() const`

14.35.3.13 `AAX_CString& AAX_CString::Erase(uint32_t pos, uint32_t n)`

Referenced by `AAX_CUnitPrefixDisplayDelegateDecorator< T >::StringToValue()`, and `AAX_CNumberDisplayDelegate< T, Precision, SpaceAfter >::ValueToString()`.

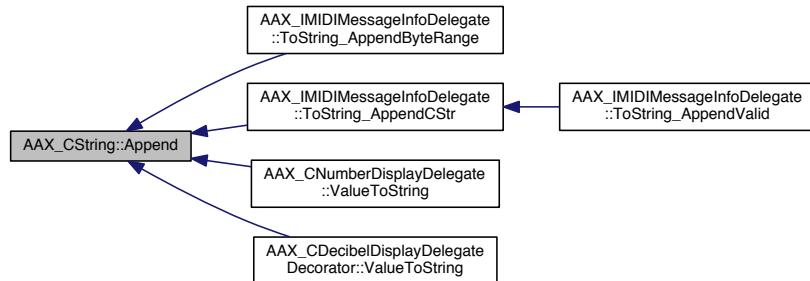
Here is the caller graph for this function:



14.35.3.14 AAX_CString& AAX_CString::Append (const AAX_CString & str)

Referenced by `AAX_IMIDIMessageInfoDelegate::ToString_AppendByteRange()`, `AAX_IMIDIMessageInfoDelegate::ToString_AppendCStr()`, `AAX_CNumberDisplayDelegate< T, Precision, SpaceAfter >::ValueToString()`, and `AAX_CDecibelDisplayDelegateDecorator< T >::ValueToString()`.

Here is the caller graph for this function:

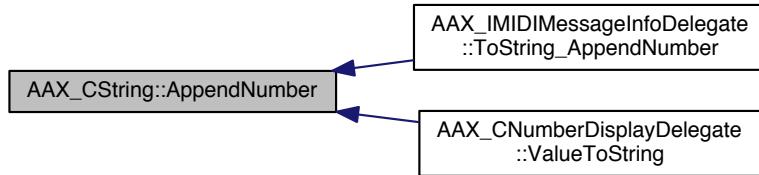


14.35.3.15 AAX_CString& AAX_CString::Append (const char * str)

14.35.3.16 AAX_CString& AAX_CString::AppendNumber (double number, int32_t precision)

Referenced by `AAX_IMIDIMessageInfoDelegate::ToString_AppendNumber()`, and `AAX_CNumberDisplayDelegate< T, Precision, SpaceAfter >::ValueToString()`.

Here is the caller graph for this function:



14.35.3.17 `AAX_CString& AAX_CString::AppendNumber(int32_t number)`

14.35.3.18 `AAX_CString& AAX_CString::AppendHex(int32_t number, int32_t width)`

Referenced by `AAX_IMIDIMessageInfoDelegate::ToString_AppendByteRange()`.

Here is the caller graph for this function:



14.35.3.19 `AAX_CString& AAX_CString::Insert(uint32_t pos, const AAX_CString & str)`

14.35.3.20 `AAX_CString& AAX_CString::Insert(uint32_t pos, const char * str)`

14.35.3.21 `AAX_CString& AAX_CString::InsertNumber(uint32_t pos, double number, int32_t precision)`

14.35.3.22 `AAX_CString& AAX_CString::InsertNumber(uint32_t pos, int32_t number)`

14.35.3.23 `AAX_CString& AAX_CString::InsertHex(uint32_t pos, int32_t number, int32_t width)`

14.35.3.24 `AAX_CString& AAX_CString::Replace(uint32_t pos, uint32_t n, const AAX_CString & str)`

14.35.3.25 `AAX_CString& AAX_CString::Replace(uint32_t pos, uint32_t n, const char * str)`

14.35.3.26 `uint32_t AAX_CString::FindFirst(const AAX_CString & findStr) const`

14.35.3.27 `uint32_t AAX_CString::FindFirst(const char * findStr) const`

14.35.3.28 `uint32_t AAX_CString::FindFirst(char findChar) const`

14.35.3.29 `uint32_t AAX_CString::FindLast(const AAX_CString & findStr) const`

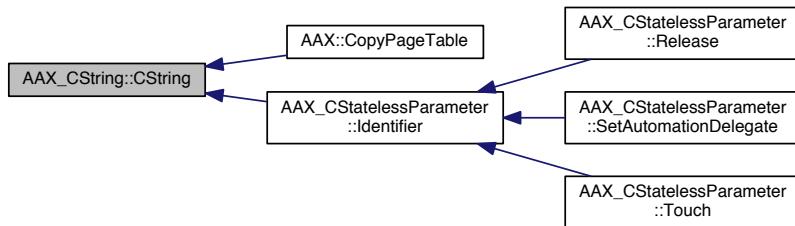
14.35.3.30 `uint32_t AAX_CString::FindLast (const char * findStr) const`

14.35.3.31 `uint32_t AAX_CString::FindLast (char findChar) const`

14.35.3.32 `const char* AAX_CString::CString () const`

Referenced by `AAX::CopyPageTable()`, and `AAX_CStatelessParameter::Identifier()`.

Here is the caller graph for this function:



14.35.3.33 `bool AAX_CString::.ToDouble (double * oValue) const`

Referenced by `AAX_CNumberDisplayDelegate< T, Precision, SpaceAfter >::StringToValue()`.

Here is the caller graph for this function:

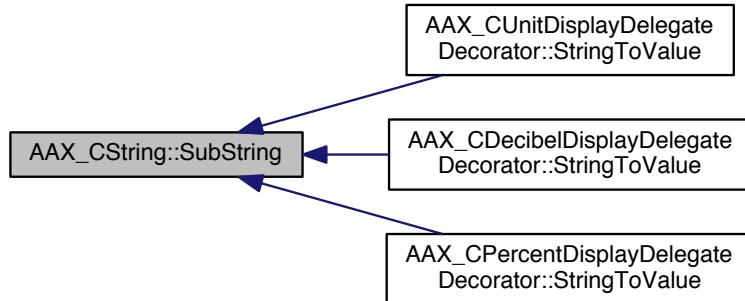


14.35.3.34 `bool AAX_CString::ToInteger (int32_t * oValue) const`

14.35.3.35 `void AAX_CString::SubString (uint32_t pos, uint32_t n, AAX_IString * outputStr) const`

Referenced by `AAX_CUnitDisplayDelegateDecorator< T >::StringToValue()`, `AAX_CDecibelDisplayDelegateDecorator< T >::StringToValue()`, and `AAX_CPercentDisplayDelegateDecorator< T >::StringToValue()`.

Here is the caller graph for this function:



14.35.3.36 bool AAX_CString::Equals (const AAX_CString & other) const [inline]

References operator==().

Here is the call graph for this function:



14.35.3.37 bool AAX_CString::Equals (const char * other) const [inline]

References operator==().

Here is the call graph for this function:



14.35.3.38 bool AAX_CString::Equals (const std::string & other) const [inline]

References operator==().

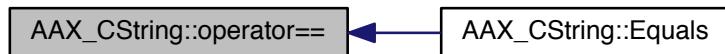
Here is the call graph for this function:



14.35.3.39 `bool AAX_CString::operator==(const AAX_CString & other) const`

Referenced by Equals().

Here is the caller graph for this function:



14.35.3.40 `bool AAX_CString::operator==(const char * otherStr) const`

14.35.3.41 `bool AAX_CString::operator==(const std::string & otherStr) const`

14.35.3.42 `bool AAX_CString::operator!=(const AAX_CString & other) const`

14.35.3.43 `bool AAX_CString::operator!=(const char * otherStr) const`

14.35.3.44 `bool AAX_CString::operator!=(const std::string & otherStr) const`

14.35.3.45 `bool AAX_CString::operator< (const AAX_CString & other) const`

14.35.3.46 `bool AAX_CString::operator> (const AAX_CString & other) const`

14.35.3.47 `const char& AAX_CString::operator[] (uint32_t index) const`

14.35.3.48 `char& AAX_CString::operator[] (uint32_t index)`

14.35.3.49 `AAX_CString& AAX_CString::operator+= (const AAX_CString & str)`

14.35.3.50 `AAX_CString& AAX_CString::operator+= (const std::string & str)`

14.35.3.51 `AAX_CString& AAX_CString::operator+= (const char * str)`

14.35.4 Friends And Related Function Documentation

14.35.4.1 `std::ostream& operator<< (std::ostream & os, const AAX_CString & str) [friend]`

output stream operator for concrete [AAX_CString](#)

14.35.4.2 `std::istream& operator>> (std::istream & os, AAX_CString & str) [friend]`

input stream operator for concrete [AAX_CString](#)

14.35.5 Member Data Documentation

14.35.5.1 `const uint32_t AAX_CString::kInvalidIndex = static_cast<uint32_t>(-1) [static]`

14.35.5.2 `const uint32_t AAX_CString::kMaxStringLength = static_cast<uint32_t>(-2) [static]`

14.35.5.3 `std::string AAX_CString::mString [protected]`

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

- [AAX_CString.h](#)

14.36 AAX_CStringAbbreviations Class Reference

```
#include <AAX_CString.h>
```

14.36.1 Description

Helper class to store a collection of name abbreviations.

Public Member Functions

- [AAX_CStringAbbreviations \(const AAX_CString &inPrimary\)](#)
- void [SetPrimary \(const AAX_CString &inPrimary\)](#)
- const [AAX_CString & Primary \(\) const](#)
- void [Add \(const AAX_CString &inAbbreviation\)](#)
- const [AAX_CString & Get \(int32_t inNumCharacters\) const](#)
- void [Clear \(\)](#)

14.36.2 Constructor & Destructor Documentation

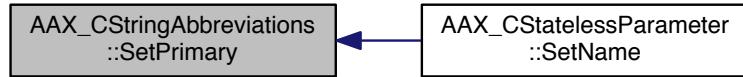
14.36.2.1 `AAX_CStringAbbreviations::AAX_CStringAbbreviations (const AAX_CString & inPrimary) [inline], [explicit]`

14.36.3 Member Function Documentation

14.36.3.1 `void AAX_CStringAbbreviations::SetPrimary (const AAX_CString & inPrimary) [inline]`

Referenced by [AAX_CStatelessParameter::SetName\(\)](#).

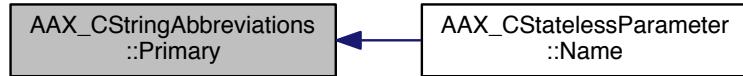
Here is the caller graph for this function:



14.36.3.2 const AAX_CString& AAX_CStringAbbreviations::Primary() const [inline]

Referenced by `AAX_CStatelessParameter::Name()`.

Here is the caller graph for this function:

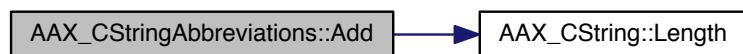


14.36.3.3 void AAX_CStringAbbreviations::Add(const AAX_CString & inAbbreviation) [inline]

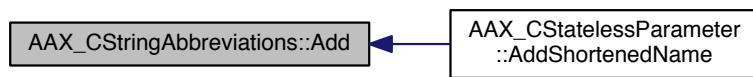
References `AAX_CString::Length()`.

Referenced by `AAX_CStatelessParameter::AddShortenedName()`.

Here is the call graph for this function:



Here is the caller graph for this function:



14.36.3.4 const AAX_CString& AAX_CStringAbbreviations::Get(int32_t inNumCharacters) const [inline]

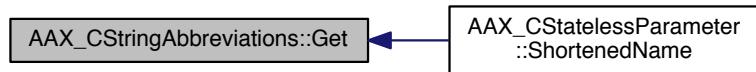
References AAX_CString::Length().

Referenced by AAX_CStatelessParameter::ShortenedName().

Here is the call graph for this function:



Here is the caller graph for this function:



14.36.3.5 void AAX_CStringAbbreviations::Clear() [inline]

Referenced by AAX_CStatelessParameter::ClearShortenedNames().

Here is the caller graph for this function:



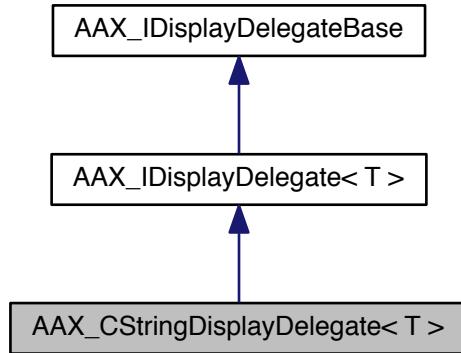
The documentation for this class was generated from the following file:

- [AAX_CString.h](#)

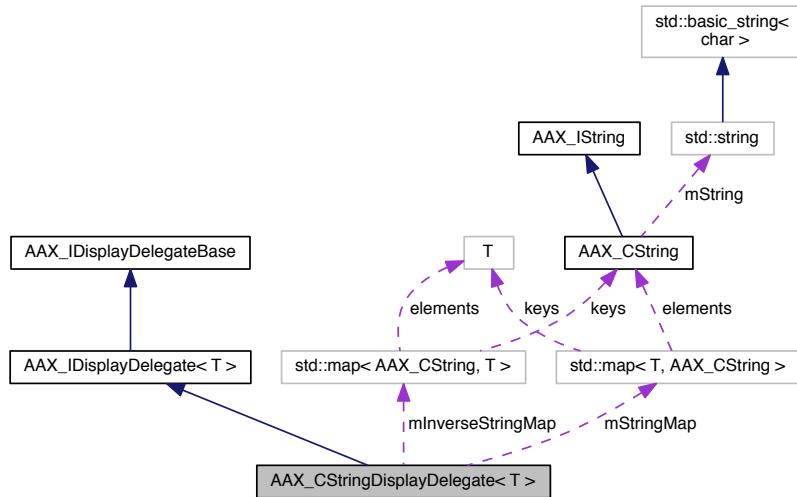
14.37 AAX_CStringDisplayDelegate< T > Class Template Reference

#include <AAX_CStringDisplayDelegate.h>

Inheritance diagram for AAX_CStringDisplayDelegate< T >:



Collaboration diagram for AAX_CStringDisplayDelegate< T >:



14.37.1 Description

```
template<typename T> class AAX_CStringDisplayDelegate< T >
```

A string, or list, display format conforming to [AAX_IDisplayDelegate](#).

This display delegate uses a string map to associate parameter values with specific strings. This kind of display delegate is most often used for control string or list parameters, which would internally use an integer parameter type. The int value would then be used as a lookup into this delegate, which would return a string for each valid int value.

Public Member Functions

- `AAX_CStringDisplayDelegate` (const std::map< T, AAX_CString > &stringMap)
Constructor.
- virtual `AAX_CStringDisplayDelegate< T > * Clone () const AAX_OVERRIDE`
Constructs and returns a copy of the display delegate.
- virtual bool `ValueToString` (T value, AAX_CString *valueString) const `AAX_OVERRIDE`
Converts a real parameter value to a string representation.
- virtual bool `ValueToString` (T value, int32_t maxNumChars, AAX_CString *valueString) const `AAX_OVERRIDE`
Converts a real parameter value to a string representation using a size hint, useful for control surfaces and other character limited displays.
- virtual bool `StringToValue` (const AAX_CString &valueString, T *value) const `AAX_OVERRIDE`
Converts a string to a real parameter value.

Protected Attributes

- std::map< T, AAX_CString > `mStringMap`
- std::map< AAX_CString, T > `mInverseStringMap`

14.37.2 Constructor & Destructor Documentation

14.37.2.1 template<typename T> AAX_CStringDisplayDelegate< T >::AAX_CStringDisplayDelegate (const std::map< T, AAX_CString > & stringMap)

Constructor.

Constructs a String Display Delegate with a provided string map.

Note

The string map should already be populated with value-string pairs, as this constructor will copy the provided map into the delegate object's own memory.

Parameters

in	stringMap	A populated map of value-string pairs
----	-----------	---------------------------------------

References AAX_CStringDisplayDelegate< T >::mInverseStringMap, and AAX_CStringDisplayDelegate< T >::mStringMap.

14.37.3 Member Function Documentation

14.37.3.1 template<typename T> AAX_CStringDisplayDelegate< T > * AAX_CStringDisplayDelegate< T >::Clone () const [virtual]

Constructs and returns a copy of the display delegate.

In general, this method's implementation can use a simple copy constructor:

```
template <typename T>
AAX_CSubclassDisplayDelegate<T>* AAX_CSubclassDisplayDelegate<T>::Clone () const
{
    return new AAX_CSubclassDisplayDelegate(*this);
}
```

Implements `AAX_IDisplayDelegate< T >`.

14.37.3.2 template<typename T> bool AAX_CStringDisplayDelegate< T >::ValueToString (T value, AAX_CString *
 valueString) const [virtual]

Converts a real parameter value to a string representation.

Parameters

in	<i>value</i>	The real parameter value that will be converted
out	<i>valueString</i>	A string corresponding to value

Return values

<i>true</i>	The string conversion was successful
<i>false</i>	The string conversion was unsuccessful

Implements [AAX_IDisplayDelegate< T >](#).

14.37.3.3 template<typename T> bool AAX_CStringDisplayDelegate< T >::ValueToString (T *value*, int32_t *maxNumChars*, AAX_CString * *valueString*) const [virtual]

Converts a real parameter value to a string representation using a size hint, useful for control surfaces and other character limited displays.

Parameters

in	<i>value</i>	The real parameter value that will be converted
in	<i>maxNumChars</i>	Size hint for the desired maximum number of characters in the string (not including null termination)
out	<i>valueString</i>	A string corresponding to value

Return values

<i>true</i>	The string conversion was successful
<i>false</i>	The string conversion was unsuccessful

Implements [AAX_IDisplayDelegate< T >](#).

14.37.3.4 template<typename T> bool AAX_CStringDisplayDelegate< T >::StringToValue (const AAX_CString & *valueString*, T * *value*) const [virtual]

Converts a string to a real parameter value.

Parameters

in	<i>valueString</i>	The string that will be converted
out	<i>value</i>	The real parameter value corresponding to valueString

Return values

<i>true</i>	The string conversion was successful
<i>false</i>	The string conversion was unsuccessful

Implements [AAX_IDisplayDelegate< T >](#).

14.37.4 Member Data Documentation

14.37.4.1 template<typename T> std::map<T, AAX_CString> AAX_CStringDisplayDelegate< T >::mStringMap [protected]

Referenced by [AAX_CStringDisplayDelegate< T >::AAX_CStringDisplayDelegate\(\)](#).

14.37.4.2 template<typename T> std::map<AAAX_CString, T> AAX_CStringDisplayDelegate< T >::mInverseStringMap [protected]

Referenced by AAX_CStringDisplayDelegate< T >::AAX_CStringDisplayDelegate().

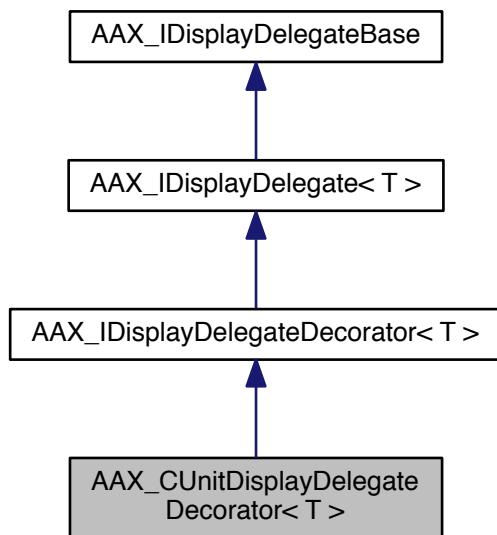
The documentation for this class was generated from the following file:

- [AAAX_CStringDisplayDelegate.h](#)

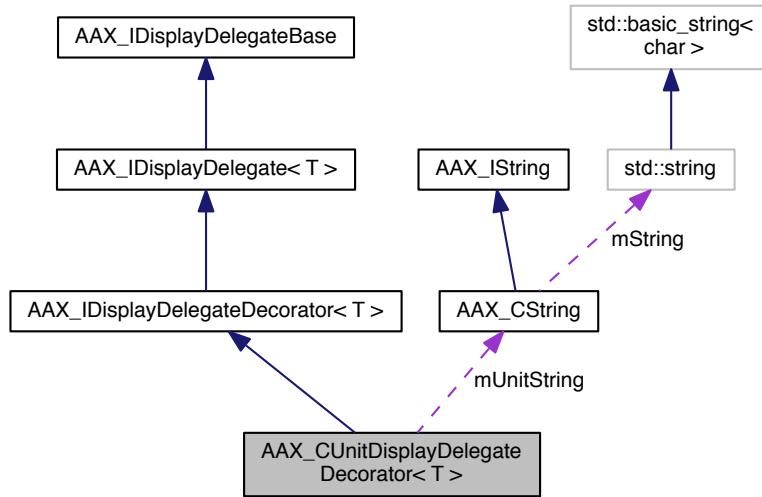
14.38 AAX_CUnitDisplayDelegateDecorator< T > Class Template Reference

#include <AAAX_CUnitDisplayDelegateDecorator.h>

Inheritance diagram for AAX_CUnitDisplayDelegateDecorator< T >:



Collaboration diagram for `AAX_CUnitDisplayDelegateDecorator< T >`:



14.38.1 Description

```
template<typename T>class AAX_CUnitDisplayDelegateDecorator< T >
```

A unit type decorator conforming to [AAX_IDisplayDelegateDecorator](#).

This class is an [AAX_IDisplayDelegateDecorator](#), meaning that it acts as a wrapper for other display delegates or concrete display types. For more information about display delegate decorators in [AAX](#), see [Display delegate decorators](#)

The behavior of this class is to decorate parameter value strings with arbitrary units, such as "Hz" or "V". The inverse is also supported, so the unit string is pulled off of value strings when they are converted to real parameter values.

Public Member Functions

- `AAX_CUnitDisplayDelegateDecorator (const AAX_IDisplayDelegate< T > &displayDelegate, const AAX_CString &unitString)`
Constructor.
- virtual `AAX_CUnitDisplayDelegateDecorator< T > * Clone () const AAX_OVERRIDE`
Constructs and returns a copy of the display delegate decorator.
- virtual `bool ValueToString (T value, AAX_CString *valueString) const AAX_OVERRIDE`
Converts a string to a real parameter value.
- virtual `bool ValueToString (T value, int32_t maxNumChars, AAX_CString *valueString) const AAX_OVERRIDE`
Converts a string to a real parameter value with a size constraint.
- virtual `bool StringToValue (const AAX_CString &valueString, T *value) const AAX_OVERRIDE`
Converts a string to a real parameter value.

Protected Attributes

- const `AAX_CString mUnitString`

14.38.2 Constructor & Destructor Documentation

14.38.2.1 template<typename T> AAX_CUnitDisplayDelegateDecorator<T>::AAX_CUnitDisplayDelegateDecorator(const AAX_IDisplayDelegate<T> & *displayDelegate*, const AAX_CString & *unitString*)

Constructor.

Along with the standard decorator pattern argument, this class also takes a unit string. This is the string that will be added to the end of valueString.

Parameters

in	<i>displayDelegate</i>
in	<i>unitString</i>

14.38.3 Member Function Documentation

14.38.3.1 template<typename T> AAX_CUnitDisplayDelegateDecorator<T> * AAX_CUnitDisplayDelegateDecorator<T>::Clone() const [virtual]

Constructs and returns a copy of the display delegate decorator.

In general, this method's implementation can use a simple copy constructor:

```
template <typename T>
AAX_CSubclassDisplayDelegate<T>* AAX_CSubclassDisplayDelegate<T>::Clone() const
{
    return new AAX_CSubclassDisplayDelegate(*this);
}
```

Note

This is an idiomatic method in the decorator pattern, so watch for potential problems if this method is ever changed or removed.

Reimplemented from [AAX_IDisplayDelegateDecorator< T >](#).

14.38.3.2 template<typename T> bool AAX_CUnitDisplayDelegateDecorator<T>::ValueToString(T *value*, AAX_CString * *valueString*) const [virtual]

Converts a string to a real parameter value.

Override of the [AAX_IDisplayDelegate](#) implementation to call into the wrapped object. Display delegate decorators should call into this implementation to pass [ValueToString\(\)](#) calls on to the wrapped object after applying their own value-to-string decoration.

Parameters

in	<i>valueString</i>	The string that will be converted
out	<i>value</i>	The real parameter value corresponding to valueString

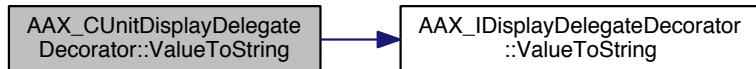
Return values

<i>true</i>	The string conversion was successful
<i>false</i>	The string conversion was unsuccessful

Reimplemented from [AAX_IDisplayDelegateDecorator< T >](#).

References [AAX_IDisplayDelegateDecorator< T >::ValueToString\(\)](#).

Here is the call graph for this function:



14.38.3.3 template<typename T> bool AAX_CUnitDisplayDelegateDecorator< T >::ValueToString (T value, int32_t maxNumChars, AAX_CString * valueString) const [virtual]

Converts a string to a real parameter value with a size constraint.

Override of the [AAX_IDisplayDelegate](#) implementation to call into the wrapped object. Display delegate decorators should call into this implementation to pass [ValueToString\(\)](#) calls on to the wrapped object after applying their own value-to-string decoration.

Parameters

in	<i>valueString</i>	The string that will be converted
in	<i>maxNumChars</i>	Size hint for the desired maximum number of characters in the string (not including null termination)
out	<i>value</i>	The real parameter value corresponding to <i>valueString</i>

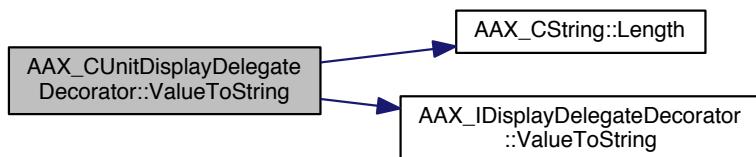
Return values

<i>true</i>	The string conversion was successful
<i>false</i>	The string conversion was unsuccessful

Reimplemented from [AAX_IDisplayDelegateDecorator< T >](#).

References [AAX_CString::Length\(\)](#), and [AAX_IDisplayDelegateDecorator< T >::ValueToString\(\)](#).

Here is the call graph for this function:



14.38.3.4 template<typename T> bool AAX_CUnitDisplayDelegateDecorator< T >::StringToValue (const AAX_CString & valueString, T * value) const [virtual]

Converts a string to a real parameter value.

Override of the DisplayDecorator implementation to call into the wrapped object. Display delegate decorators should call into this implementation to pass [StringToValue\(\)](#) calls on to the wrapped object after applying their own string-to-value decoding.

Parameters

in	valueString	The string that will be converted
out	value	The real parameter value corresponding to valueString

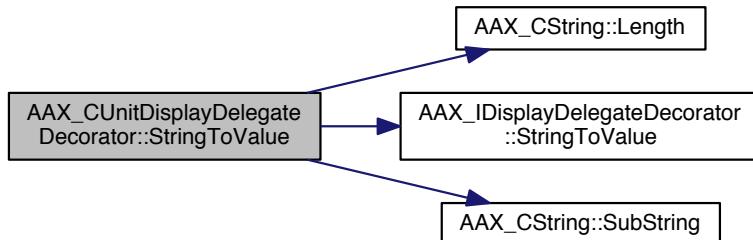
Return values

true	The string conversion was successful
false	The string conversion was unsuccessful

Reimplemented from [AAX_IDisplayDelegateDecorator< T >](#).

References [AAX_CString::Length\(\)](#), [AAX_IDisplayDelegateDecorator< T >::StringToValue\(\)](#), and [AAX_CString::SubString\(\)](#).

Here is the call graph for this function:



14.38.4 Member Data Documentation

14.38.4.1 template<typename T> const AAX_CString AAX_CUnitDisplayDelegateDecorator< T >::mUnitString [protected]

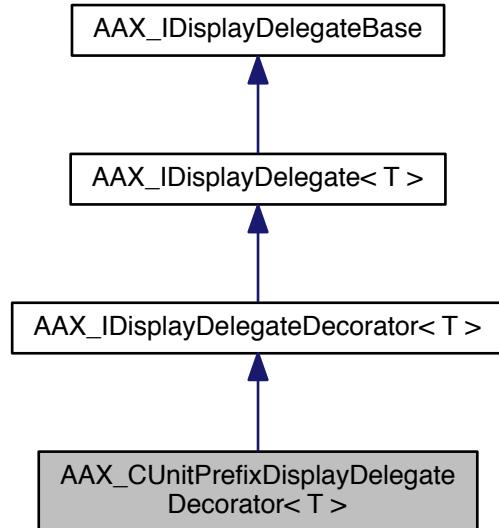
The documentation for this class was generated from the following file:

- [AAX_CUnitDisplayDelegateDecorator.h](#)

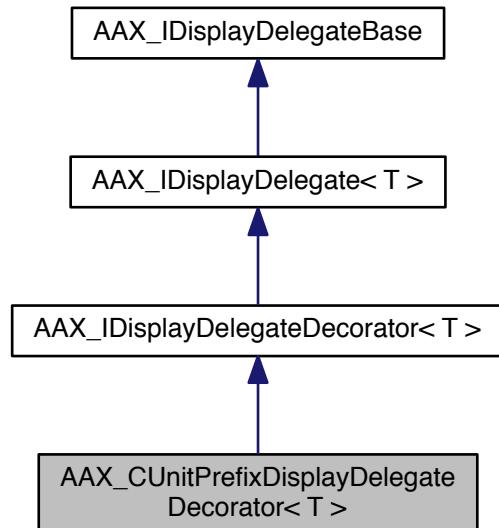
14.39 AAX_CUnitPrefixDisplayDelegateDecorator< T > Class Template Reference

```
#include <AAX_CUnitPrefixDisplayDelegateDecorator.h>
```

Inheritance diagram for AAX_CUnitPrefixDisplayDelegateDecorator< T >:



Collaboration diagram for AAX_CUnitPrefixDisplayDelegateDecorator< T >:



14.39.1 Description

```
template<typename T> class AAX_CUnitPrefixDisplayDelegateDecorator< T >
```

A unit prefix decorator conforming to [AAX_IDisplayDelegateDecorator](#).

This class is an [AAX_IDisplayDelegateDecorator](#), meaning that it acts as a wrapper for other display delegates or concrete display types. For more information about display delegate decorators in [AAX](#), see [Display delegate decorators](#)

The behavior of this class is to provide unit prefixes such as the k in kHz or the m in mm. It takes the value passed in and determines if the value is large or small enough to benefit from a unit modifier. If so, it adds that unit prefix character to the display string after scaling the number and calling deeper into the decorator pattern to get the concrete [ValueToString\(\)](#) result.

The inverse is also supported, so if you type 1.5k in a text box and this decorator is in place, it should find the k and multiply the value by 1000 before converting it to a real value.

This decorator supports the following unit prefixes:

- M (mega-)
- k (kilo-)
- m (milli-)
- u (micro-)

Note

This class is not implemented for integer values as the conversions result in fractional numbers. Those would get truncated through the system and be pretty much useless.

Public Member Functions

- [AAX_CUnitPrefixDisplayDelegateDecorator](#) (const [AAX_IDisplayDelegate< T >](#) &displayDelegate)
- virtual [AAX_CUnitPrefixDisplayDelegateDecorator< T >](#) * [Clone](#) () const [AAX_OVERRIDE](#)
Constructs and returns a copy of the display delegate decorator.
- virtual bool [ValueToString](#) (T value, [AAX_CString](#) *valueString) const [AAX_OVERRIDE](#)
Converts a string to a real parameter value.
- virtual bool [ValueToString](#) (T value, int32_t maxNumChars, [AAX_CString](#) *valueString) const [AAX_OVERRIDE](#)
Converts a string to a real parameter value with a size constraint.
- virtual bool [StringToValue](#) (const [AAX_CString](#) &valueString, T *value) const [AAX_OVERRIDE](#)
Converts a string to a real parameter value.

14.39.2 Constructor & Destructor Documentation

14.39.2.1 template<typename T > [AAX_CUnitPrefixDisplayDelegateDecorator< T >](#)::[AAX_CUnitPrefixDisplayDelegateDecorator](#) (const [AAX_IDisplayDelegate< T >](#) & displayDelegate)

14.39.3 Member Function Documentation

14.39.3.1 template<typename T > [AAX_CUnitPrefixDisplayDelegateDecorator< T >](#) * [AAX_CUnitPrefixDisplayDelegateDecorator< T >](#)::[Clone](#) () const [virtual]

Constructs and returns a copy of the display delegate decorator.

In general, this method's implementation can use a simple copy constructor:

```
template <typename T>
AAX_CSubclassDisplayDelegate<T>* AAX_CSubclassDisplayDelegate<T>::Clone() const
{
    return new AAX_CSubclassDisplayDelegate(*this);
}
```

Note

This is an idiomatic method in the decorator pattern, so watch for potential problems if this method is ever changed or removed.

Reimplemented from [AAX_IDisplayDelegateDecorator< T >](#).

14.39.3.2 template<typename T > bool AAX_CUnitPrefixDisplayDelegateDecorator< T >::ValueToString (T value, AAX_CString * valueString) const [virtual]

Converts a string to a real parameter value.

Override of the [AAX_IDisplayDelegate](#) implementation to call into the wrapped object. Display delegate decorators should call into this implementation to pass [ValueToString\(\)](#) calls on to the wrapped object after applying their own value-to-string decoration.

Parameters

in	valueString	The string that will be converted
out	value	The real parameter value corresponding to valueString

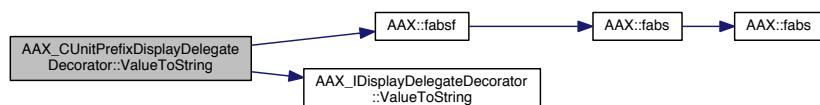
Return values

true	The string conversion was successful
false	The string conversion was unsuccessful

Reimplemented from [AAX_IDisplayDelegateDecorator< T >](#).

References [AAX::fabsf\(\)](#), and [AAX_IDisplayDelegateDecorator< T >::ValueToString\(\)](#).

Here is the call graph for this function:



14.39.3.3 template<typename T > bool AAX_CUnitPrefixDisplayDelegateDecorator< T >::ValueToString (T value, int32_t maxNumChars, AAX_CString * valueString) const [virtual]

Converts a string to a real parameter value with a size constraint.

Override of the [AAX_IDisplayDelegate](#) implementation to call into the wrapped object. Display delegate decorators should call into this implementation to pass [ValueToString\(\)](#) calls on to the wrapped object after applying their own value-to-string decoration.

Parameters

in	<i>valueString</i>	The string that will be converted
in	<i>maxNumChars</i>	Size hint for the desired maximum number of characters in the string (not including null termination)
out	<i>value</i>	The real parameter value corresponding to <i>valueString</i>

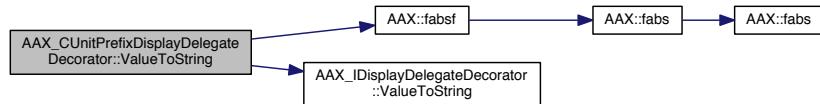
Return values

<i>true</i>	The string conversion was successful
<i>false</i>	The string conversion was unsuccessful

Reimplemented from [AAX_IDisplayDelegateDecorator< T >](#).

References `AAX::fabsf()`, and [AAX_IDisplayDelegateDecorator< T >::ValueToString\(\)](#).

Here is the call graph for this function:



14.39.3.4 template<typename T> bool AAX_CUnitPrefixDisplayDelegateDecorator< T >::StringToValue (const AAX_CString & valueString, T * value) const [virtual]

Converts a string to a real parameter value.

Override of the DisplayDecorator implementation to call into the wrapped object. Display delegate decorators should call into this implementation to pass [StringToValue\(\)](#) calls on to the wrapped object after applying their own string-to-value decoding.

Parameters

in	<i>valueString</i>	The string that will be converted
out	<i>value</i>	The real parameter value corresponding to <i>valueString</i>

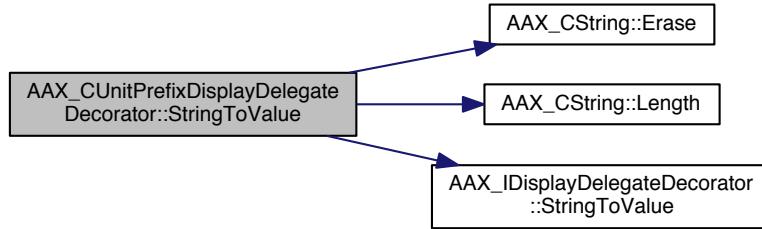
Return values

<i>true</i>	The string conversion was successful
<i>false</i>	The string conversion was unsuccessful

Reimplemented from [AAX_IDisplayDelegateDecorator< T >](#).

References `AAX_CString::Erase()`, `AAX_CString::Length()`, and [AAX_IDisplayDelegateDecorator< T >::StringToValue\(\)](#).

Here is the call graph for this function:



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

- [AAX_CUnitPrefixDisplayDelegateDecorator.h](#)

14.40 AAX_FastInterpolatedTableLookup< TFLOAT, DFLOAT > Class Template Reference

```
#include <AAX_FastInterpolatedTableLookup.h>
```

Public Member Functions

- void [SetParameters](#) (int iTableSize, TFLOAT iMin=0.0, TFLOAT iMax=1.0, int iNumTables=1)
Set the table lookup parameters.
- DFLOAT [DoTableLookupExtraFast](#) (const TFLOAT *const iTable, DFLOAT iValue) const
Perform an extra fast table lookup :)
- void [DoTableLookupExtraFastMulti](#) (const TFLOAT *iTable, DFLOAT iValue, DFLOAT *oValues) const
- void [DoTableLookupExtraFast](#) (const TFLOAT *const iTable, const TFLOAT *const inpBuf, DFLOAT *const outBuf, int blockSize)
- TFLOAT [GetMin](#) ()
- TFLOAT [GetMaxMinusMin](#) ()

14.40.1 Member Function Documentation

14.40.1.1 template<class TFLOAT , class DFLOAT > void AAX_FastInterpolatedTableLookup< TFLOAT, DFLOAT >::SetParameters (int iTableSize, TFLOAT iMin = 0 . 0 , TFLOAT iMax = 1 . 0 , int iNumTables = 1) [inline]

Set the table lookup parameters.

Parameters

in	<i>iTableSize</i>	Size of the lookup table
in	<i>iMin</i>	Minimum input value
in	<i>iMax</i>	Maximum input value

in	<i>iNumTables</i>	Number of tables to index
----	-------------------	---------------------------

Note

For future use...

14.40.1.2 template<class TFLOAT , class DFLOAT > DFLOAT AAX_FastInterpolatedTableLookup< TFLOAT, DFLOAT >::DoTableLookupExtraFast (const TFLOAT *const *iTable*, DFLOAT *iValue*) const [inline]

Perform an extra fast table lookup :)

Parameters

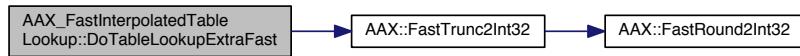
in	<i>iTable</i>	Lookup table
in	<i>iValue</i>	Table value

Note

This version requires that the lookup table is padded with one extra location so we can avoid one of the checks to see if our pointers are out of bounds.

References AAX::FastTrunc2Int32().

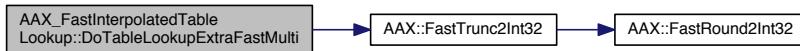
Here is the call graph for this function:



14.40.1.3 template<class TFLOAT , class DFLOAT > void AAX_FastInterpolatedTableLookup< TFLOAT, DFLOAT >::DoTableLookupExtraFastMulti (const TFLOAT * *iTable*, DFLOAT *iValue*, DFLOAT * *oValues*) const [inline]

References AAX::FastTrunc2Int32().

Here is the call graph for this function:



14.40.1.4 template<class TFLOAT , class DFLOAT > void AAX_FastInterpolatedTableLookup< TFLOAT, DFLOAT >::DoTableLookupExtraFast (const TFLOAT *const *iTable*, const TFLOAT *const *inpBuf*, DFLOAT *const *outBuf*, int *blockSize*) [inline]

14.40.1.5 template<class TFLOAT , class DFLOAT > TFLOAT AAX_FastInterpolatedTableLookup< TFLOAT, DFLOAT >::GetMin () [inline]

```
14.40.1.6 template<class TFLOAT , class DFLOAT > TFLOAT AAX_FastInterpolatedTableLookup< TFLOAT, DFLOAT
>::GetMaxMinusMin( ) [inline]
```

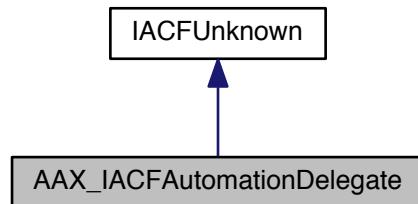
The documentation for this class was generated from the following files:

- [AAX_FastInterpolatedTableLookup.h](#)
- [AAX_FastInterpolatedTableLookup.hpp](#)

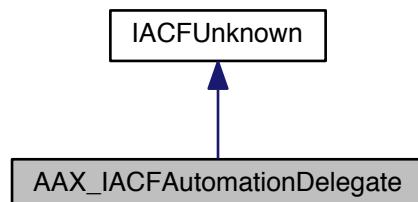
14.41 AAX_IACFAutomationDelegate Class Reference

```
#include <AAX_IACFAutomationDelegate.h>
```

Inheritance diagram for AAX_IACFAutomationDelegate:



Collaboration diagram for AAX_IACFAutomationDelegate:



14.41.1 Description

Versioned interface allowing an AAX plug-in to interact with the host's automation system.

See also

- [Parameter updates](#)
- [AAX_IAutomationDelegate](#)

Public Member Functions

- virtual [AAX_Result RegisterParameter \(AAX_CParamID iParameterID\)=0](#)
- virtual [AAX_Result UnregisterParameter \(AAX_CParamID iParameterID\)=0](#)
- virtual [AAX_Result PostSetValueRequest \(AAX_CParamID iParameterID, double normalizedValue\) const =0](#)
- virtual [AAX_Result PostCurrentValue \(AAX_CParamID iParameterID, double normalizedValue\) const =0](#)
- virtual [AAX_Result PostTouchRequest \(AAX_CParamID iParameterID\)=0](#)
- virtual [AAX_Result PostReleaseRequest \(AAX_CParamID iParameterID\)=0](#)
- virtual [AAX_Result GetTouchState \(AAX_CParamID iParameterID, AAX_CBoolean *oTouched\)=0](#)

14.41.2 Member Function Documentation

14.41.2.1 virtual AAX_Result AAX_IACFAutomationDelegate::RegisterParameter (AAX_CParamID *iParameterID*) [pure virtual]

Register a control with the automation system using a char* based control identifier

The automation delegate owns a list of the IDs of all of the parameters that have been registered with it. This list is used to set up listeners for all of the registered parameters such that the automation delegate may update the plug-in when the state of any of the registered parameters have been modified.

See also

[AAX_IAutomationDelegate::UnregisterParameter\(\)](#)

Parameters

in	<i>iParameterID</i>	Parameter ID that is being registered
----	---------------------	---------------------------------------

14.41.2.2 virtual AAX_Result AAX_IACFAutomationDelegate::UnregisterParameter (AAX_CParamID *iParameterID*) [pure virtual]

Unregister a control with the automation system using a char* based control identifier

Note

All registered controls should be unregistered or the system might leak.

See also

[AAX_IAutomationDelegate::RegisterParameter\(\)](#)

Parameters

in	<i>iParameterID</i>	Parameter ID that is being registered
----	---------------------	---------------------------------------

14.41.2.3 virtual AAX_Result AAX_IACFAutomationDelegate::PostSetValueRequest (AAX_CParamID *iParameterID*, double *normalizedValue*) const [pure virtual]

Submits a request for the given parameter's value to be changed

Parameters

in	<i>iParameterID</i>	ID of the parameter for which a change is requested
in	<i>normalizedValue</i>	The requested new parameter value, formatted as a double and normalized to [0 1]

14.41.2.4 virtual AAX_Result AAX_IACFAutomationDelegate::PostCurrentValue (AAX_CParamID *iParameterID*, double *normalizedValue*) const [pure virtual]

Notifies listeners that a parameter's value has changed

Parameters

in	<i>iParameterID</i>	ID of the parameter that has been updated
in	<i>normalizedValue</i>	The current parameter value, formatted as a double and normalized to [0 1]

14.41.2.5 virtual AAX_Result AAX_IACFAutomationDelegate::PostTouchRequest (AAX_CParamID *iParameterID*) [pure virtual]

Requests that the given parameter be "touched", i.e. locked for updates by the current client

Parameters

in	<i>iParameterID</i>	ID of the parameter that will be touched
----	---------------------	--

14.41.2.6 virtual AAX_Result AAX_IACFAutomationDelegate::PostReleaseRequest (AAX_CParamID *iParameterID*) [pure virtual]

Requests that the given parameter be "released", i.e. available for updates from any client

Parameters

in	<i>iParameterID</i>	ID of the parameter that will be released
----	---------------------	---

14.41.2.7 virtual AAX_Result AAX_IACFAutomationDelegate::GetTouchState (AAX_CParamID *iParameterID*, AAX_CBoolean * *oTouched*) [pure virtual]

Gets the current touched state of a parameter

Parameters

in	<i>iParameterID</i>	ID of the parameter that is being queried
out	<i>oTouched</i>	The current touch state of the parameter

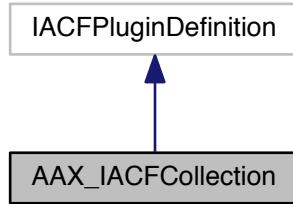
The documentation for this class was generated from the following file:

- [AAX_IACFAutomationDelegate.h](#)

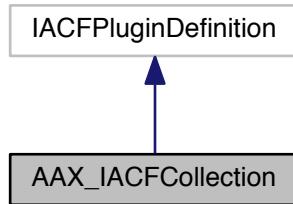
14.42 AAX_IACFCollection Class Reference

```
#include <AAX_IACFCollection.h>
```

Inheritance diagram for AAX_IACFCollection:



Collaboration diagram for AAX_IACFCollection:



14.42.1 Description

Versioned interface to represent a plug-in binary's static description.

Public Member Functions

- virtual [AAAX_Result AddEffect](#) (const char *inEffectID, [IACFUnknown](#) *inEffectDescriptor)=0
Add an Effect description to the collection.
- virtual [AAAX_Result SetManufacturerName](#) (const char *inPackageName)=0
Set the plug-in manufacturer name.
- virtual [AAAX_Result AddPackageName](#) (const char *inPackageName)=0
Set the plug-in package name.
- virtual [AAAX_Result SetPackageVersion](#) (uint32_t inVersion)=0
Set the plug-in package version number.
- virtual [AAAX_Result SetProperties](#) ([IACFUnknown](#) *inProperties)=0
Set the properties of the collection.

14.42.2 Member Function Documentation

14.42.2.1 virtual AAX_Result AAX_IACFCollection::AddEffect (const char * *inEffectID*, IACFUnknown * *inEffectDescriptor*) [pure virtual]

Add an Effect description to the collection.

Each Effect that a plug-in registers with [AAX_ICollection::AddEffect\(\)](#) is considered a completely different user-facing product. For example, in Avid's Dynamics III plug-in the Expander, Compressor, and DeEsser are each registered as separate Effects. All stem format variations within each Effect are registered within that Effect's [AA←X_IEffectDescriptor](#) using [AddComponent\(\)](#).

The [AAX_eProperty_ProductID](#) value for all ProcessProcs within a single Effect must be identical.

This method passes ownership of an [AAX_IEffectDescriptor](#) object to the [AAX_ICollection](#). The [AAX_IEffectDescriptor](#) must not be deleted by the AAX plug-in, nor should it be edited in any way after it is passed to the [AAX_ICollection](#).

Parameters

in	<i>inEffectID</i>	The effect ID.
in	<i>inEffectDescriptor</i>	The Effect descriptor.

14.42.2.2 virtual AAX_Result AAX_IACFCollection::SetManufacturerName (const char * *inPackageName*) [pure virtual]

Set the plug-in manufacturer name.

Parameters

in	<i>inPackageName</i>	The name of the manufacturer.
----	----------------------	-------------------------------

14.42.2.3 virtual AAX_Result AAX_IACFCollection::AddPackageName (const char * *inPackageName*) [pure virtual]

Set the plug-in package name.

May be called multiple times to add abbreviated package names.

Note

Every plug-in must include at least one name variant with 16 or fewer characters, plus a null terminating character. Used for Presets folder.

Parameters

in	<i>inPackageName</i>	The name of the package.
----	----------------------	--------------------------

14.42.2.4 virtual AAX_Result AAX_IACFCollection::SetPackageVersion (uint32_t *inVersion*) [pure virtual]

Set the plug-in package version number.

Parameters

in	<i>inVersion</i>	The package version numner.
----	------------------	-----------------------------

14.42.2.5 virtual AAX_Result AAX_IACFCollection::SetProperties (IACFUnknown * *inProperties*) [pure virtual]

Set the properties of the collection.

Parameters

in	<i>inProperties</i>	Collection properties
----	---------------------	-----------------------

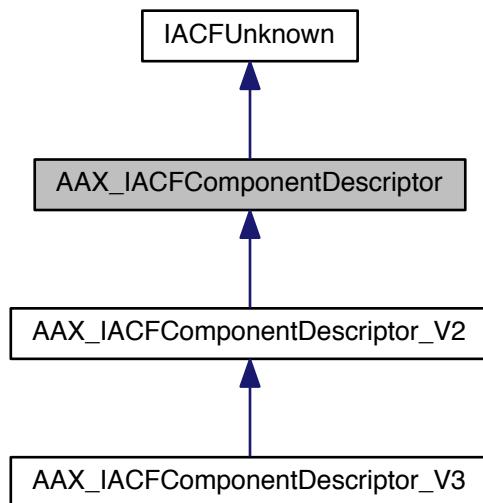
The documentation for this class was generated from the following file:

- [AAX_IACFCollection.h](#)

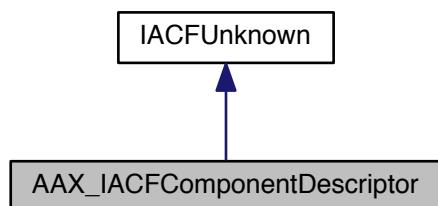
14.43 AAX_IACFComponentDescriptor Class Reference

```
#include <AAX_IACFComponentDescriptor.h>
```

Inheritance diagram for AAX_IACFComponentDescriptor:



Collaboration diagram for AAX_IACFComponentDescriptor:



14.43.1 Description

Versioned description interface for an AAX plug-in algorithm callback.

Public Member Functions

- virtual [AAx_Result Clear \(\)=0](#)
Clears the descriptor.
- virtual [AAx_Result AddReservedField \(AAx_CFieldIndex inFieldIndex, uint32_t inFieldType\)=0](#)
Subscribes a context field to host-provided services or information.
- virtual [AAx_Result AddAudioIn \(AAx_CFieldIndex inFieldIndex\)=0](#)
Subscribes an audio input context field.
- virtual [AAx_Result AddAudioOut \(AAx_CFieldIndex inFieldIndex\)=0](#)
Subscribes an audio output context field.
- virtual [AAx_Result AddAudioBufferLength \(AAx_CFieldIndex inFieldIndex\)=0](#)
Subscribes a buffer length context field.
- virtual [AAx_Result AddSampleRate \(AAx_CFieldIndex inFieldIndex\)=0](#)
Subscribes a sample rate context field.
- virtual [AAx_Result AddClock \(AAx_CFieldIndex inFieldIndex\)=0](#)
Subscribes a clock context field.
- virtual [AAx_Result AddSideChainIn \(AAx_CFieldIndex inFieldIndex\)=0](#)
Subscribes a side-chain input context field.
- virtual [AAx_Result AddDataInPort \(AAx_CFieldIndex inFieldIndex, uint32_t inPacketSize, AAX_EDataInPortType inPortType\)=0](#)
Adds a custom data port to the algorithm context.
- virtual [AAx_Result AddAuxOutputStem \(AAx_CFieldIndex inFieldIndex, int32_t inStemFormat, const char inNameUTF8\[\]\)=0](#)
Adds an auxiliary output stem for a plug-in.
- virtual [AAx_Result AddPrivateData \(AAx_CFieldIndex inFieldIndex, int32_t inDataSize, uint32_t inOptions=AAX_ePrivateDataOptions_DefaultOptions\)=0](#)
Adds a private data port to the algorithm context.
- virtual [AAx_Result AddDmaInstance \(AAx_CFieldIndex inFieldIndex, AAX_IDma::EMode inDmaMode\)=0](#)
Adds a DMA field to the plug-in's context.
- virtual [AAx_Result AddMIDINode \(AAx_CFieldIndex inFieldIndex, AAX_EMIDINodeType inNodeType, const char inNodeName\[\], uint32_t channelMask\)=0](#)
Adds a MIDI node field to the plug-in's context.
- virtual [AAx_Result AddProcessProc_Native \(AAx_CProcessProc inProcessProc, IACFUnknown *inProperties, AAX_CInstanceInitProc inInstanceInitProc, AAX_CBackgroundProc inBackgroundProc, AAX_CSelector *outProcID\)=0](#)
Registers an algorithm processing entrypoint (process procedure) for the native architecture.
- virtual [AAx_Result AddProcessProc_TI \(const char inDLLFileNameUTF8\[\], const char inProcessProcSymbol\[\], IACFUnknown *inProperties, const char inInstanceInitProcSymbol\[\], const char inBackgroundProcSymbol\[\], AAX_CSelector *outProcID\)=0](#)
Registers an algorithm processing entrypoint (process procedure) for the native architecture.
- virtual [AAx_Result AddMeters \(AAx_CFieldIndex inFieldIndex, const AAX_CTypeID *inMeterIDs, const uint32_t inMeterCount\)=0](#)
Adds a meter field to the plug-in's context.

14.43.2 Member Function Documentation

14.43.2.1 virtual AAX_Result AAX_IACFComponentDescriptor::Clear() [pure virtual]

Clears the descriptor.

Clears the descriptor and readies it for the next algorithm description

14.43.2.2 virtual AAX_Result AAX_IACFComponentDescriptor::AddReservedField(AAX_CFieldIndex *inFieldIndex*, uint32_t *inFieldType*) [pure virtual]

Subscribes a context field to host-provided services or information.

Note

Currently for internal use only.

Parameters

in	<i>inFieldIndex</i>	Unique identifier for the field, generated using AAX_FIELD_INDEX
in	<i>inFieldType</i>	Type of field that is being added

14.43.2.3 virtual AAX_Result AAX_IACFComponentDescriptor::AddAudioIn(AAX_CFieldIndex *inFieldIndex*) [pure virtual]

Subscribes an audio input context field.

Defines an audio in port for host-provided information in the algorithm's context structure.

- Data type: float**
- Data kind: An array of float arrays, one for each input channel

Parameters

in	<i>inFieldIndex</i>	Unique identifier for the field, generated using AAX_FIELD_INDEX
----	---------------------	--

14.43.2.4 virtual AAX_Result AAX_IACFComponentDescriptor::AddAudioOut(AAX_CFieldIndex *inFieldIndex*) [pure virtual]

Subscribes an audio output context field.

Defines an audio out port for host-provided information in the algorithm's context structure.

- Data type: float**
- Data kind: An array of float arrays, one for each output channel

Parameters

in	<i>inFieldIndex</i>	Unique identifier for the field, generated using AAX_FIELD_INDEX
----	---------------------	--

14.43.2.5 virtual AAX_Result AAX_IACFComponentDescriptor::AddAudioBufferLength(AAX_CFieldIndex *inFieldIndex*) [pure virtual]

Subscribes a buffer length context field.

Defines a buffer length port for host-provided information in the algorithm's context structure.

- Data type: int32_t*
- Data kind: The number of samples in the current audio buffer

Parameters

in	<i>inFieldIndex</i>	Unique identifier for the field, generated using AAX_FIELD_INDEX
----	---------------------	--

14.43.2.6 virtual AAX_Result AAX_IACFComponentDescriptor::AddSampleRate (AAX_CFieldIndex *inFieldIndex*) [pure virtual]

Subscribes a sample rate context field.

Defines a sample rate port for host-provided information in the algorithm's context structure.

- Data type: [AAX_CSsampleRate](#) *
- Data kind: The current sample rate

Parameters

in	<i>inFieldIndex</i>	Unique identifier for the field, generated using AAX_FIELD_INDEX
----	---------------------	--

14.43.2.7 virtual AAX_Result AAX_IACFComponentDescriptor::AddClock (AAX_CFieldIndex *inFieldIndex*) [pure virtual]

Subscribes a clock context field.

Defines a clock port for host-provided information in the algorithm's context structure.

- Data type: [AAX_CTimestamp](#) *
- Data kind: A running counter which increments even when the transport is not playing. The counter increments exactly once per sample quantum.

Host Compatibility Notes As of Pro Tools 11.1, this field may be used in both Native and DSP plug-ins. The DSP clock data is a 16-bit cycling counter. This field was only available for Native plug-ins in previous Pro Tools versions.

Parameters

in	<i>inFieldIndex</i>	Unique identifier for the field, generated using AAX_FIELD_INDEX
----	---------------------	--

14.43.2.8 virtual AAX_Result AAX_IACFComponentDescriptor::AddSideChainIn (AAX_CFieldIndex *inFieldIndex*) [pure virtual]

Subscribes a side-chain input context field.

Defines a side-chain input port for host-provided information in the algorithm's context structure.

- Data type: int32_t*
- Data kind: The index of the plug-in's first side-chain input channel within the array of input audio buffers

Parameters

in	<i>inFieldIndex</i>	Unique identifier for the field, generated using AAX_FIELD_INDEX
----	---------------------	--

14.43.2.9 virtual [AAX_Result](#) [AAX_IACFComponentDescriptor::AddDataInPort](#)([AAX_CFieldIndex](#) *inFieldIndex*, [uint32_t](#) *inPacketSize*, [AAX_EDataInPortType](#) *inPortType*) [pure virtual]

Adds a custom data port to the algorithm context.

Defines a read-only data port for plug-in information in the algorithm's context structure. The plug-in can send information to this port using [AAX_IController::PostPacket\(\)](#).

The host guarantees that all packets will be delivered to this port in the order in which they were posted, up to the point of a packet buffer overflow, though some packets may be dropped depending on the *inPortType* and host implementation.

Note

When a plug-in is operating in offline (AudioSuite) mode, all data ports operate as [AAX_eDataInPortType_](#)→ [Unbuffered](#) ports

Parameters

in	<i>inFieldIndex</i>	Unique identifier for the port, generated using AAX_FIELD_INDEX
in	<i>inPacketSize</i>	Size of the data packets that will be sent to this port
in	<i>inPortType</i>	The requested packet delivery behavior for this port

14.43.2.10 virtual [AAX_Result](#) [AAX_IACFComponentDescriptor::AddAuxOutputStem](#)([AAX_CFieldIndex](#) *inFieldIndex*, [int32_t](#) *inStemFormat*, const char *inNameUTF8[]*) [pure virtual]

Adds an auxiliary output stem for a plug-in.

Use this method to add additional output channels to the algorithm context.

The aux output stem audio buffers will be added to the end of the audio outputs array in the order in which they are described. When writing audio data to a specific aux output, find the proper starting channel by accumulating all of the channels of the main output stem format and any previously-described aux output stems.

The plug-in is responsible for providing a meaningful name for each aux outputs. At the very least, individual outputs should be labeled "Output xx", where "xx" is the aux output number as it is defined in the plug-in. The output name should also include the words "mono" and "stereo" to support when users are looking for an output with a specific stem format.

Host Compatibility Notes There is a hard limit to the number of outputs that Pro Tools supports for a single plug-in instance. This limit is currently set at 256 channels, which includes all of the plug-in's output channels in addition to the sum total of all of its aux output stem channels.

Host Compatibility Notes Pro Tools supports only mono and stereo auxiliary output stem formats

Warning

This method will return an error code on hosts which do not support auxiliary output stems. This indicates that the host will not provide audio buffers for auxiliary output stems during processing. A plug-in must not attempt to write data into auxiliary output stem buffers which have not been provided by the host!

Parameters

in	<i>inFieldIndex</i>	DEPRECATED: This parameter is no longer needed by the host, but is included in the interface for binary compatibility
in	<i>inStemFormat</i>	The stem format of the new aux output
in	<i>inNameUTF8</i>	The name of the aux output. This name is static and cannot be changed after the descriptor is submitted to the host

```
14.43.2.11 virtual AAX_Result AAX_IACFComponentDescriptor::AddPrivateData ( AAX_CFieldIndex inFieldIndex,  
int32_t inDataSize, uint32_t inOptions = AAX_ePrivateDataOptions_DefaultOptions ) [pure  
virtual]
```

Adds a private data port to the algorithm context.

Defines a read/write data port for private state data. Data written to this port will be maintained by the host between calls to the algorithm context.

See also

[alg_pd_registration](#)

Parameters

in	<i>inFieldIndex</i>	Unique identifier for the port, generated using AAX_FIELD_INDEX
in	<i>inDataSize</i>	Size of the data packets that will be sent to this port
in	<i>inOptions</i>	Options that define the private data port's behavior

```
14.43.2.12 virtual AAX_Result AAX_IACFComponentDescriptor::AddDmaInstance ( AAX_CFieldIndex inFieldIndex,  
AAX_IDma::EMode inDmaMode ) [pure virtual]
```

Adds a DMA field to the plug-in's context.

DMA (direct memory access) provides efficient reads from and writes to external memory on the DSP. DMA behavior is emulated in host-based plug-ins for cross-platform portability.

Note

The order in which DMA instances are added defines their priority and therefore order of execution of DMA operations. In most plug-ins, Scatter fields should be placed first in order to achieve the lowest possible access latency.

For more information, see [Direct Memory Access](#).

Todo Update the DMA system management such that operation priority can be set arbitrarily

Parameters

in	<i>inFieldIndex</i>	Unique identifier for the field, generated using AAX_FIELD_INDEX
in	<i>inDmaMode</i>	AAX_IDma::EMode that will apply to this field

```
14.43.2.13 virtual AAX_Result AAX_IACFComponentDescriptor::AddMIDINode ( AAX_CFieldIndex inFieldIndex,  
AAX_EMIDINodeType inNodeType, const char innodeName[ ], uint32_t channelMask ) [pure  
virtual]
```

Adds a MIDI node field to the plug-in's context.

- Data type: [AAX_IMIDINode *](#)

The resulting MIDI node data will be available both in the algorithm context and in the plug-in's [data model](#) via [UpdateMIDINodes\(\)](#).

To add a MIDI node that is only accessible to the plug-in's data model, use [AAX_IEffectDescriptor::AddControlMIDI→DINode\(\)](#)

Host Compatibility Notes Due to current restrictions MIDI data won't be delivered to DSP algorithms, only to AAX Native.

Parameters

in	<i>inFieldIndex</i>	The ID of the port. MIDI node ports should be formatted as a pointer to an AA→X_IMIDINode .
in	<i>inNodeType</i>	The type of MIDI node, as AAX_EMIDINodeType
in	<i>innodeName</i>	The name of the MIDI node as it should appear in the host's UI
in	<i>channelMask</i>	The channel mask for the MIDI node. This parameter specifies used MIDI channels. For Global MIDI nodes, use a mask of AAX_EMidiGlobalNode→Selectors

14.43.2.14 virtual AAX_Result AAX_IACFComponentDescriptor::AddProcessProc_Native (AAX_CProcessProc
inProcessProc, IACFUnknown * *inProperties*, AAX_CInstanceInitProc *inInstanceInitProc*,
AAX_CBackgroundProc *inBackgroundProc*, AAX_CSelector * *outProcID*) [pure virtual]

Registers an algorithm processing entrypoint (process procedure) for the native architecture.

Parameters

in	<i>inProcessProc</i>	Symbol for this processing callback
in	<i>inProperties</i>	A property map for this processing callback. The property map's values are copied by the host and associated with the new ProcessProc. The property map contents are unchanged and the map may be re-used when registering additional ProcessProcs.
in	<i>inInstanceInit→Proc</i>	Initialization routine that will be called when a new instance of the Effect is created. See Algorithm initialization .
in	<i>inBackground→Proc</i>	Background routine that will be called in an idle context within the same address space as the associated process procedure. See Background processing callback
out	<i>outProcID</i>	

Todo document this parameter

14.43.2.15 virtual AAX_Result AAX_IACFComponentDescriptor::AddProcessProc_TI (const char *inDLLFileNameUTF8*[],
const char *inProcessProcSymbol*[], IACFUnknown * *inProperties*, const char *inInstanceInitProcSymbol*[], const
char *inBackgroundProcSymbol*[], AAX_CSelector * *outProcID*) [pure virtual]

Registers an algorithm processing entrypoint (process procedure) for the native architecture.

Parameters

in	<i>inDLLFile→NameUTF8</i>	UTF-8 encoded filename for the ELF DLL containing the algorithm code fragment
----	---------------------------	---

in	<i>inProcessProcSymbol</i>	Symbol for this processing callback
in	<i>inProperties</i>	A property map for this processing callback. The property map's values are copied by the host and associated with the new ProcessProc. The property map contents are unchanged and the map may be re-used when registering additional ProcessProcs.
in	<i>inInstanceInitProcSymbol</i>	Initialization routine that will be called when a new instance of the Effect is created. Must be included in the same DLL as the main algorithm entrypoint. See Algorithm initialization .
in	<i>inBackgroundProcSymbol</i>	Background routine that will be called in an idle context within the same address space as the associated process procedure. Must be included in the same DLL as the main algorithm entrypoint. See Background processing callback
out	<i>outProcID</i>	

Todo document this parameter

14.43.2.16 virtual AAX_Result AAX_IACFComponentDescriptor::AddMeters(AAX_CFieldIndex *inFieldIndex*, const AAX_CTypeID * *inMeterIDs*, const uint32_t *inMeterCount*) [pure virtual]

Adds a meter field to the plug-in's context.

Meter fields include an array of meter tap values, with one tap per meter per context. Only one meter field should be added per Component. Individual meter behaviors can be described at the Effect level.

For more information, see [Plug-in meters](#) .

Parameters

in	<i>inFieldIndex</i>	Unique identifier for the field, generated using AAX_FIELD_INDEX
in	<i>inMeterIDs</i>	Array of 32-bit IDs, one for each meter. Meter IDs must be unique within the Effect.
in	<i>inMeterCount</i>	The number of meters included in this field

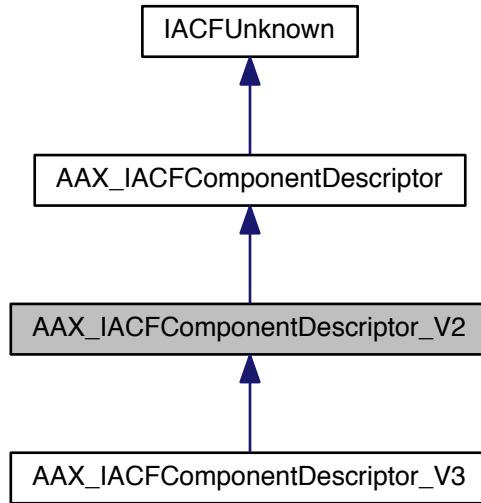
The documentation for this class was generated from the following file:

- [AAX_IACFComponentDescriptor.h](#)

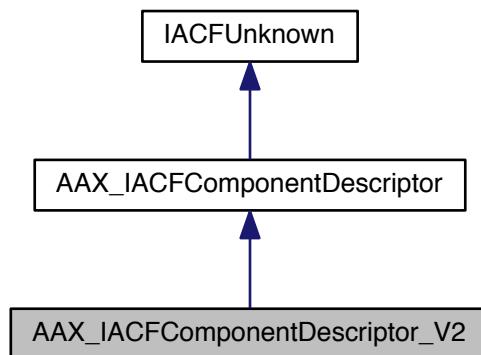
14.44 AAX_IACFComponentDescriptor_V2 Class Reference

```
#include <AAX_IACFComponentDescriptor.h>
```

Inheritance diagram for AAX_IACFComponentDescriptor_V2:



Collaboration diagram for AAX_IACFComponentDescriptor_V2:



14.44.1 Description

Versioned description interface for an AAX plug-in algorithm callback.

Public Member Functions

- virtual [AAAX_Result AddTemporaryData \(AAAX_CFieldIndex inFieldIndex, uint32_t inDataElementSize\)=0](#)
Adds a block of data to a context that is not saved between callbacks and is scaled by the system buffer size.

14.44.2 Member Function Documentation

14.44.2.1 virtual AAX_Result AAX_IACFComponentDescriptor_V2::AddTemporaryData (AAX_CFieldIndex *inFieldIndex*, uint32_t *inDataElementSize*) [pure virtual]

Adds a block of data to a context that is not saved between callbacks and is scaled by the system buffer size.

This can be very useful if you use block processing and need to store intermediate results. Just specify your base element size and the system will scale the overall block size by the buffer size. For example, to create a buffer of floats that is the length of the block, specify 4 bytes as the elementsize.

This data block does not retain state across callback and can also be reused across instances on memory contrained systems.

Parameters

in	<i>inFieldIndex</i>	Unique identifier for the port, generated using AAX_FIELD_INDEX
in	<i>inDataElementSize</i>	The size of a single piece of data in the block. This number will be multiplied by the processing block size to determine total block size.

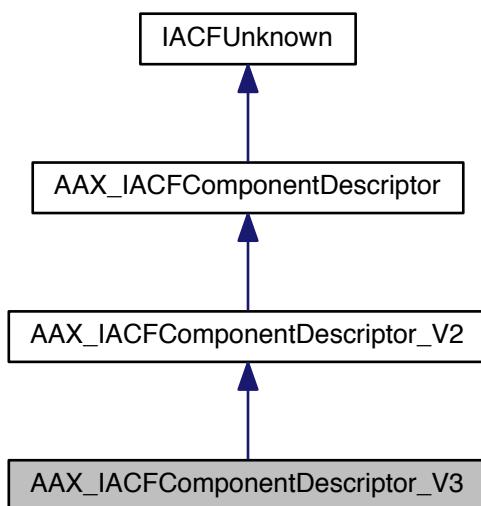
The documentation for this class was generated from the following file:

- [AAX_IACFComponentDescriptor.h](#)

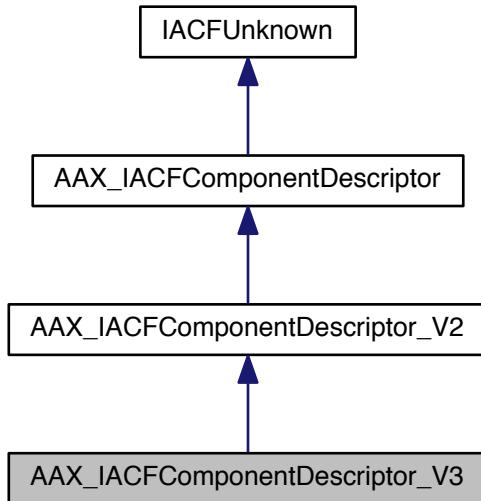
14.45 AAX_IACFComponentDescriptor_V3 Class Reference

```
#include <AAX_IACFComponentDescriptor.h>
```

Inheritance diagram for AAX_IACFComponentDescriptor_V3:



Collaboration diagram for AAX_IACFComponentDescriptor_V3:



14.45.1 Description

Versioned description interface for an AAX plug-in algorithm callback.

Public Member Functions

- virtual [AAX_Result AddProcessProc \(IACFUnknown *inProperties, AAX_CSelector *outProcIDs, int32_t inProcIDsSize\)=0](#)
Registers one or more algorithm processing entrypoints (process procedures)

14.45.2 Member Function Documentation

14.45.2.1 virtual [AAX_Result AAX_IACFComponentDescriptor_V3::AddProcessProc \(IACFUnknown * inProperties, AAX_CSelector * outProcIDs, int32_t inProcIDsSize \) \[pure virtual\]](#)

Registers one or more algorithm processing entrypoints (process procedures)

Any non-overlapping set of processing entrypoints may be specified. Typically this can be used to specify both Native and TI entrypoints using the same call.

The AAX Library implementation of this method includes backwards compatibility logic to complete the Process→Proc registration on hosts which do not support this method. Therefore plug-in code may use this single registration routine instead of separate calls to [AddProcessProc_Native\(\)](#), [AddProcessProc_TI\(\)](#), etc. regardless of the host version.

The following properties replace the input arguments to the platform-specific registration methods:

[AddProcessProc_Native\(\)](#) ([AAX_eProperty_PlugInID_Native](#), [AAX_eProperty_PlugInID_AudioSuite](#))

- [AAX_CProcessProc iProcessProc: AAX_eProperty_NativeProcessProc](#) (required)
- [AAX_CInstanceInitProc iInstanceInitProc: AAX_eProperty_NativeInstanceInitProc](#) (optional)

- `AAX_CBackgroundProc` `iBackgroundProc`: `AAX_eProperty_NativeBackgroundProc` (optional)

`AddProcessProc_TI()` (`AAX_eProperty_PluginID_TI`)

- `const char inDLLFileNameUTF8[]`: `AAX_eProperty_TIDLLFileName` (required)
- `const char iProcessProcSymbol[]`: `AAX_eProperty_TIProcessProc` (required)
- `const char iInstanceInitProcSymbol[]`: `AAX_eProperty_TIInstanceInitProc` (optional)
- `const char iBackgroundProcSymbol[]`: `AAX_eProperty_TIBackgroundProc` (optional)

If any platform-specific plug-in ID property is present in `iProperties` then `AddProcessProc()` will check for the required properties for that platform.

Note

`AAX_eProperty_AudioBufferLength` will be ignored for the Native and AudioSuite ProcessProcs since it should only be used for AAX DSP.

Parameters

in	<code>inProperties</code>	A property map for this processing callback. The property map's values are copied by the host and associated with the new ProcessProc. The property map contents are unchanged and the map may be re-used when registering additional ProcessProcs.
out	<code>outProcIDs</code>	

Todo document this parameter Returned array will be NULL-terminated

Parameters

in	<code>inProcIDsSize</code>	The size of the array provided to <code>oProcIDs</code> . If <code>oProcIDs</code> is non-NULL but <code>iProcIDsSize</code> is not large enough for all of the registered ProcessProcs (plus one for NULL termination) then this method will fail with <code>AAX_ERROR_ARGUMENT_OVERFLOW</code>
----	----------------------------	--

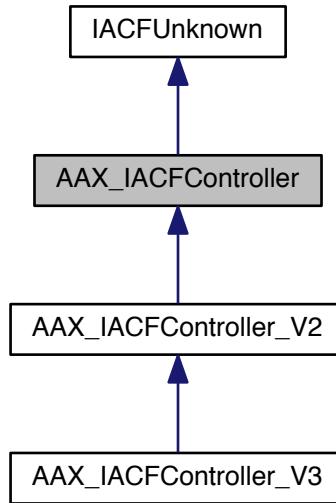
The documentation for this class was generated from the following file:

- `AAX_IACFComponentDescriptor.h`

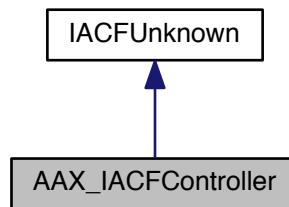
14.46 AAX_IACFController Class Reference

```
#include <AAX_IACFController.h>
```

Inheritance diagram for AAX_IACFController:



Collaboration diagram for AAX_IACFController:



14.46.1 Description

Interface for the AAX host's view of a single instance of an effect. Used by both clients of the AAXHost and by effect components.

Public Member Functions

- virtual [AAX_Result GetEffectID \(AAX_IString *outEffectID\) const =0](#)
- virtual [AAX_Result GetSampleRate \(AAX_CSampleRate *outSampleRate\) const =0](#)
CALL: Returns the current literal sample rate.
- virtual [AAX_Result GetInputStemFormat \(AAX_EStemFormat *outStemFormat\) const =0](#)
CALL: Returns the plug-in's input stem format.

- virtual `AAX_Result GetOutputStemFormat (AAX_EStemFormat *outStemFormat) const =0`
`CALL: Returns the plug-in's output stem format.`
- virtual `AAX_Result GetSignalLatency (int32_t *outSamples) const =0`
`CALL: Returns the most recent signal (algorithmic) latency that has been published by the plug-in.`
- virtual `AAX_Result GetCycleCount (AAX_EProperty inWhichCycleCount, AAX_CPropertyValue *outNumCycles) const =0`
`CALL: returns the plug-in's current real-time DSP cycle count.`
- virtual `AAX_Result GetTODLocation (AAX_CTimeOfDay *outTODLocation) const =0`
`CALL: Returns the current Time Of Day (TOD) of the system.`
- virtual `AAX_Result SetSignalLatency (int32_t inNumSamples)=0`
`CALL: Submits a request to change the delay compensation value that the host uses to account for the plug-in's signal (algorithmic) latency.`
- virtual `AAX_Result SetCycleCount (AAX_EProperty *inWhichCycleCounts, AAX_CPropertyValue *iValues, int32_t numValues)=0`
`CALL: Indicates a change in the plug-in's real-time DSP cycle count.`
- virtual `AAX_Result PostPacket (AAX_CFieldIndex inFieldIndex, const void *inPayloadP, uint32_t inPayloadSize)=0`
`CALL: Posts a data packet to the host for routing between plug-in components.`
- virtual `AAX_Result GetCurrentMeterValue (AAC_CTypeID inMeterID, float *outMeterValue) const =0`
`CALL: Retrieves the current value of a host-managed plug-in meter.`
- virtual `AAX_Result GetMeterPeakValue (AAC_CTypeID inMeterID, float *outMeterPeakValue) const =0`
`CALL: Retrieves the currently held peak value of a host-managed plug-in meter.`
- virtual `AAX_Result ClearMeterPeakValue (AAC_CTypeID inMeterID) const =0`
`CALL: Clears the peak value from a host-managed plug-in meter.`
- virtual `AAX_Result GetMeterClipped (AAC_CTypeID inMeterID, AAC_CBoolean *outClipped) const =0`
`CALL: Retrieves the clipped flag from a host-managed plug-in meter.`
- virtual `AAX_Result ClearMeterClipped (AAC_CTypeID inMeterID) const =0`
`CALL: Clears the clipped flag from a host-managed plug-in meter.`
- virtual `AAX_Result GetMeterCount (uint32_t *outMeterCount) const =0`
`CALL: Retrieves the number of host-managed meters registered by a plug-in.`
- virtual `AAX_Result GetNextMIDIpacket (AAX_CFieldIndex *outPort, AAC_CMidiPacket *outPacket)=0`
`CALL: Retrieves MIDI packets for described MIDI nodes.`

14.46.2 Member Function Documentation

14.46.2.1 virtual `AAX_Result AAC_IACFController::GetEffectID (AAC_IString * outEffectID) const [pure virtual]`

14.46.2.2 virtual `AAX_Result AAC_IACFController::GetSampleRate (AAC_CSampleRate * outSampleRate) const [pure virtual]`

CALL: Returns the current literal sample rate.

Parameters

<code>out</code>	<code>outSampleRate</code>	The current sample rate
------------------	----------------------------	-------------------------

14.46.2.3 virtual `AAX_Result AAC_IACFController::GetInputStemFormat (AAC_EStemFormat * outStemFormat) const [pure virtual]`

CALL: Returns the plug-in's input stem format.

Parameters

<code>out</code>	<code>outStemFormat</code>	The current input stem format
------------------	----------------------------	-------------------------------

14.46.2.4 virtual **AAX_Result** **AAX_IACFController::GetOutputStemFormat** (**AAX_EStemFormat** * *outStemFormat*)
const [pure virtual]

CALL: Returns the plug-in's output stem format.

Parameters

<code>out</code>	<code>outStemFormat</code>	The current output stem format
------------------	----------------------------	--------------------------------

14.46.2.5 virtual **AAX_Result** **AAX_IACFController::GetSignalLatency** (**int32_t** * *outSamples*) **const** [pure virtual]

CALL: Returns the most recent signal (algorithmic) latency that has been published by the plug-in.

This method provides the most recently published signal latency. The host may not have updated its delay compensation to match this signal latency yet, so plug-ins that dynamically change their latency using [SetSignalLatency\(\)](#) should always wait for an [AAX_eNotificationEvent_SignalLatencyChanged](#) notification before updating its algorithm to incur this latency.

See also

[SetSignalLatency\(\)](#)

Parameters

<code>out</code>	<code>outSamples</code>	The number of samples of signal delay published by the plug-in
------------------	-------------------------	--

14.46.2.6 virtual **AAX_Result** **AAX_IACFController::GetCycleCount** (**AAX_EProperty** *inWhichCycleCount*,
AAX_CPropertyValue * *outNumCycles*) **const** [pure virtual]

CALL: returns the plug-in's current real-time DSP cycle count.

This method provides the number of cycles that the AAX host expects the DSP plug-in to consume. The host uses this value when allocating DSP resources for the plug-in.

Note

A plug-in should never apply a DSP algorithm with more demanding resource requirements than what is currently accounted for by the host. To set a higher cycle count value, a plug-in must call [AAX_IController::SetCycleCount\(\)](#), then poll [AAX_IController::GetCycleCount\(\)](#) until the new value has been applied. Once the host has recognized the new cycle count value, the plug-in may apply the more demanding algorithm.

Parameters

<code>in</code>	<code>inWhichCycleCount</code>	Selector for the requested cycle count metric. One of: <ul style="list-style-type: none">• AAX_eProperty_TI_SharedCycleCount• AAX_eProperty_TI_InstanceCycleCount• AAX_eProperty_TI_MaxInstancesPerChip
<code>in</code>	<code>outNumCycles</code>	The current value of the selected cycle count metric

Todo PLACEHOLDER - NOT CURRENTLY IMPLEMENTED IN HOST

14.46.2.7 virtual AAX_Result AAX_IACFController::GetTODLocation (*AAX_CTimeOfDay * outTODLocation*) const
 [pure virtual]

CALL: Returns the current Time Of Day (TOD) of the system.

This method provides a plug-in the TOD (in samples) of the current system. TOD is the number of samples that the playhead has traversed since the beginning of playback.

Note

The TOD value is the immediate value of the audio engine playhead. This value is incremented within the audio engine's real-time rendering context; it is not synchronized with non-real-time calls to plug-in interface methods.

Parameters

<i>out</i>	<i>outTODLocation</i>	The current Time Of Day as set by the host
------------	-----------------------	--

14.46.2.8 virtual AAX_Result AAX_IACFController::SetSignalLatency (*int32_t inNumSamples*) [pure virtual]

CALL: Submits a request to change the delay compensation value that the host uses to account for the plug-in's signal (algorithmic) latency.

This method is used to request a change in the number of samples that the AAX host expects the plug-in to delay a signal.

The host is not guaranteed to immediately apply the new latency value. A plug-in should avoid incurring an actual algorithmic latency that is different than the latency accounted for by the host.

To set a new latency value, a plug-in must call [AAX_IController::SetSignalLatency\(\)](#), then wait for an [AAX_eNotificationEvent_SignalLatencyChanged](#) notification. Once this notification has been received, [AAX_IController::GetSignalLatency\(\)](#) will reflect the updated latency value and the plug-in should immediately apply any relevant algorithmic changes that alter its latency to this new value.

Warning

Parameters which affect the latency of a plug-in should not be made available for control through automation. This will result in audible glitches when delay compensation is adjusted while playing back automation for these parameters.

Parameters

<i>in</i>	<i>inNumSamples</i>	The number of samples of signal delay that the plug-in requests to incur
-----------	---------------------	--

14.46.2.9 virtual AAX_Result AAX_IACFController::SetCycleCount (*AAX_EProperty * inWhichCycleCounts,*
*AAX_CPropertyValue * iValues, int32_t numValues*) [pure virtual]

CALL: Indicates a change in the plug-in's real-time DSP cycle count.

This method is used to request a change in the number of cycles that the AAX host expects the DSP plug-in to consume.

Note

A plug-in should never apply a DSP algorithm with more demanding resource requirements than what is currently accounted for by the host. To set a higher cycle count value, a plug-in must call [AAX_IController::SetCycleCount\(\)](#), then poll [AAX_IController::GetCycleCount\(\)](#) until the new value has been applied. Once the host has recognized the new cycle count value, the plug-in may apply the more demanding algorithm.

Parameters

in	<i>inWhichCycleCounts</i>	Array of selectors indicating the specific cycle count metrics that should be set. Each selector must be one of: <ul style="list-style-type: none"> • AAX_eProperty_TI_SharedCycleCount • AAX_eProperty_TI_InstanceCycleCount • AAX_eProperty_TI_MaxInstancesPerChip
in	<i>iValues</i>	An array of values requested, one for each of the selected cycle count metrics.
in	<i>numValues</i>	The size of <i>iValues</i>

Todo PLACEHOLDER - NOT CURRENTLY IMPLEMENTED IN HOST

14.46.2.10 virtual [AAX_Result](#) [AAX_IACFController::PostPacket](#)([AAX_CFieldIndex](#) *inFieldIndex*, const void * *inPayloadP*, [uint32_t](#) *inPayloadSize*) [pure virtual]

CALL: Posts a data packet to the host for routing between plug-in components.

The posted packet is identified with a [AAX_CFieldIndex](#) packet index value, which is equivalent to the target data port's identifier. The packet's payload must have the expected size for the given packet index / data port, as defined when the port is created in [Describe](#). See [AAX_IComponentDescriptor::AddDataInPort\(\)](#).

Warning

Any data structures that will be passed between platforms (for example, sent to a TI DSP in an AAX DSP plug-in) must be properly data-aligned for compatibility across both platforms. See [AAX_ALIGN_FILE_ALG](#) for more information about guaranteeing cross-platform compatibility of data structures used for algorithm processing.

Note

All calls to this method should be made within the scope of [AAX_IEffectParameters::GenerateCoefficients\(\)](#). Calls from outside this method may result in packets not being delivered. See [PT-206161](#)

Parameters

in	<i>inFieldIndex</i>	The packet's destination port
in	<i>inPayloadP</i>	A pointer to the packet's payload data
in	<i>inPayloadSize</i>	The size, in bytes, of the payload data

14.46.2.11 virtual [AAX_Result](#) [AAX_IACFController::GetCurrentMeterValue](#)([AAX_CTypeID](#) *inMeterID*, float * *outMeterValue*) const [pure virtual]

CALL: Retrieves the current value of a host-managed plug-in meter.

Parameters

in	<i>inMeterID</i>	ID of the meter that is being queried
out	<i>outMeterValue</i>	The queried meter's current value

14.46.2.12 virtual [AAX_Result](#) [AAX_IACFController::GetMeterPeakValue](#)([AAX_CTypeID](#) *inMeterID*, float * *outMeterPeakValue*) const [pure virtual]

CALL: Retrieves the currently held peak value of a host-managed plug-in meter.

Parameters

in	<i>inMeterID</i>	ID of the meter that is being queried
out	<i>outMeterPeakValue</i>	The queried meter's currently held peak value

14.46.2.13 virtual AAX_Result AAX_IACFController::ClearMeterPeakValue (AAX_CTypeID *inMeterID*) const [pure virtual]

CALL: Clears the peak value from a host-managed plug-in meter.

Parameters

in	<i>inMeterID</i>	ID of the meter that is being cleared
----	------------------	---------------------------------------

14.46.2.14 virtual AAX_Result AAX_IACFController::GetMeterClipped (AAX_CTypeID *inMeterID*, AAX_CBoolean * *outClipped*) const [pure virtual]

CALL: Retrieves the clipped flag from a host-managed plug-in meter.

See [AAX_IComponentDescriptor::AddMeters\(\)](#).

Parameters

in	<i>inMeterID</i>	ID of the meter that is being queried.
out	<i>outClipped</i>	The queried meter's clipped flag.

14.46.2.15 virtual AAX_Result AAX_IACFController::ClearMeterClipped (AAX_CTypeID *inMeterID*) const [pure virtual]

CALL: Clears the clipped flag from a host-managed plug-in meter.

See [AAX_IComponentDescriptor::AddMeters\(\)](#).

Parameters

in	<i>inMeterID</i>	ID of the meter that is being cleared.
----	------------------	--

14.46.2.16 virtual AAX_Result AAX_IACFController::GetMeterCount (uint32_t * *outMeterCount*) const [pure virtual]

CALL: Retrieves the number of host-managed meters registered by a plug-in.

See [AAX_IComponentDescriptor::AddMeters\(\)](#).

Parameters

out	<i>outMeterCount</i>	The number of registered plug-in meters.
-----	----------------------	--

14.46.2.17 virtual AAX_Result AAX_IACFController::GetNextMIDIPacket (AAX_CFieldIndex * *outPort*, AAX_CMidiPacket * *outPacket*) [pure virtual]

CALL: Retrieves MIDI packets for described MIDI nodes.

Parameters

out	<i>outPort</i>	port ID of the MIDI node that has unhandled packet
out	<i>outPacket</i>	The MIDI packet

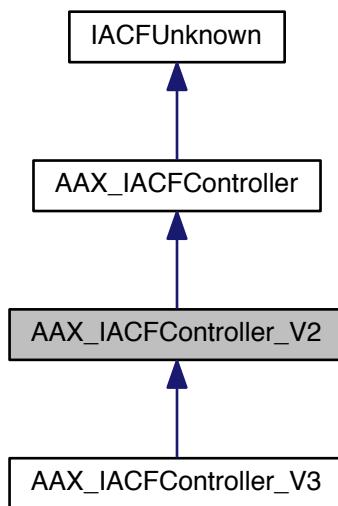
The documentation for this class was generated from the following file:

- [AAX_IACFController.h](#)

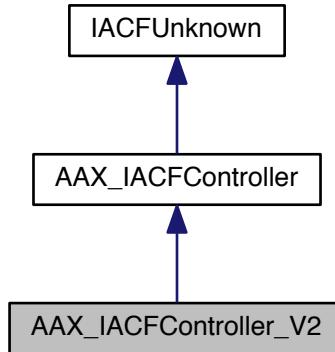
14.47 AAX_IACFController_V2 Class Reference

```
#include <AAX_IACFController.h>
```

Inheritance diagram for AAX_IACFController_V2:



Collaboration diagram for AAX_IACFController_V2:



14.47.1 Description

Interface for the AAX host's view of a single instance of an effect. Used by both clients of the AAXHost and by effect components.

Public Member Functions

- virtual [AAX_Result SendNotification \(AAX_CTypeID inNotificationType, const void *inNotificationData, uint32_t inNotificationDataSize\)=0](#)
CALL: Dispatch a notification.
- virtual [AAX_Result GetHybridSignalLatency \(int32_t *outSamples\) const =0](#)
CALL: Returns the latency between the algorithm normal input samples and the inputs returning from the hybrid component.
- virtual [AAX_Result GetCurrentAutomationTimestamp \(AAX_CTransportCounter *outTimestamp\) const =0](#)
CALL: Returns the current automation timestamp if called during the [GenerateCoefficients\(\)](#) call AND the generation of coefficients is being triggered by an automation point instead of immediate changes.
- virtual [AAX_Result GetHostName \(AAX_IString *outHostNameString\) const =0](#)
CALL: Returns name of the host application this plug-in instance is being loaded by. This string also typically includes version information.

14.47.2 Member Function Documentation

14.47.2.1 virtual [AAX_Result AAX_IACFController_V2::SendNotification \(AAX_CTypeID inNotificationType, const void *inNotificationData, uint32_t inNotificationDataSize \) \[pure virtual\]](#)

CALL: Dispatch a notification.

The notification is handled by the host and may be delivered back to other plug-in components such as the G_{UI} or data model (via [AAX_IEffectGUI::NotificationReceived\(\)](#) or [AAX_IEffectParameters::NotificationReceived\(\)](#), respectively) depending on the notification type.

The host may choose to dispatch the posted notification either synchronously or asynchronously.

See the [AAX_ENotificationEvent](#) documentation for more information.

This method is supported by AAX V2 Hosts only. Check the return code on the return of this function. If the error is [AAX_ERROR_UNIMPLEMENTED](#), your plug-in is being loaded into a host that doesn't support this feature.

Parameters

in	<i>inNotificationType</i>	Type of notification to send
in	<i>inNotificationData</i>	Block of notification data
in	<i>inNotificationDataSize</i>	Size of <i>inNotificationData</i> , in bytes

14.47.2.2 virtual AAX_Result AAX_IACFController_V2::GetHybridSignalLatency (int32_t * *outSamples*) const [pure virtual]

CALL: Returns the latency between the algorithm normal input samples and the inputs returning from the hybrid component.

This method provides the number of samples that the AAX host expects the plug-in to delay a signal. The host will use this value when accounting for latency across the system.

Note

This value will generally scale up with sample rate, although it's not a simple multiple due to some fixed overhead. This value will be fixed for any given sample rate regardless of other buffer size settings in the host app.

Parameters

out	<i>outSamples</i>	The number of samples of hybrid signal delay
-----	-------------------	--

14.47.2.3 virtual AAX_Result AAX_IACFController_V2::GetCurrentAutomationTimestamp (AAX_CTransportCounter * *outTimestamp*) const [pure virtual]

CALL: Returns the current automation timestamp if called during the [GenerateCoefficients\(\)](#) call AND the generation of coefficients is being triggered by an automation point instead of immediate changes.

Note

This function will return 0 if called from outside of [GenerateCoefficients\(\)](#) or if the [GenerateCoefficients\(\)](#) call was initiated due to a non-automated change. In those cases, you can get your sample offset from the transport start using [GetTODLocation\(\)](#).

Parameters

out	<i>outTimestamp</i>	The current coefficient timestamp. Sample count from transport start.
-----	---------------------	---

14.47.2.4 virtual AAX_Result AAX_IACFController_V2::GetHostName (AAX_IString * *outHostNameString*) const [pure virtual]

CALL: Returns name of the host application this plug-in instance is being loaded by. This string also typically includes version information.

Host Compatibility Notes Pro Tools versions from Pro Tools 11.0 to Pro Tools 12.3.1 will return a generic version string to this call. This issue is resolved beginning in Pro Tools 12.4.

Parameters

out	<i>outHostName</i> ↳ <i>String</i>	The name of the current host application.
-----	---------------------------------------	---

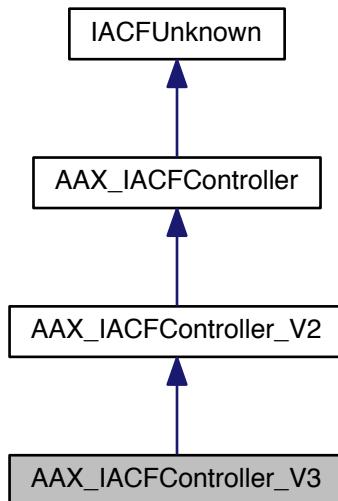
The documentation for this class was generated from the following file:

- [AAX_IACFController.h](#)

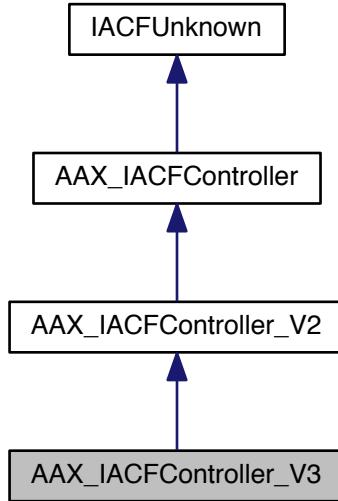
14.48 AAX_IACFController_V3 Class Reference

```
#include <AAX_IACFController.h>
```

Inheritance diagram for AAX_IACFController_V3:



Collaboration diagram for AAX_IACFController_V3:



14.48.1 Description

Interface for the AAX host's view of a single instance of an effect. Used by both clients of the AAXHost and by effect components.

Public Member Functions

- virtual [AAX_Result GetPlugInTargetPlatform \(AAX_CTargetPlatform *outTargetPlatform\) const =0](#)
CALL: Returns execution platform type, native or TI.
- virtual [AAX_Result GetIsAudioSuite \(AAX_CBoolean *outIsAudioSuite\) const =0](#)
CALL: Returns true for AudioSuite instances.

14.48.2 Member Function Documentation

14.48.2.1 virtual [AAX_Result AAX_IACFController_V3::GetPlugInTargetPlatform \(AAX_CTargetPlatform * outTargetPlatform \) const \[pure virtual\]](#)

CALL: Returns execution platform type, native or TI.

Parameters

out	<i>outTargetPlatform</i>	The type of the current execution platform as one of AAX_ETargetPlatform .
-----	--------------------------	--

14.48.2.2 virtual [AAX_Result AAX_IACFController_V3::GetIsAudioSuite \(AAX_CBoolean * outIsAudioSuite \) const \[pure virtual\]](#)

CALL: Returns true for AudioSuite instances.

Parameters

<code>out</code>	<code>outIsAudioSuite</code>	The boolean flag which indicate true for AudioSuite instances.
------------------	------------------------------	--

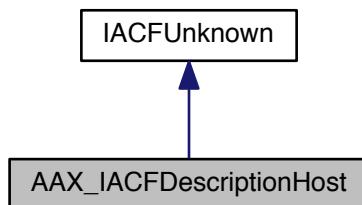
The documentation for this class was generated from the following file:

- [AAX_IACFController.h](#)

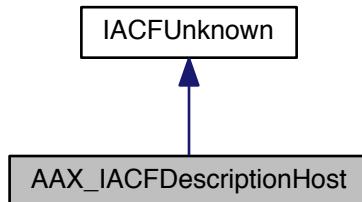
14.49 AAX_IACFDscriptionHost Class Reference

```
#include <AAX_IACFDscriptionHost.h>
```

Inheritance diagram for AAX_IACFDscriptionHost:



Collaboration diagram for AAX_IACFDscriptionHost:



14.49.1 Description

Interface to host services provided during plug-in description

Public Member Functions

- virtual [AAX_Result AcquireFeatureProperties](#) (const [AAX_Feature_UID](#) &inFeatureID, [IACFUnknown](#) **outFeatureProperties)=0

14.49.2 Member Function Documentation

14.49.2.1 `virtual AAX_Result AAX_IACFDescriptionHost::AcquireFeatureProperties (const AAX_Feature_UID & inFeatureID, IACFUnknown ** outFeatureProperties) [pure virtual]`

`outFeatureProperties` must support `AAX_IACFFeatureInfo` const methods

See also

[AAX_IDescriptionHost::AcquireFeatureProperties\(\)](#)

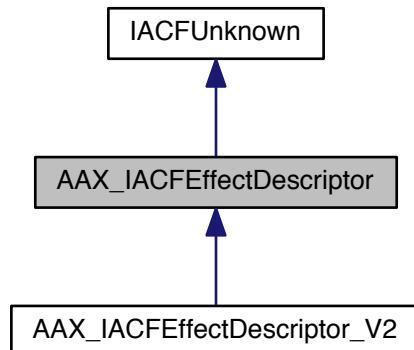
The documentation for this class was generated from the following file:

- [AAX_IACFDescriptionHost.h](#)

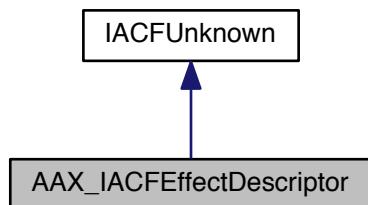
14.50 AAX_IACFEffectorDescriptor Class Reference

#include <`AAX_IACFEffectorDescriptor.h`>

Inheritance diagram for `AAX_IACFEffectorDescriptor`:



Collaboration diagram for `AAX_IACFEffectorDescriptor`:



14.50.1 Description

Versioned interface for an [AAX_IEffectDescriptor](#).

Public Member Functions

- virtual [AAX_Result AddComponent](#) ([IACFUnknown](#) **inComponentDescriptor*)=0
Add a component to an instance of a component descriptor.
- virtual [AAX_Result AddName](#) (const char **inPlugInName*)=0
Add a name to the Effect.
- virtual [AAX_Result AddCategory](#) (uint32_t *inCategory*)=0
Add a category to your plug-in. See [AAX_EPlugInCategory](#).
- virtual [AAX_Result AddCategoryBypassParameter](#) (uint32_t *inCategory*, [AAC_CParamID](#) *inParamID*)=0
Add a category to your plug-in. See [AAX_EPlugInCategory](#).
- virtual [AAX_Result AddProcPtr](#) (void **inProcPtr*, [AAC_CProcPtrID](#) *inProcID*)=0
Add a process pointer.
- virtual [AAX_Result SetProperties](#) ([IACFUnknown](#) **inProperties*)=0
Set the properties of a new property map.
- virtual [AAX_Result AddResourceInfo](#) ([AAC_EResourceType](#) *in ResourceType*, const char **inInfo*)=0
Set resource file info.
- virtual [AAX_Result AddMeterDescription](#) ([AAC_CTypeID](#) *inMeterID*, const char **inMeterName*, [IACFUnknown](#) **inProperties*)=0
Add name and property map to meter with given ID.

14.50.2 Member Function Documentation

14.50.2.1 virtual [AAX_Result AAX_IACFEffector::AddComponent](#) ([IACFUnknown](#) * *inComponentDescriptor*) [pure virtual]

Add a component to an instance of a component descriptor.

Unlike with [AAX_ICollection::AddEffect\(\)](#), the [AAX_IEffectDescriptor](#) does not take ownership of the [AAX_IComponentDescriptor](#) that is passed to it in this method. The host copies out the contents of this descriptor, and thus the plug-in may re-use the same descriptor object when creating additional similar components.

Parameters

<i>in</i>	<i>inComponentDescriptor</i>	
-----------	------------------------------	--

14.50.2.2 virtual [AAX_Result AAX_IACFEffector::AddName](#) (const char * *inPlugInName*) [pure virtual]

Add a name to the Effect.

May be called multiple times to add abbreviated Effect names.

Note

Every Effect must include at least one name variant with 31 or fewer characters, plus a null terminating character

Parameters

in	<i>inPlugInName</i>	The name assigned to the plug-in.
----	---------------------	-----------------------------------

14.50.2.3 virtual AAX_Result AAX_IACFEffectDescriptor::AddCategory (uint32_t *inCategory*) [pure virtual]

Add a category to your plug-in. See [AAX_EPlugInCategory](#).

Parameters

in	<i>inCategory</i>	One of the categories for the plug-in.
----	-------------------	--

14.50.2.4 virtual AAX_Result AAX_IACFEffectDescriptor::AddCategoryBypassParameter (uint32_t *inCategory*, AAX_CParamID *inParamID*) [pure virtual]

Add a category to your plug-in. See [AAX_EPlugInCategory](#).

Parameters

in	<i>inCategory</i>	One of the categories for the plug-in.
in	<i>inParamID</i>	The parameter ID of the parameter used to bypass the category section of the plug-in.

14.50.2.5 virtual AAX_Result AAX_IACFEffectDescriptor::AddProcPtr (void * *inProcPtr*, AAX_CProcPtrID *inProcID*) [pure virtual]

Add a process pointer.

Parameters

in	<i>inProcPtr</i>	A process pointer.
in	<i>inProcID</i>	A process ID.

14.50.2.6 virtual AAX_Result AAX_IACFEffectDescriptor::SetProperties (IACFUnknown * *inProperties*) [pure virtual]

Set the properties of a new property map.

Parameters

in	<i>inProperties</i>	Description
----	---------------------	-------------

14.50.2.7 virtual AAX_Result AAX_IACFEffectDescriptor::AddResourceInfo (AAX_EResourceType *in ResourceType*, const char * *inInfo*) [pure virtual]

Set resource file info.

Parameters

in	<i>inResourceType</i>	See AAX_EResourceType.
in	<i>inInfo</i>	Definition varies on the resource type.

```
14.50.2.8 virtual AAX_Result AAX_IACFEffectorDescriptor::AddMeterDescription ( AAX_CTypeID inMeterID, const char *  
                                inMeterName, IACFUnknown * inProperties ) [pure virtual]
```

Add name and property map to meter with given ID.

Parameters

in	<i>inMeterID</i>	The ID of the meter being described.
in	<i>inMeterName</i>	The name of the meter.
in	<i>inProperties</i>	The property map containing meter related data such as meter type, orientation, etc.

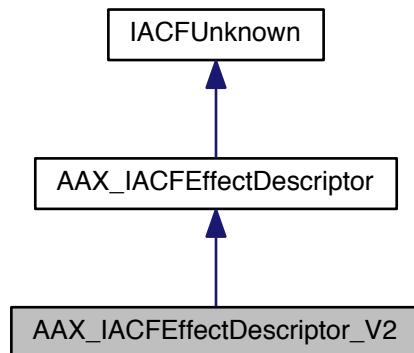
The documentation for this class was generated from the following file:

- [AAX_IACFEffectorDescriptor.h](#)

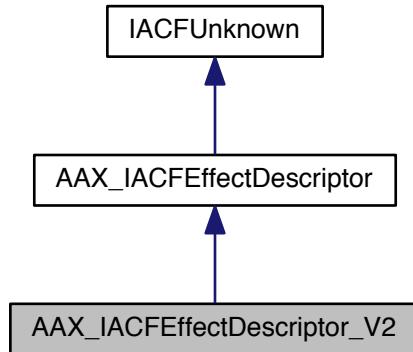
14.51 AAX_IACFEffectorDescriptor_V2 Class Reference

```
#include <AAX_IACFEffectorDescriptor.h>
```

Inheritance diagram for AAX_IACFEffectorDescriptor_V2:



Collaboration diagram for AAX_IACFEffectorDescriptor_V2:



14.51.1 Description

Versioned interface for an [AAX_IEffectDescriptor](#).

Public Member Functions

- virtual [AAX_Result AAX_IACFEffectorDescriptor_V2::AddControlMIDIINode](#) ([AAX_CTypeID](#) *inNodeID*, [AAX_EMIDIINodeType](#) *inNodeType*, const char *innodeName*[], uint32_t *inChannelMask*)=0
Add a control MIDI node to the plug-in data model.

14.51.2 Member Function Documentation

14.51.2.1 virtual [AAX_Result AAX_IACFEffectorDescriptor_V2::AddControlMIDIINode](#) ([AAX_CTypeID](#) *inNodeID*, [AAX_EMIDIINodeType](#) *inNodeType*, const char *innodeName*[], uint32_t *inChannelMask*) [pure virtual]

Add a control MIDI node to the plug-in data model.

- This MIDI node may receive note data as well as control data.
- To send MIDI data to the plug-in's algorithm, use [AAX_IComponentDescriptor::AddMIDIINode\(\)](#).

See also

[AAX_IACFEffectorParameters_V2::UpdateControlMIDIINodes\(\)](#)

Parameters

in	<i>inNodeID</i>	The ID for the new control MIDI node.
----	-----------------	---------------------------------------

in	<i>inNodeType</i>	The type of the node.
in	<i>innodeName</i>	The name of the node.
in	<i>inChannelMask</i>	The bit mask for required nodes channels (up to 16) or required global events for global node.

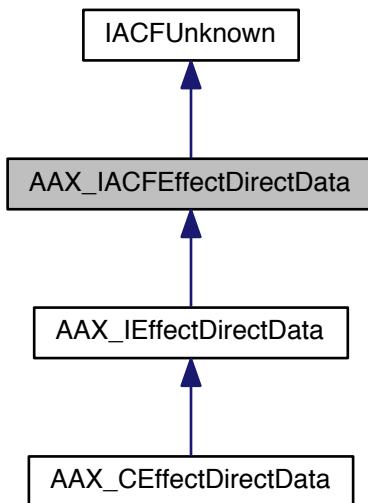
The documentation for this class was generated from the following file:

- [AAX_IACFEfectDescriptor.h](#)

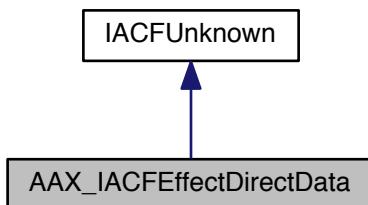
14.52 AAX_IACFEfectDirectData Class Reference

```
#include <AAX_IACFEfectDirectData.h>
```

Inheritance diagram for AAX_IACFEfectDirectData:



Collaboration diagram for AAX_IACFEfectDirectData:



14.52.1 Description

Optional interface for direct access to a plug-in's alg memory.

Direct data access allows a plug-in to directly manipulate the data in its algorithm's private data blocks. The callback methods in this interface provide a safe context from which this kind of access may be attempted.

Public Member Functions

Initialization and uninitialization

- virtual [AAX_Result Initialize \(IACFUnknown *iController\)=0](#)
Main initialization.
- virtual [AAX_Result Uninitialize \(\)=0](#)
Main uninitialization.

Safe data update callbacks

These callbacks provide a safe context from which to directly access the algorithm's private data blocks. Each callback is called regularly with roughly the schedule of its corresponding [AAX_IEffectParameters](#) counterpart.

Note

Do not attempt to directly access the algorithm's data from outside these callbacks. Instead, use the packet system to route data to the algorithm using the AAX host's buffered data transfer facilities.

- virtual [AAX_Result TimerWakeup \(IACFUnknown *iDataAccessInterface\)=0](#)
Periodic wakeup callback for idle-time operations.

14.52.2 Member Function Documentation

14.52.2.1 virtual [AAX_Result AAX_IACEffectDirectData::Initialize \(IACFUnknown * iController \) \[pure virtual\]](#)

Main initialization.

Called when the interface is created

Parameters

in	<i>iController</i>	A versioned reference that resolves to an AAX_IController interface
----	--------------------	---

Implemented in [AAX_CEffectDirectData](#).

14.52.2.2 virtual [AAX_Result AAX_IACEffectDirectData::Uninitialize \(\) \[pure virtual\]](#)

Main uninitialization.

Called when the interface is destroyed.

Implemented in [AAX_CEffectDirectData](#).

14.52.2.3 virtual [AAX_Result AAX_IACEffectDirectData::TimerWakeup \(IACFUnknown * iDataAccessInterface \) \[pure virtual\]](#)

Periodic wakeup callback for idle-time operations.

Direct alg data updates must be triggered from this method.

This method is called from the host using a non-main thread. In general, it should be driven at approximately one call per 30 ms. However, the wakeup is not guaranteed to be called at any regular interval - for example, it could

be held off by a high real-time processing load - and there is no host contract regarding maximum latency between wakeup calls.

This wakeup thread runs continuously and cannot be armed/disarmed or by the plug-in.

Parameters

in	<i>iDataAccess</i> ↪ <i>Interface</i>	Reference to the direct access interface.
----	--	---

Note

It is not safe to save this address or call the methods in this interface from other threads.

Implemented in [AAX_CEffectDirectData](#).

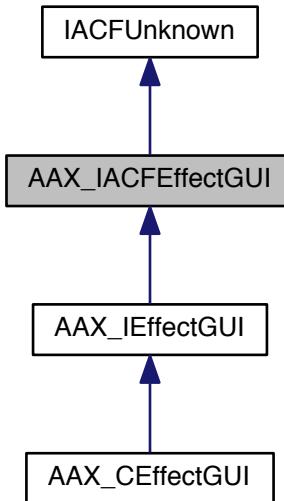
The documentation for this class was generated from the following file:

- [AAX_IACFEfectDirectData.h](#)

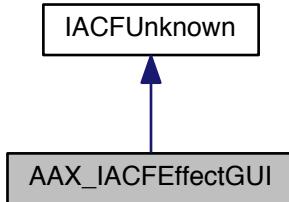
14.53 AAX_IACFEfectGUI Class Reference

```
#include <AAX_IACFEfectGUI.h>
```

Inheritance diagram for AAX_IACFEfectGUI:



Collaboration diagram for AAX_IACFEffetGUI:



14.53.1 Description

The interface for a AAX Plug-in's GUI (graphical user interface).

This is the interface for an instance of a plug-in's GUI that gets exposed to the host application. The AAX host interacts with your plug-in's GUI via this interface. See [GUI interface](#).

The plug-in's implementation of this interface is responsible for managing the plug-in's window and graphics objects and for defining the interactions between GUI views and the plug-in's data model.

At [initialization](#), the host provides this interface with a reference to [AAX_IController](#). The GUI may use this controller to retrieve a pointer to the plug-in's [AAX_IEffectParameters](#) interface, allowing the GUI to request changes to the plug-in's data model in response to view events. In addition, the controller provides a means of querying information from the host such as stem format or sample rate

When managing a plug-in's GUI it is important to remember that this is just one of many possible sets of views for the plug-in's parameters. Other views and editors, such as automation lanes or control surfaces, also have the ability to synchronously interact with the plug-in's abstract data model interface. Therefore, the GUI should not take asymmetric control over the data model, act as a secondary data model, or otherwise assume exclusive ownership of the plug-in's state. In general, the data model's abstraction to a pure virtual interface will protect against such aberrations, but this remains an important consideration when managing sophisticated GUI interactions.

You will most likely inherit your implementation of this interface from [AAX_CEffectGUI](#), a default implementation that provides basic GUI functionality and which you can override and customize as needed.

The SDK includes several examples of how the GUI interface may be extended and implemented in order to provide support for third-party frameworks. These examples can be found in the /Extensions/GUI directory in the SDK.

Note

Your implementation of this interface must inherit from [AAX_IEffectGUI](#).

Legacy Porting Notes In the legacy plug-in SDK, these methods were found in CProcess and CEfectProcess.
For additional CProcess methods, see [AAX_IEffectParameters](#).

Public Member Functions

Initialization and uninitialization

- virtual [AAX_Result Initialize \(IACFUnknown *iController\)=0](#)
Main GUI initialization.
- virtual [AAX_Result Uninitialize \(\)=0](#)
Main GUI uninitialization.

AAX host and plug-in event notification

- virtual [AAX_Result NotificationReceived \(AAX_CTypeID inNotificationType, const void *inNotificationData, uint32_t inNotificationDataSize\)=0](#)
Notification Hook.

View accessors

- virtual [AAX_Result SetViewContainer \(IACFUnknown *iViewContainer\)=0](#)
Provides a handle to the main plug-in window.
- virtual [AAX_Result GetViewSize \(AAX_Point *oViewSize\) const =0](#)
Retrieves the size of the plug-in window.

GUI update methods

- virtual [AAX_Result Draw \(AAX_Rect *iDrawRect\)=0](#)
DEPRECATED, Not called from host any longer. Your chosen graphics framework should be directly handling draw events from the OS.
- virtual [AAX_Result TimerWakeup \(\)=0](#)
Periodic wakeup callback for idle-time operations.
- virtual [AAX_Result ParameterUpdated \(AAX_CParamID inParamID\)=0](#)
Notifies the GUI that a parameter value has changed.

Host interface methods

Miscellaneous methods to provide host-specific functionality

- virtual [AAX_Result GetCustomLabel \(AAX_EPlugInStrings iSelector, AAX_IString *oString\) const =0](#)
Called by host application to retrieve a custom plug-in string.
- virtual [AAX_Result SetControlHighlightInfo \(AAX_CParamID iParamID, AAX_CBoolean iIsHighlighted, AAX_EHighlightColor iColor\)=0](#)
Called by host application. Indicates that a control widget should be updated with a highlight color.

14.53.2 Member Function Documentation

14.53.2.1 virtual [AAX_Result AAX_IACEEffectGUI::Initialize \(IACFUnknown * iController \) \[pure virtual\]](#)

Main GUI initialization.

Called when the GUI is created

Parameters

in	<i>iController</i>	A versioned reference that resolves to an AAX_IController interface
----	--------------------	---

Implemented in [AAX_CEffectGUI](#).

14.53.2.2 virtual [AAX_Result AAX_IACEEffectGUI::Uninitialize \(\) \[pure virtual\]](#)

Main GUI uninitialization.

Called when the GUI is destroyed. Frees the GUI.

Implemented in [AAX_CEffectGUI](#).

14.53.2.3 virtual AAX_Result AAX_IACFEfectGUI::NotificationReceived (**AAX_CTypeID** *inNotificationType*, const void * *inNotificationData*, uint32_t *inNotificationDataSize*) [pure virtual]

Notification Hook.

Called from the host to deliver notifications to this object.

Look at the [AAX_ENotificationEvent](#) enumeration to see a description of events you can listen for and the data they come with.

Note

- some notifications are sent only to the plug-in GUI while other notifications are sent only to the plug-in data model. If you are not seeing an expected notification, try checking the other plug-in objects' [NotificationReceived\(\)](#) methods.

Note

- the host may dispatch notifications synchronously or asynchronously, and calls to this method may occur concurrently on multiple threads.

A plug-in may also dispatch custom notifications using [AAX_IController::SendNotification\(\)](#). Custom notifications will be posted back to the plug-in's other objects which support a [NotificationReceived\(\)](#) method (e.g. the data model).

Parameters

in	<i>inNotificationType</i>	Type of notification being received. Notifications from the host are one of AAX_ENotificationEvent
in	<i>inNotificationData</i>	Block of incoming notification data
in	<i>inNotificationDataSize</i>	Size of <i>inNotificationData</i> , in bytes

Implemented in [AAX_CEffectGUI](#).

14.53.2.4 virtual AAX_Result AAX_IACFEfectGUI::SetViewContainer (IACFUnknown * *iViewContainer*) [pure virtual]

Provides a handle to the main plug-in window.

Parameters

in	<i>iViewContainer</i>	An AAX_IViewContainer providing a native handle to the plug-in's window
----	-----------------------	---

Implemented in [AAX_CEffectGUI](#).

14.53.2.5 virtual AAX_Result AAX_IACFEfectGUI::GetViewSize (AAX_Point * *oViewSize*) const [pure virtual]

Retrieves the size of the plug-in window.

See also

[AAX_IViewContainer::SetViewSize\(\)](#)

Parameters

out	<i>oViewSize</i>	The size of the plug-in window as a point (width, height)
-----	------------------	---

Implemented in [AAX_CEffectGUI](#).

14.53.2.6 virtual AAX_Result AAX_IACFEffector::Draw (AAX_Rect * *iDrawRect*) [pure virtual]

DEPRECATED, Not called from host any longer. Your chosen graphics framework should be directly handling draw events from the OS.

Implemented in [AAX_CEffectGUI](#).

14.53.2.7 virtual AAX_Result AAX_IACFEffector::TimerWakeup () [pure virtual]

Periodic wakeup callback for idle-time operations.

GUI animation events such as meter updates should be triggered from this method.

This method is called from the host's main thread. In general, it should be driven at approximately one call per 30 ms. However, the wakeup is not guaranteed to be called at any regular interval - for example, it could be held off by a high real-time processing load - and there is no host contract regarding maximum latency between wakeup calls.

This wakeup runs continuously and cannot be armed/disarmed by the plug-in.

Implemented in [AAX_CEffectGUI](#).

14.53.2.8 virtual AAX_Result AAX_IACFEffector::ParameterUpdated (AAX_CParamID *inParamID*) [pure virtual]

Notifies the GUI that a parameter value has changed.

This method is called by the host whenever a parameter value has been modified

This method may be called on a non-main thread

Implemented in [AAX_CEffectGUI](#).

14.53.2.9 virtual AAX_Result AAX_IACFEffector::GetCustomLabel (AAX_EPlugInStrings *iSelector*, AAX_IString * *oString*) const [pure virtual]

Called by host application to retrieve a custom plug-in string.

If no string is provided then the host's default will be used.

Parameters

in	<i>iSelector</i>	The requested strong. One of AAX_EPlugInStrings
out	<i>oString</i>	The plug-in's custom value for the requested string

Implemented in [AAX_CEffectGUI](#).

14.53.2.10 virtual AAX_Result AAX_IACFEffector::SetControlHighlightInfo (AAX_CParamID *iParameterID*, AAX_CBoolean *iIsHighlighted*, AAX_EHighlightColor *iColor*) [pure virtual]

Called by host application. Indicates that a control widget should be updated with a highlight color.

Todo Document this method

Legacy Porting Notes This method was re-named from `SetControlHighliteInfo()`, its name in the legacy plug-in SDK.

Parameters

in	<i>iParameterID</i>	ID of parameter whose widget(s) must be highlighted
in	<i>iIsHighlighted</i>	True if turning highlight on, false if turning it off
in	<i>iColor</i>	Desired highlight color. One of AAX_EHighlightColor

Implemented in [AAX_CEffectGUI](#).

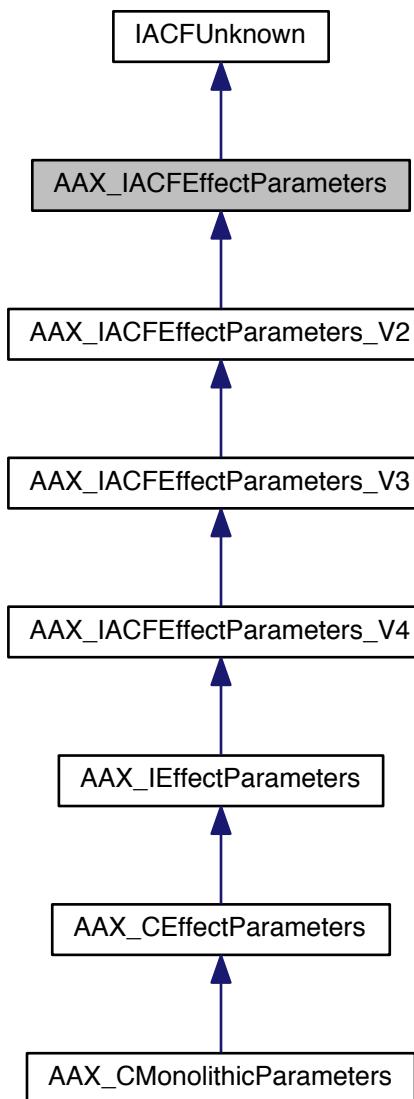
The documentation for this class was generated from the following file:

- [AAX_IACEffectGUI.h](#)

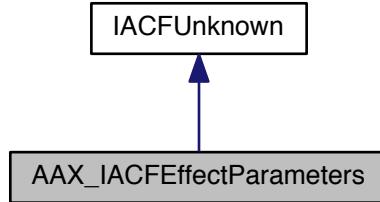
14.54 AAX_IACEffectParameters Class Reference

```
#include <AAX_IACEffectParameters.h>
```

Inheritance diagram for AAX_IACFEfectParameters:



Collaboration diagram for AAX_IACFEffectorParameters:



14.54.1 Description

The interface for an AAX Plug-in's data model.

This is the interface for an instance of a plug-in's data model that gets exposed to the host application. The AAX host interacts with your plug-in's data model via this interface, which includes methods that store and update of your plug-in's internal data. See [Data model interface](#).

Note

Your implementation of this interface must inherit from [AAX_IEffectParameters](#).

Todo Add documentation for expected error state return values

Public Member Functions

Initialization and uninitialization

- virtual [AAX_Result Initialize \(IACFUnknown *iController\)=0](#)
Main data model initialization. Called when plug-in instance is first instantiated.
- virtual [AAX_Result Uninitialize \(\)=0](#)
Main data model uninitialization.

AAX host and plug-in event notification

- virtual [AAX_Result NotificationReceived \(AAX_CTypeID inNotificationType, const void *inNotificationData, uint32_t inNotificationDataSize\)=0](#)
Notification Hook.

Parameter information

These methods are used by the AAX host to retrieve information about the plug-in's data model.

For information about adding parameters to the plug-in and otherwise modifying the plug-in's data model, see [AAX_CParameterManager](#). For information about parameters, see [AAX_IParameter](#).

- virtual [AAX_Result GetNumberOfParameters \(int32_t *oNumControls\) const =0](#)
CALL: Retrieves the total number of plug-in parameters.
- virtual [AAX_Result GetMasterBypassParameter \(AAX_IString *oIDString\) const =0](#)
CALL: Retrieves the ID of the plug-in's Master Bypass parameter.

- virtual `AAX_Result GetParameterIsAutomatable (AAX_CParamID iParameterID, AAX_CBoolean *oAutomatable) const =0`
CALL: Retrieves information about a parameter's automatable status.
- virtual `AAX_Result GetParameterNumberOfSteps (AAX_CParamID iParameterID, int32_t *oNumSteps) const =0`
CALL: Retrieves the number of discrete steps for a parameter.
- virtual `AAX_Result GetParameterName (AAX_CParamID iParameterID, AAX_IString *oName) const =0`
CALL: Retrieves the full name for a parameter.
- virtual `AAX_Result GetParameterNameOfLength (AAX_CParamID iParameterID, AAX_IString *oName, int32_t iNameLength) const =0`
CALL: Retrieves an abbreviated name for a parameter.
- virtual `AAX_Result GetParameterDefaultNormalizedValue (AAX_CParamID iParameterID, double *oValue) const =0`
CALL: Retrieves default value of a parameter.
- virtual `AAX_Result SetParameterDefaultNormalizedValue (AAX_CParamID iParameterID, double iValue)=0`
CALL: Sets the default value of a parameter.
- virtual `AAX_Result GetParameterType (AAX_CParamID iParameterID, AAX_EParameterType *oParameterType) const =0`
CALL: Retrieves the type of a parameter.
- virtual `AAX_Result GetParameterOrientation (AAX_CParamID iParameterID, AAX_EParameterOrientation *oParameterOrientation) const =0`
CALL: Retrieves the orientation that should be applied to a parameter's controls.
- virtual `AAX_Result GetParameter (AAX_CParamID iParameterID, AAX_IPParameter **oParameter)=0`
CALL: Retrieves an arbitrary setting within a parameter.
- virtual `AAX_Result GetParameterIndex (AAX_CParamID iParameterID, int32_t *oControlIndex) const =0`
CALL: Retrieves the index of a parameter.
- virtual `AAX_Result GetParameterIDFromIndex (int32_t iControlIndex, AAX_IString *oParameterIDString) const =0`
CALL: Retrieves the ID of a parameter.
- virtual `AAX_Result GetParameterValueInfo (AAX_CParamID iParameterID, int32_t iSelector, int32_t *oValue) const =0`
CALL: Retrieves a property of a parameter.

Parameter setters and getters

These methods are used by the AAX host and by the plug-in's UI to retrieve and modify the values of the plug-in's parameters.

Note

The parameter setters in this section may generate asynchronous requests.

- virtual `AAX_Result GetParameterValueFromString (AAX_CParamID iParameterID, double *oValue, const AAX_IString &iValueString) const =0`
CALL: Converts a value string to a value.
- virtual `AAX_Result GetParameterStringFromValue (AAX_CParamID iParameterID, double iValue, AAX_IString *oValueString, int32_t iMaxLength) const =0`
CALL: Converts a normalized parameter value into a string representing its corresponding real value.
- virtual `AAX_Result GetParameterValueString (AAX_CParamID iParameterID, AAX_IString *oValueString, int32_t iMaxLength) const =0`
CALL: Retrieves the value string associated with a parameter's current value.
- virtual `AAX_Result GetParameterNormalizedValue (AAX_CParamID iParameterID, double *oValuePtr) const =0`
CALL: Retrieves a parameter's current value.
- virtual `AAX_Result SetParameterNormalizedValue (AAX_CParamID iParameterID, double iValue)=0`
CALL: Sets the specified parameter to a new value.
- virtual `AAX_Result SetParameterNormalizedRelative (AAX_CParamID iParameterID, double iValue)=0`
CALL: Sets the specified parameter to a new value relative to its current value.

Automated parameter helpers

These methods are used to lock and unlock automation system 'resources' when updating automatable parameters.

Note

You should never need to override these methods to extend their behavior beyond what is provided in [AAX_CEffectParameters](#) and [AAX_IParameter](#)

- virtual [AAX_Result TouchParameter \(AAX_CParamID iParameterID\)=0](#)
"Touches" (locks) a parameter in the automation system to a particular control in preparation for updates
- virtual [AAX_Result ReleaseParameter \(AAX_CParamID iParameterID\)=0](#)
Releases a parameter from a "touched" state.
- virtual [AAX_Result UpdateParameterTouch \(AAX_CParamID iParameterID, AAX_CBoolean iTouch← State\)=0](#)
Sets a "touched" state on a parameter.

Asynchronous parameter update methods

These methods are called by the AAX host when parameter values have been updated. They are called by the host and can be triggered by other plug-in modules via calls to [AAX_IParameter](#)'s `SetValue` methods, e.g. `SetValueWithFloat()`

These methods are responsible for updating parameter values.

Do not call these methods directly! To ensure proper synchronization and to avoid problematic dependency chains, other methods (e.g. `SetParameterNormalizedValue()`) and components (e.g. [AAX_IEffectGUI](#)) should always call a `SetValue` method on [AAX_IParameter](#) to update parameter values. The `SetValue` method will properly manage automation locks and other system resources.

- virtual [AAX_Result UpdateParameterNormalizedValue \(AAX_CParamID iParameterID, double iValue, AAX_EUpdateSource iSource\)=0](#)
Updates a single parameter's state to its current value.
- virtual [AAX_Result UpdateParameterNormalizedRelative \(AAX_CParamID iParameterID, double i← Value\)=0](#)
Updates a single parameter's state to its current value, as a difference with the parameter's previous value.
- virtual [AAX_Result GenerateCoefficients \(\)=0](#)
Generates and dispatches new coefficient packets.

State reset handlers

- virtual [AAX_Result ResetFieldData \(AAX_CFieldIndex inFieldIndex, void *oData, uint32_t inDataSize\)](#)
const =0
Called by the host to reset a private data field in the plug-in's algorithm.

Chunk methods

These methods are used to save and restore collections of plug-in state information, known as chunks. Chunks are used by the host when saving or restoring presets and session settings and when providing "compare" functionality for plug-ins.

The default implementation of these methods in [AAX_CEffectParameters](#) supports a single chunk that includes state information for all of the plug-in's registered parameters. Override all of these methods to add support for additional chunks in your plug-in, for example if your plug-in contains any persistent state that is not encapsulated by its set of registered parameters.

Warning

Remember that plug-in chunk data may be loaded on a different platform from the one where it is saved. All data structures in the chunk must be properly data-aligned for compatibility across all platforms that the plug-in supports. See [AAX_ALIGN_FILE_ALG](#) for notes about common cross-platform pitfalls for data structure alignment.

For reference, see also:

- [AAX_CChunkDataParser](#)
- [AAX_SPlugInChunk](#)

- virtual [AAX_Result GetNumberOfChunks](#) (int32_t *oNumChunks) const =0

Retrieves the number of chunks used by this plug-in.
- virtual [AAX_Result GetChunkIDFromIndex](#) (int32_t iIndex, [AAX_CTypeID](#) *oChunkID) const =0

Retrieves the ID associated with a chunk index.
- virtual [AAX_Result GetChunkSize](#) ([AAX_CTypeID](#) iChunkID, uint32_t *oSize) const =0

Get the size of the data structure that can hold all of a chunk's information.
- virtual [AAX_Result GetChunk](#) ([AAX_CTypeID](#) iChunkID, [AAX_SPlugInChunk](#) *oChunk) const =0

Fills a block of data with chunk information representing the plug-in's current state.
- virtual [AAX_Result SetChunk](#) ([AAX_CTypeID](#) iChunkID, const [AAX_SPlugInChunk](#) *iChunk)=0

Restores a set of plug-in parameters based on chunk information.
- virtual [AAX_Result CompareActiveChunk](#) (const [AAX_SPlugInChunk](#) *iChunkP, [AAX_CBoolean](#) *ols←Equal) const =0

Determine if a chunk represents settings that are equivalent to the plug-in's current state.
- virtual [AAX_Result GetNumberOfChanges](#) (int32_t *oNumChanges) const =0

Retrieves the number of parameter changes made since the plug-in's creation.

Thread methods

- virtual [AAX_Result TimerWakeup](#) ()=0

Periodic wakeup callback for idle-time operations.

Auxiliary UI methods

- virtual [AAX_Result GetCurveData](#) ([AAX_CTypeID](#) iCurveType, const float *iValues, uint32_t iNumValues, float *oValues) const =0

Generate a set of output values based on a set of given input values.

Custom data methods

These functions exist as a proxiable way to move data between different modules (e.g. [AAX_IEffectParameters](#) and [AAX_IEffectGUI](#).) Using these, the GUI can query any data through [GetCustomData\(\)](#) with a plug-in defined typeID, void and size. This has an advantage over just sharing memory in that this function can work as a remote proxy as we enable those sorts of features later in the platform. Likewise, the GUI can also set arbitrary data on the data model by using the [SetCustomData\(\)](#) function with the same idea.*

Note

These are plug-in internal only. They are not called from the host right now, or likely ever.

- virtual [AAX_Result GetCustomData](#) ([AAX_CTypeID](#) iDataBlockID, uint32_t inDataSize, void *oData, uint32_t *oDataWritten) const =0

An optional interface hook for getting custom data from another module.
- virtual [AAX_Result SetCustomData](#) ([AAX_CTypeID](#) iDataBlockID, uint32_t inDataSize, const void *i←Data)=0

An optional interface hook for setting custom data for use by another module.

MIDI methods

- virtual [AAX_Result DoMIDITransfers](#) ()=0

MIDI update callback.

14.54.2 Member Function Documentation

14.54.2.1 virtual **AAX_Result** AAX_IACFEffectorParameters::Initialize (**IACFUnknown** * *iController*) [pure virtual]

Main data model initialization. Called when plug-in instance is first instantiated.

Note

Most plug-ins should override [AAX_CEffectParameters::EffectInit\(\)](#) rather than directly overriding this method

Parameters

in	<i>iController</i>	A versioned reference that resolves to an AAX_IController interface
----	--------------------	---

Implemented in [AAX_CEffectParameters](#).

14.54.2.2 virtual **AAX_Result** AAX_IACFEffectorParameters::Uninitialize () [pure virtual]

Main data model uninitialization.

Todo Docs: When exactly is [AAX_IACFEffectorParameters::Uninitialize\(\)](#) called, and under what conditions?

Implemented in [AAX_CEffectParameters](#).

14.54.2.3 virtual **AAX_Result** AAX_IACFEffectorParameters::NotificationReceived (**AAX_CTypeID** *inNotificationType*, const void * *inNotificationData*, uint32_t *inNotificationDataSize*) [pure virtual]

Notification Hook.

Called from the host to deliver notifications to this object.

Look at the [AAX_ENotificationEvent](#) enumeration to see a description of events you can listen for and the data they come with.

Note

- some notifications are sent only to the plug-in GUI while other notifications are sent only to the plug-in data model. If you are not seeing an expected notification, try checking the other plug-in objects' [NotificationReceived\(\)](#) methods.

Note

- the host may dispatch notifications synchronously or asynchronously, and calls to this method may occur concurrently on multiple threads.

A plug-in may also dispatch custom notifications using [AAX_IController::SendNotification\(\)](#). Custom notifications will be posted back to the plug-in's other objects which support a [NotificationReceived\(\)](#) method (e.g. the GUI).

Parameters

in	<i>inNotificationType</i>	Type of notification being received. Notifications from the host are one of AAX_ENotificationEvent
in	<i>inNotificationData</i>	Block of incoming notification data

in	<i>inNotificationDataSize</i>	Size of <i>inNotificationData</i> , in bytes
----	-------------------------------	--

Implemented in [AAX_CEffectParameters](#).

14.54.2.4 virtual AAX_Result AAX_IACEffectParameters::GetNumberOfParameters (int32_t * *oNumControls*) const [pure virtual]

CALL: Retrieves the total number of plug-in parameters.

Parameters

out	<i>oNumControls</i>	The number of parameters in the plug-in's Parameter Manager
-----	---------------------	---

Implemented in [AAX_CEffectParameters](#).

14.54.2.5 virtual AAX_Result AAX_IACEffectParameters::GetMasterBypassParameter (AAX_IString * *oIDString*) const [pure virtual]

CALL: Retrieves the ID of the plug-in's Master Bypass parameter.

This is required if you want our master bypass functionality in the host to hook up to your bypass parameters.

Parameters

out	<i>oIDString</i>	The ID of the plug-in's Master Bypass control
-----	------------------	---

Implemented in [AAX_CEffectParameters](#).

14.54.2.6 virtual AAX_Result AAX_IACEffectParameters::GetParameterIsAutomatable (AAX_CParamID *iParameterID*, AAX_CBoolean * *oAutomatable*) const [pure virtual]

CALL: Retrieves information about a parameter's automatable status.

Parameters

in	<i>iParameterID</i>	The ID of the parameter that is being queried
out	<i>oAutomatable</i>	True if the queried parameter is automatable, false if it is not

Implemented in [AAX_CEffectParameters](#).

14.54.2.7 virtual AAX_Result AAX_IACEffectParameters::GetParameterNumberOfSteps (AAX_CParamID *iParameterID*, int32_t * *oNumSteps*) const [pure virtual]

CALL: Retrieves the number of discrete steps for a parameter.

Note

The value returned for *oNumSteps* MUST be greater than zero. All other values will be considered an error by the host.

Parameters

in	<i>iParameterID</i>	The ID of the parameter that is being queried
out	<i>oNumSteps</i>	The number of steps for this parameter

Implemented in [AAX_CEffectParameters](#).

14.54.2.8 virtual AAX_Result AAX_IACEffectParameters::GetParameterName (**AAX_CParamID iParameterID,**
AAX_IString * oName) const [pure virtual]

CALL: Retrieves the full name for a parameter.

Parameters

in	<i>iParameterID</i>	The ID of the parameter that is being queried
out	<i>oName</i>	Reference to an AAX_IString owned by the host. The plug-in must set this string equal to the parameter's full name.

Implemented in [AAX_CEffectParameters](#).

14.54.2.9 virtual AAX_Result AAX_IACEffectParameters::GetParameterNameOfLength (**AAX_CParamID iParameterID,**
AAX_IString * oName, int32_t iNameLength) const [pure virtual]

CALL: Retrieves an abbreviated name for a parameter.

In general, lengths of 3 through 8 and 31 should be specifically addressed.

Host Compatibility Notes In most cases, the AAX host will call [GetParameterName\(\)](#) or [GetParameterNameOfLength\(\)](#) to retrieve parameter names for display. However, when Pro Tools is retrieving a plug-in name for display on a control surface the XML data stored in the plug-in's page tables will be used in preference to values retrieved from these methods.

Parameters

in	<i>iParameterID</i>	The ID of the parameter that is being queried
out	<i>oName</i>	Reference to an AAX_IString owned by the host. The plug-in must set this string equal to an abbreviated name for the parameter, using <i>iNameLength</i> characters or fewer.
in	<i>iNameLength</i>	The maximum number of characters in <i>oName</i>

Implemented in [AAX_CEffectParameters](#).

14.54.2.10 virtual AAX_Result AAX_IACEffectParameters::GetParameterDefaultNormalizedValue (**AAX_CParamID iParameterID,**
double * oValue) const [pure virtual]

CALL: Retrieves default value of a parameter.

Parameters

in	<i>iParameterID</i>	The ID of the parameter that is being queried
out	<i>oValue</i>	The parameter's default value

Implemented in [AAX_CEffectParameters](#).

14.54.2.11 virtual AAX_Result AAX_IACEffectParameters::SetParameterDefaultNormalizedValue (**AAX_CParamID iParameterID,**
double iValue) [pure virtual]

CALL: Sets the default value of a parameter.

Parameters

in	<i>iParameterID</i>	The ID of the parameter that is being updated
out	<i>iValue</i>	The parameter's new default value

Todo THIS IS NOT CALLED FROM HOST. USEFUL FOR INTERNAL USE ONLY?

Implemented in [AAX_CEffectParameters](#).

14.54.2.12 virtual AAX_Result AAX_IACFEFFECTPARAMETERS::GetParameterType (AAX_CParamID *iParameterID*,
 AAX_EParameterType * *oParameterType*) const [pure virtual]

CALL: Retrieves the type of a parameter.

Todo The concept of parameter type needs more documentation

Parameters

in	<i>iParameterID</i>	The ID of the parameter that is being queried
out	<i>oParameterType</i>	The parameter's type

Implemented in [AAX_CEffectParameters](#).

14.54.2.13 virtual AAX_Result AAX_IACFEFFECTPARAMETERS::GetParameterOrientation (AAX_CParamID *iParameterID*,
 AAX_EParameterOrientation * *oParameterOrientation*) const [pure virtual]

CALL: Retrieves the orientation that should be applied to a parameter's controls.

Todo update this documentation

This method allows you to specify the orientation of knob controls that are managed by the host (e.g. knobs on an attached control surface.)

Here is an example override of this method that reverses the orientation of a control for a parameter:

```
// AAX_IParameter* myBackwardsParameter
if (iParameterID == myBackwardsParameter->Identifier())
{
    *oParameterType =
        AAX_eParameterOrientation_BottomMinTopMax |
        AAX_eParameterOrientation_LeftMinRightMax |
        AAX_eParameterOrientation_RotaryWrapMode |
        AAX_eParameterOrientation_RotaryLeftMinRightMax;
}
```

The orientation options are set according to [AAX_EParameterOrientationBits](#)

Legacy Porting Notes `AAX_IEffectParameters::GetParameterOrientation()` corresponds to the `GetControlOrientation()` method in the legacy RTAS/TDM SDK.

Parameters

in	<i>iParameterID</i>	The ID of the parameter that is being queried
out	<i>oParameterOrientation</i>	The orientation of the parameter

Implemented in [AAX_CEffectParameters](#).

14.54.2.14 virtual AAX_Result AAX_IACEffectParameters::GetParameter (AAX_CParamID *iParameterID*, AAX_IParameter ** *oParameter*) [pure virtual]

CALL: Retrieves an arbitrary setting within a parameter.

This is a convenience function for accessing the richer parameter interface from the plug-in's other modules.

Note

This function must not be called by the host; [AAX_IParameter](#) is not safe for passing across the binary boundary with the host!

Parameters

in	<i>iParameterID</i>	The ID of the parameter that is being queried
out	<i>oParameter</i>	A pointer to the returned parameter

Implemented in [AAX_CEffectParameters](#).

14.54.2.15 virtual AAX_Result AAX_IACEffectParameters::GetParameterIndex (AAX_CParamID *iParameterID*, int32_t * *oControlIndex*) const [pure virtual]

CALL: Retrieves the index of a parameter.

Although parameters are normally referenced by their AAX_CParamID, each parameter is also associated with a unique numeric index.

Parameters

in	<i>iParameterID</i>	The ID of the parameter that is being queried
out	<i>oControlIndex</i>	The parameter's numeric index

Implemented in [AAX_CEffectParameters](#).

14.54.2.16 virtual AAX_Result AAX_IACEffectParameters::GetParameterIDFromIndex (int32_t *iControlIndex*, AAX_IString * *oParameterIDString*) const [pure virtual]

CALL: Retrieves the ID of a parameter.

This method can be used to convert a parameter's unique numeric index to its AAX_CParamID

Parameters

in	<i>iControlIndex</i>	The numeric index of the parameter that is being queried
out	<i>oParameterIDString</i>	Reference to an AAX_IString owned by the host. The plug-in must set this string equal to the parameter's ID.

Implemented in [AAX_CEffectParameters](#).

14.54.2.17 virtual AAX_Result AAX_IACEffectParameters::GetParameterValueInfo (AAX_CParamID *iParameterID*, int32_t *iSelector*, int32_t * *oValue*) const [pure virtual]

CALL: Retrieves a property of a parameter.

This is a general purpose query that is specialized based on the value of *iSelector*. The currently supported selector values are described by [AAX_EParameterValueInfoSelector](#). The meaning of *oValue* is dependent upon *iSelector*.

Parameters

in	<i>iParameterID</i>	The ID of the parameter that is being queried
in	<i>iSelector</i>	The selector of the parameter value to retrieve. See AAX_EParameterValueInfoSelector
out	<i>oValue</i>	The value of the specified parameter

Implemented in [AAX_CEffectParameters](#).

14.54.2.18 virtual AAX_Result AAX_IACFEFFECTPARAMETERS::GetParameterValueFromString (AAX_CParamID *iParameterID*, double * *oValue*, const AAX_IString & *iValueString*) const [pure virtual]

CALL: Converts a value string to a value.

This method uses the queried parameter's display delegate and taper to convert a `char*` string into its corresponding value. The formatting of *valueString* must be supported by the parameter's display delegate in order for this call to succeed.

Legacy Porting Notes This method corresponds to `CProcess::MapControlStringToVal()` in the RTAS/TDM SDK

Parameters

in	<i>iParameterID</i>	The ID of the parameter that is being queried
out	<i>oValue</i>	The value associated with <i>valueString</i>
in	<i>iValueString</i>	The formatted value string that will be converted into a value

Implemented in [AAX_CEffectParameters](#).

14.54.2.19 virtual AAX_Result AAX_IACFEFFECTPARAMETERS::GetParameterStringFromValue (AAX_CParamID *iParameterID*, double *iValue*, AAX_IString * *oValueString*, int32_t *iMaxLength*) const [pure virtual]

CALL: Converts a normalized parameter value into a string representing its corresponding real value.

This method uses the queried parameter's display delegate and taper to convert a normalized value into the corresponding `char*` value string for its real value.

Legacy Porting Notes This method corresponds to `CProcess::MapControlValToString()` in the RTAS/TDM SDK

Parameters

in	<i>iParameterID</i>	The ID of the parameter that is being queried
in	<i>iValue</i>	The normalized value that will be converted to a formatted <i>valueString</i>
out	<i>oValueString</i>	The formatted value string associated with <i>value</i>
in	<i>iMaxLength</i>	The maximum length of <i>valueString</i>

Implemented in [AAX_CEffectParameters](#).

14.54.2.20 virtual AAX_Result AAX_IACFEFFECTPARAMETERS::GetParameterValueString (AAX_CParamID *iParameterID*, AAX_IString * *oValueString*, int32_t *iMaxLength*) const [pure virtual]

CALL: Retrieves the value string associated with a parameter's current value.

This method uses the queried parameter's display delegate and taper to convert its current value into a corresponding `char*` value string.

Parameters

in	<i>iParameterID</i>	The ID of the parameter that is being queried
out	<i>oValueString</i>	The formatted value string associated with the parameter's current value
in	<i>iMaxLength</i>	The maximum length of valueString

Implemented in [AAX_CEffectParameters](#).

14.54.2.21 virtual AAX_Result AAX_IACEffectParameters::GetParameterNormalizedValue (AAX_CParamID *iParameterID*, double * *oValuePtr*) const [pure virtual]

CALL: Retrieves a parameter's current value.

Parameters

in	<i>iParameterID</i>	The ID of the parameter that is being queried
out	<i>oValuePtr</i>	The parameter's current value

Implemented in [AAX_CEffectParameters](#).

14.54.2.22 virtual AAX_Result AAX_IACEffectParameters::SetParameterNormalizedValue (AAX_CParamID *iParameterID*, double *iValue*) [pure virtual]

CALL: Sets the specified parameter to a new value.

[SetParameterNormalizedValue\(\)](#) is responsible for initiating any process that is required in order to update all of the parameter's controls (e.g. in the plug-in's GUI, on control surfaces, in automation lanes, etc.) In most cases, the parameter manager will handle this initiation step.

Parameters

in	<i>iParameterID</i>	The ID of the parameter that is being set
in	<i>iValue</i>	The value to which the parameter should be set

Implemented in [AAX_CEffectParameters](#).

14.54.2.23 virtual AAX_Result AAX_IACEffectParameters::SetParameterNormalizedRelative (AAX_CParamID *iParameterID*, double *iValue*) [pure virtual]

CALL: Sets the specified parameter to a new value relative to its current value.

This method is used in cases when a relative control value is more convenient, for example when updating a GUI control using a mouse wheel or the arrow keys. Note that the host may apply the parameter's step size prior to calling [SetParameterNormalizedRelative\(\)](#) in order to determine the correct value for aValue.

[SetParameterNormalizedRelative\(\)](#) can be used to incorporate "wrapping" behavior in a parameter's controls, if desired. If this behavior is not desired, then this method must properly account for overflow of the parameter's normalized value.

[SetParameterNormalizedRelative\(\)](#) is responsible for initiating any process that is required in order to update all of the parameter's controls (e.g. in the plug-in's GUI, on control surfaces, in automation lanes, etc.) In most cases, the parameter manager will handle this initiation step.

See also [UpdateParameterNormalizedRelative\(\)](#).

Todo REMOVE THIS METHOD (?)

Parameters

in	<i>iParameterID</i>	The ID of the parameter that is being queried
in	<i>iValue</i>	The change in value that should be applied to the parameter

Todo NOT CURRENTLY CALLED FROM THE HOST. USEFUL FOR INTERNAL USE ONLY?

Implemented in [AAX_CEffectParameters](#).

14.54.2.24 virtual AAX_Result AAX_IACFEffectParameters::TouchParameter (AAX_CParamID *iParameterID*) [pure virtual]

"Touches" (locks) a parameter in the automation system to a particular control in preparation for updates

This method is called by the Parameter Manager to prime a parameter for receiving new automation data. When an automatable parameter is touched by a control, it will reject input from other controls until it is released.

Note

You should never need to override this method when using [AAX_CEffectParameters](#).

Parameters

in	<i>iParameterID</i>	The parameter that is being touched
----	---------------------	-------------------------------------

Implemented in [AAX_CEffectParameters](#).

14.54.2.25 virtual AAX_Result AAX_IACFEffectParameters::ReleaseParameter (AAX_CParamID *iParameterID*) [pure virtual]

Releases a parameter from a "touched" state.

This method is called by the Parameter Manager to release a parameter so that any control may send updates to the parameter.

Note

You should never need to override this method when using [AAX_CEffectParameters](#).

Parameters

in	<i>iParameterID</i>	The parameter that is being released
----	---------------------	--------------------------------------

Implemented in [AAX_CEffectParameters](#).

14.54.2.26 virtual AAX_Result AAX_IACFEffectParameters::UpdateParameterTouch (AAX_CParamID *iParameterID*, AAX_CBoolean *iTouchState*) [pure virtual]

Sets a "touched" state on a parameter.

Note

This method should be overridden when dealing with linked parameters. Do NOT use this method to keep track of touch states. Use the [automation delegate](#) for that.

Parameters

in	<i>iParameterID</i>	The parameter that is changing touch states.
in	<i>iTouchState</i>	The touch state of the parameter.

Implemented in [AAX_CEffectParameters](#).

14.54.2.27 virtual AAX_Result AAX_IACEffectParameters::UpdateParameterNormalizedValue (AAX_CParamID *iParameterID*, double *iValue*, AAX_EUpdateSource *iSource*) [pure virtual]

Updates a single parameter's state to its current value.

Note

Do *not* call this method from the plug-in. This method should be called by the host only. To set parameter values from within the plug-in, use the [AAX_IParameter](#) interface.

Todo FLAGGED FOR CONSIDERATION OF REVISION

Parameters

in	<i>iParameterID</i>	The ID of the parameter that is being updated
in	<i>iValue</i>	The parameter's current value, to which its internal state must be updated
in	<i>iSource</i>	The source of the update

Implemented in [AAX_CMonolithicParameters](#), and [AAX_CEffectParameters](#).

14.54.2.28 virtual AAX_Result AAX_IACEffectParameters::UpdateParameterNormalizedRelative (AAX_CParamID *iParameterID*, double *iValue*) [pure virtual]

Updates a single parameter's state to its current value, as a difference with the parameter's previous value.

Deprecated This is not called from the host. It *may* still be useful for internal calls within the plug-in, though it should only ever be used to update non-automatable parameters. Automatable parameters should always be updated through the [AAX_IParameter](#) interface, which will ensure proper coordination with other automation clients.

[UpdateParameterNormalizedRelative\(\)](#) can be used to incorporate "wraparound" behavior in a parameter's controls, if desired. If this behavior is not desired, then this method must properly account for overflow of the parameter's normalized value.

See also

[SetParameterNormalizedRelative\(\)](#)

Parameters

in	<i>iParameterID</i>	The ID of the parameter that is being updated
in	<i>iValue</i>	The difference between the parameter's current value and its previous value (normalized). The parameter's state must be updated to reflect this difference.

Implemented in [AAX_CEffectParameters](#).

14.54.2.29 virtual AAX_Result AAX_IACEffectParameters::GenerateCoefficients () [pure virtual]

Generates and dispatches new coefficient packets.

This method is responsible for updating the coefficient packets associated with all parameters whose states have changed since the last call to [GenerateCoefficients\(\)](#). The host may call this method once for every parameter

update, or it may "batch" parameter updates such that changes for several parameters are all handled by a single call to [GenerateCoefficients\(\)](#).

For more information on tracking parameters' statuses using the [AA_X_CPacketDispatcher](#), helper class, see [AA_X_CPacketDispatcher::SetDirty\(\)](#).

Note

Do *not* call this method from the plug-in. This method should be called by the host only. To set parameter values from within the plug-in, use the [AA_X_IParameter](#) interface.

Implemented in [AA_X_CMonolithicParameters](#), and [AA_X_CEffectParameters](#).

14.54.2.30 virtual **AAX_Result** **AAX_IACFEffectParameters::ResetFieldData** (**AAX_CFieldIndex** *inFieldIndex*, **void *** *oData*, **uint32_t** *inDataSize*) const [pure virtual]

Called by the host to reset a private data field in the plug-in's algorithm.

This method is called sequentially for all private data fields on Effect initialization and during any "reset" event, such as priming for a non-real-time render. This method is called before the algorithm's optional initialization callback, and the initialized private data will be available within that callback via its context block.

See also

[Algorithm initialization](#).

Warning

Any data structures that will be passed between platforms (for example, sent to a TI DSP in an AAX DSP plug-in) must be properly data-aligned for compatibility across both platforms. See [AAX_ALIGN_FILE_ALG](#) for more information about guaranteeing cross-platform compatibility of data structures used for algorithm processing.

Parameters

in	<i>inFieldIndex</i>	The index of the field that is being initialized
out	<i>oData</i>	The pre-allocated block of data that should be initialized
in	<i>inDataSize</i>	The size of the data block, in bytes

Implemented in [AA_X_CMonolithicParameters](#), and [AA_X_CEffectParameters](#).

14.54.2.31 virtual **AAX_Result** **AAX_IACFEffectParameters::GetNumberOfChunks** (**int32_t *** *oNumChunks*) const [pure virtual]

Retrieves the number of chunks used by this plug-in.

Parameters

out	<i>oNumChunks</i>	The number of distinct chunks used by this plug-in
-----	-------------------	--

Implemented in [AA_X_CEffectParameters](#).

14.54.2.32 virtual **AAX_Result** **AAX_IACFEffectParameters::GetChunkIDFromIndex** (**int32_t** *iIndex*, **AAX_CTypeID** * *oChunkID*) const [pure virtual]

Retrieves the ID associated with a chunk index.

Parameters

in	<i>iIndex</i>	Index of the queried chunk
out	<i>oChunkID</i>	ID of the queried chunk

Implemented in [AAX_CEffectParameters](#).

14.54.2.33 virtual AAX_Result AAX_IACEffectParameters::GetChunkSize (AAX_CTypeID *iChunkID*, uint32_t * *oSize*) const [pure virtual]

Get the size of the data structure that can hold all of a chunk's information.

If *chunkID* is one of the plug-in's custom chunks, initialize **size* to the size of the chunk's data in bytes.

This method is invoked every time a chunk is saved, therefore it is possible to have dynamically sized chunks. However, note that each call to [GetChunkSize\(\)](#) will correspond to a following call to [GetChunk\(\)](#). The chunk provided in [GetChunk\(\)](#) *must* have the same size as the *size* provided by [GetChunkSize\(\)](#).

Legacy Porting Notes In [AAX](#), the value provided by [GetChunkSize\(\)](#) should *NOT* include the size of the chunk header. The value should *ONLY* reflect the size of the chunk's data.

Parameters

in	<i>iChunkID</i>	ID of the queried chunk
out	<i>oSize</i>	The chunk's size in bytes

Implemented in [AAX_CEffectParameters](#).

14.54.2.34 virtual AAX_Result AAX_IACEffectParameters::GetChunk (AAX_CTypeID *iChunkID*, AAX_SPlugInChunk * *oChunk*) const [pure virtual]

Fills a block of data with chunk information representing the plug-in's current state.

By calling this method, the host is requesting information about the current state of the plug-in. The following chunk fields should be explicitly populated in this method. Other fields will be populated by the host.

- [AAX_SPlugInChunk::fData](#)
- [AAX_SPlugInChunk::fVersion](#)
- [AAX_SPlugInChunk::fName](#) (Optional)
- [AAX_SPlugInChunk::fSize](#) (Data size only)

Warning

Remember that this chunk data may be loaded on a different platform from the one where it is saved. All data structures in the chunk must be properly data-aligned for compatibility across all platforms that the plug-in supports. See [AAX_ALIGN_FILE_ALG](#) for notes about common cross-platform pitfalls for data structure alignment.

Parameters

in	<i>iChunkID</i>	ID of the chunk that should be provided
out	<i>oChunk</i>	A preallocated block of memory that should be populated with the chunk's data.

Implemented in [AAX_CEffectParameters](#).

14.54.2.35 virtual AAX_Result AAX_IACFEffectorParameters::SetChunk (AAX_CTypeID *iChunkID*, const AAX_SPlugInChunk * *iChunk*) [pure virtual]

Restores a set of plug-in parameters based on chunk information.

By calling this method, the host is attempting to update the plug-in's current state to match the data stored in a chunk. The plug-in should initialize itself to this new state by calling [SetParameterNormalizedValue\(\)](#) for each of the relevant parameters.

Parameters

in	<i>iChunkID</i>	ID of the chunk that is being set
in	<i>iChunk</i>	The chunk

Implemented in [AAX_CEffectParameters](#).

14.54.2.36 virtual AAX_Result AAX_IACFEffectorParameters::CompareActiveChunk (const AAX_SPlugInChunk * *iChunkP*, AAX_CBoolean * *oIsEqual*) const [pure virtual]

Determine if a chunk represents settings that are equivalent to the plug-in's current state.

Host Compatibility Notes In Pro Tools, this method will only be called if a prior call to [GetNumberOfChanges\(\)](#) has indicated that the plug-in's state has changed. If the plug-in's current settings are different from the settings in *aChunkP* then the plug-in's Compare Light will be illuminated in the plug-in header, allowing users to toggle between the plug-in's custom state and its saved state.

Parameters

in	<i>iChunkP</i>	The chunk that is to be tested
out	<i>oIsEqual</i>	True if the chunk represents equivalent settings when compared with the plug-in's current state. False if the chunk represents non-equivalent settings

Implemented in [AAX_CEffectParameters](#).

14.54.2.37 virtual AAX_Result AAX_IACFEffectorParameters::GetNumberOfChanges (int32_t * *oNumChanges*) const [pure virtual]

Retrieves the number of parameter changes made since the plug-in's creation.

This method is polled regularly by the host, and can additionally be triggered by some events such as mouse clicks. When the number provided by this method changes, the host subsequently calls [CompareActiveChunk\(\)](#) to determine if the plug-in's Compare light should be activated.

The value provided by this method should increment with each call to [UpdateParameterNormalizedValue\(\)](#)

Parameters

out	<i>oNumChanges</i>	Must be set to indicate the number of parameter changes that have occurred since plug-in initialization.
-----	--------------------	--

Implemented in [AAX_CEffectParameters](#).

14.54.2.38 virtual AAX_Result AAX_IACFEffectorParameters::TimerWakeup () [pure virtual]

Periodic wakeup callback for idle-time operations.

This method is called from the host using a non-main thread. In general, it should be driven at approximately one call per 30 ms. However, the wakeup is not guaranteed to be called at any regular interval - for example, it could be held off by a high real-time processing load - and there is no host contract regarding maximum latency between wakeup calls.

This wakeup thread runs continuously and cannot be armed/disarmed or by the plug-in.

Implemented in [AAX_CMonolithicParameters](#), and [AAX_CEffectParameters](#).

14.54.2.39 virtual **AAX_Result** **AAX_IACFEFFECTPARAMETERS::GetCustomData** (**AAX_CTypeID** *iDataBlockID*, **uint32_t** *inDataSize*, **void *** *oData*, **uint32_t *** *oDataWritten*) const [pure virtual]

An optional interface hook for getting custom data from another module.

Parameters

in	<i>iDataBlockID</i>	Identifier for the requested block of custom data
in	<i>inDataSize</i>	Size of provided buffer, in bytes
out	<i>oData</i>	Pointer to an allocated buffer. Data will be written here.
out	<i>oDataWritten</i>	The number of bytes actually written

Implemented in [AAX_CEffectParameters](#).

14.54.2.40 virtual **AAX_Result** **AAX_IACFEFFECTPARAMETERS::SetCustomData** (**AAX_CTypeID** *iDataBlockID*, **uint32_t** *inDataSize*, **const void *** *iData*) [pure virtual]

An optional interface hook for setting custom data for use by another module.

Parameters

in	<i>iDataBlockID</i>	Identifier for the provided block of custom data
in	<i>inDataSize</i>	Size of provided buffer, in bytes
in	<i>iData</i>	Pointer to the data buffer

Implemented in [AAX_CEffectParameters](#).

14.54.2.41 virtual **AAX_Result** **AAX_IACFEFFECTPARAMETERS::DoMIDITransfers** () [pure virtual]

MIDI update callback.

Call [AAX_IController::GetNextMIDIpacket\(\)](#) from within this method to retrieve and process MIDI packets directly within the Effect's data model. MIDI data will also be delivered to the Effect algorithm.

This method is called regularly by the host, similarly to [AAX_IEffectParameters::TimerWakeUp\(\)](#)

Implemented in [AAX_CEffectParameters](#).

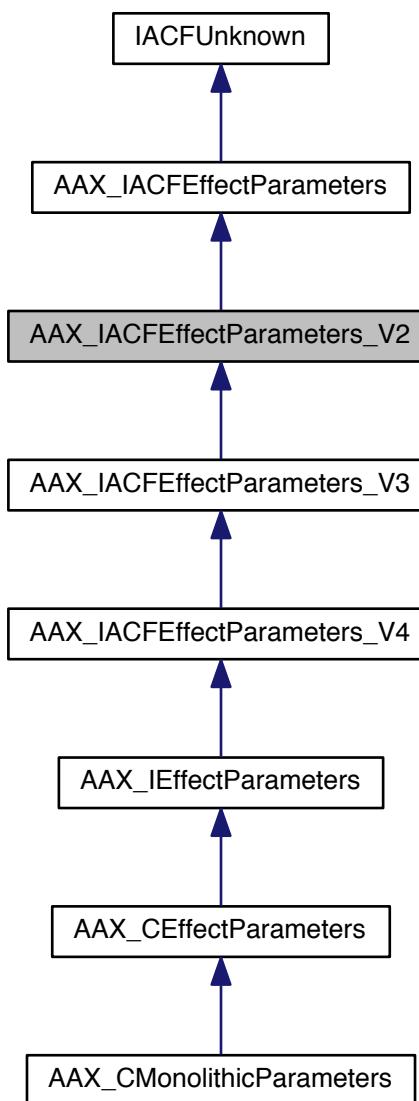
The documentation for this class was generated from the following file:

- [AAX_IACFEFFECTPARAMETERS.h](#)

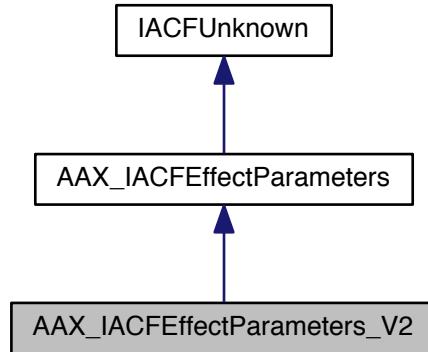
14.55 AAX_IACFEFFECTPARAMETERS_V2 Class Reference

```
#include <AAX_IACFEFFECTPARAMETERS.h>
```

Inheritance diagram for AAX_IACFEfectParameters_V2:



Collaboration diagram for AAX_IACFEFFECTPARAMETERS_V2:



14.55.1 Description

Supplemental interface for an AAX Plug-in's data model.

This is a supplemental interface for an instance of a plug-in's data model. This interface gets exposed to the host application. Host applications that support AAX versioned features may call into these methods. See [Data model interface](#).

Note

Your implementation of this interface must inherit from [AAX_IEffectParameters](#).

Todo Add documentation for expected error state return values

Public Member Functions

Hybrid audio methods

- virtual [AAX_Result RenderAudio_Hybrid \(AAX_SHybridRenderInfo *ioRenderInfo\)=0](#)
Hybrid audio render function.

MIDI methods

- virtual [AAX_Result UpdateMIDINodes \(AAX_CFieldIndex inFieldIndex, AAX_CMidiPacket &iPacket\)=0](#)
MIDI update callback.
- virtual [AAX_Result UpdateControlMIDINodes \(AAX_CTypeID nodeID, AAX_CMidiPacket &iPacket\)=0](#)
MIDI update callback for control MIDI nodes.

14.55.2 Member Function Documentation

14.55.2.1 virtual [AAX_Result AAX_IACFEFFECTPARAMETERS_V2::UpdateMIDINodes \(AAX_CFieldIndex inFieldIndex, AAX_CMidiPacket & iPacket \) \[pure virtual\]](#)

MIDI update callback.

This method is called by the host for each pending MIDI packet for MIDI nodes in algorithm context structure. Overwrite this method in Plug-In's EffectParameter class if you want to receive MIDI data packets directly in the data model. MIDI data will also be delivered to the Effect algorithm.

The host calls this method in Effects that register one or more MIDI nodes using [AAX_IComponentDescriptor::AddMIDINode\(\)](#). Effects that do not require MIDI data to be sent to the plug-in algorithm should override [UpdateControlMIDINodes\(\)](#).

Parameters

in	<i>inFieldIndex</i>	MIDI node field index in algorithm context structure
in	<i>iPacket</i>	The incoming MIDI packet for the node

Implemented in [AAX_CEffectParameters](#).

14.55.2.2 virtual AAX_Result AAX_IACFEFFECTPARAMETERS_V2::UpdateControlMIDINodes (AAX_CTypeID *nodeID*, AAX_CMidiPacket & *iPacket*) [pure virtual]

MIDI update callback for control MIDI nodes.

This method is called by the host for each pending MIDI packet for Control MIDI nodes. Overwrite this method in Plug-In's EffectParameter class if you want to receive MIDI data packets directly in the data model.

The host calls this method in Effects that register one or more Control MIDI nodes using [AAX_IEffectDescriptor::AddControlMIDI](#)Node(). Effects with algorithms that use MIDI data nodes should override [UpdateMIDINodes\(\)](#).

Note

This method will not be called if an Effect includes any MIDI nodes in its algorithm context structure.

Parameters

in	<i>nodeID</i>	Identifier for the MIDI node
in	<i>iPacket</i>	The incoming MIDI packet for the node

Implemented in [AAX_CEffectParameters](#).

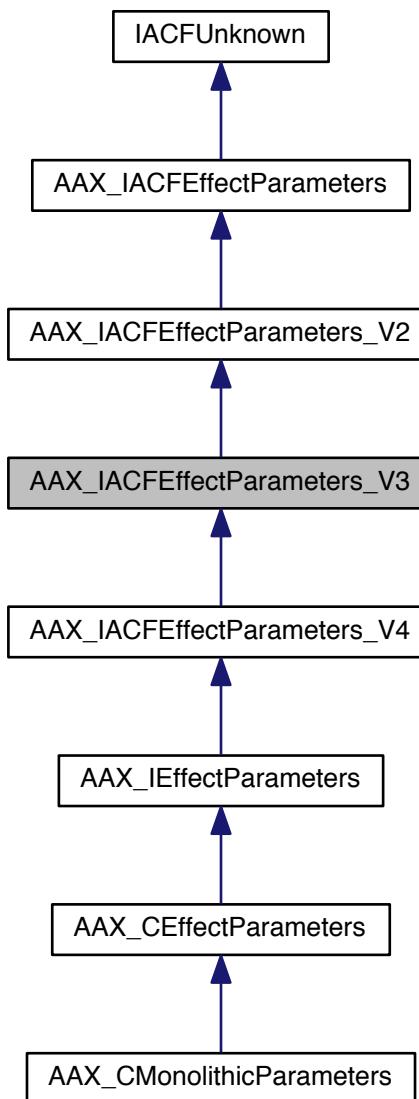
The documentation for this class was generated from the following file:

- [AAX_IACFEFFECTPARAMETERS.h](#)

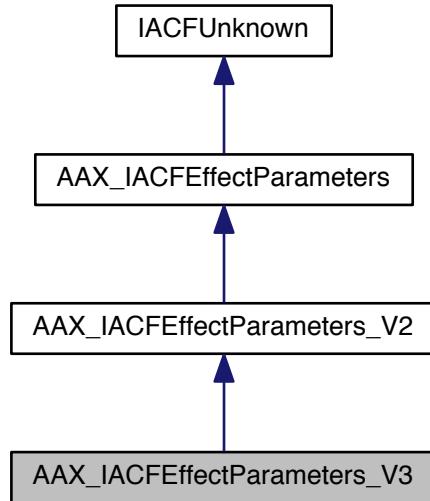
14.56 AAX_IACFEFFECTPARAMETERS_V3 Class Reference

```
#include <AAX_IACFEFFECTPARAMETERS.h>
```

Inheritance diagram for AAX_IACFEfectParameters_V3:



Collaboration diagram for AAX_IACFEffectorParameters_V3:



14.56.1 Description

Supplemental interface for an AAX Plug-in's data model.

This is a supplemental interface for an instance of a plug-in's data model. This interface gets exposed to the host application. Host applications that support AAX versioned features may call into these methods. See [Data model interface](#).

Note

Your implementation of this interface must inherit from [AAX_IEffectParameters](#).

Todo Add documentation for expected error state return values

Public Member Functions

Auxiliary UI methods

- virtual [AAX_Result GetCurveDataMeterIds](#) ([AAX_CTypeID](#) iCurveType, uint32_t *oXMeterId, uint32_t *oYMeterId) const =0
Indicates which meters correspond to the X and Y axes of the EQ or Dynamics graph.
- virtual [AAX_Result GetCurveDataDisplayRange](#) ([AAX_CTypeID](#) iCurveType, float *oXMin, float *oXMax, float *oYMin, float *oYMax) const =0
Determines the range of the graph shown by the plug-in.

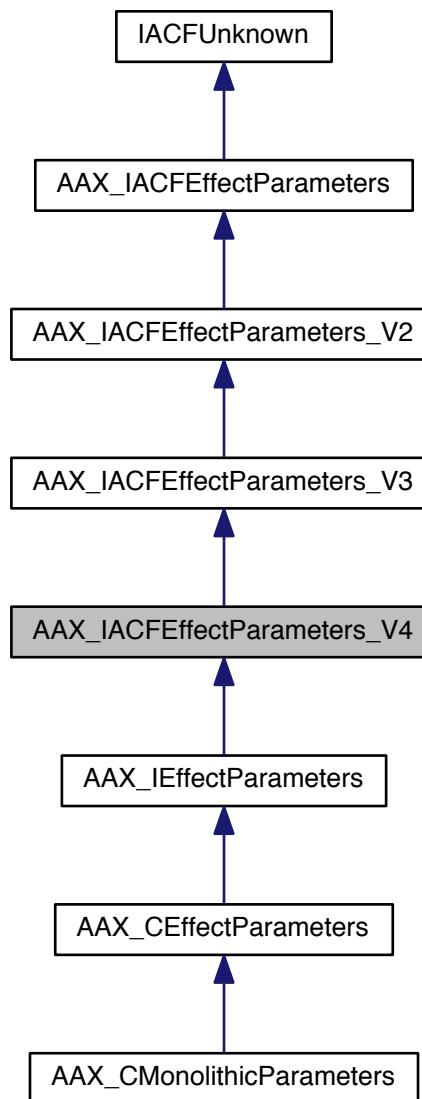
The documentation for this class was generated from the following file:

- [AAX_IACFEffectorParameters.h](#)

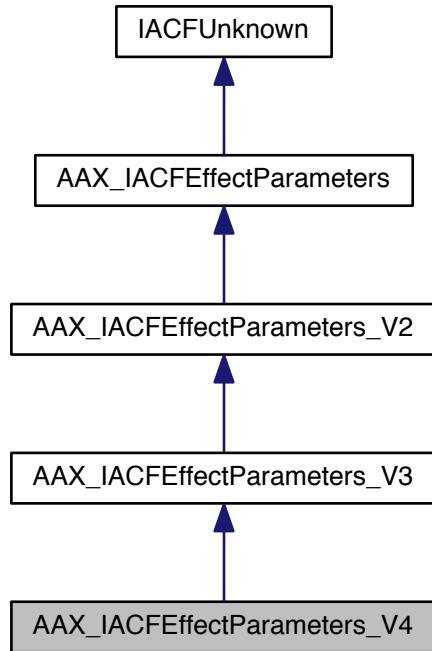
14.57 AAX_IACFEfectParameters_V4 Class Reference

```
#include <AAC_IACFEfectParameters.h>
```

Inheritance diagram for AAX_IACFEfectParameters_V4:



Collaboration diagram for AAX_IACFEffectorParameters_V4:



14.57.1 Description

Supplemental interface for an AAX Plug-in's data model.

This is a supplemental interface for an instance of a plug-in's data model. This interface gets exposed to the host application. Host applications that support AAX versioned features may call into these methods. See [Data model interface](#).

Note

Your implementation of this interface must inherit from [AAX_IEffectParameters](#).

Todo Add documentation for expected error state return values

Public Member Functions

Auxiliary UI methods

- virtual [AAX_Result UpdatePageTable](#) (uint32_t inTableType, int32_t inTablePageSize, [IACFUnknown](#) *iHostUnknown, [IACFUnknown](#) *ioPageTableUnknown) const =0
Allow the plug-in to update its page tables.

14.57.2 Member Function Documentation

```
14.57.2.1 virtual AAX_Result AAX_IACFEffectorParameters_V4::UpdatePageTable( uint32_t inTableType, int32_t
    inTablePageSize, IACFUnknown * iHostUnknown, IACFUnknown * ioPageTableUnknown ) const [pure
    virtual]
```

Allow the plug-in to update its page tables.

Called by the plug-in host, usually in response to a [AAX_eNotificationEvent_ParameterMappingChanged](#) notification sent from the plug-in.

Use this method to change the page table mapping for the plug-in instance or to apply other changes to auxiliary UIs which use the plug-in page tables, such as setting focus to a new page.

See [Page Table Guide](#) for more information about page tables.

Parameters

in	<i>inTableType</i>	Four-char type identifier for the table type (e.g. 'PgTL', 'Av81', etc.)
in	<i>inTablePageSize</i>	Page size for the table
in	<i>iHostUnknown</i>	Unknown interface from the host which may support interfaces providing additional features or information. All interfaces queried from this unknown will be valid only within the scope of this UpdatePageTable() execution and will be relevant for only the current plug-in instance.
in, out	<i>ioPageTableUnknown</i>	Unknown interface which supports AAX_IPageTable . This object represents the page table data which is currently stored by the host for this plug-in instance for the given table type and page size. This data and may be edited within the scope of UpdatePageTable() to change the page table mapping for this plug-in instance.

Returns

This method should return [AAX_ERROR_UNIMPLEMENTED](#) if the plug-in does not implement it or when no change is requested by the plug-in. This allows optimizations to be used in the host when no UI update is required following this call.

See also

[AAX_eNotificationEvent_ParameterMappingChanged](#)

Implemented in [AAX_CEffectParameters](#).

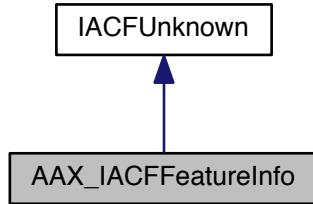
The documentation for this class was generated from the following file:

- [AAX_IACFEffectorParameters.h](#)

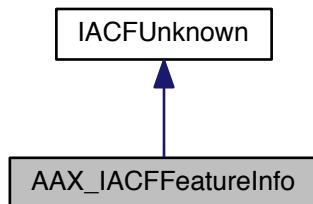
14.58 AAX_IACFFeatureInfo Class Reference

```
#include <AAX_IACFFeatureInfo.h>
```

Inheritance diagram for AAX_IACFFeatureInfo:



Collaboration diagram for AAX_IACFFeatureInfo:



14.58.1 Description

Information about host support for a particular feature

Acquired using [AAX_IACFDescriptionHost::AcquireFeatureProperties\(\)](#)

This interface is shared between multiple features. The specific feature which this object represents is the feature whose ID was used in the call to acquire this interface.

See the feature UID documentation for which properties support additional property map data

IID: [IID_IAAXFeatureInfoV1](#)

Note

Do not [QueryInterface\(\)](#) for [IID_IAAXFeatureInfoV1](#) since this does not indicate which specific feature is being requested. Instead, use [AAX_IDescriptionHost::AcquireFeatureProperties\(\)](#)

Public Member Functions

- virtual [AAX_Result SupportLevel \(AAX_ESupportLevel *oSupportLevel\) const =0](#)
- virtual [AAX_Result AcquireProperties \(IACFUnknown **outProperties\)=0](#)

14.58.2 Member Function Documentation

14.58.2.1 **virtual AAX_Result AAX_IACFFeatureInfo::SupportLevel (AAX_ESupportLevel * oSupportLevel) const [pure virtual]**

Determine the level of support for this feature by the host

Note

The host will not provide an underlying [AAX_IACFFeatureInfo](#) interface for features which it does not recognize at all, resulting in a [AAX_ERROR_NULL_OBJECT](#) error code

See also

[AAX_IFeatureInfo::SupportLevel\(\)](#) Determine the level of support for this feature by the host

Note

The host will not provide an underlying [AAX_IACFFeatureInfo](#) interface for features which it does not recognize at all, resulting in a [AAX_ERROR_NULL_OBJECT](#) error code

14.58.2.2 **virtual AAX_Result AAX_IACFFeatureInfo::AcquireProperties (IACFUnknown ** outProperties) [pure virtual]**

`outProperties` must support [AAX_IACFPropertyMap](#) const methods

See also

[AAX_IFeatureInfo::AcquireProperties\(\)](#)

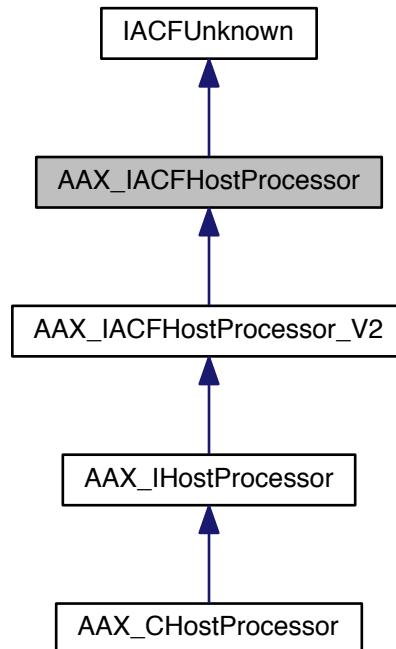
The documentation for this class was generated from the following file:

- [AAX_IACFFeatureInfo.h](#)

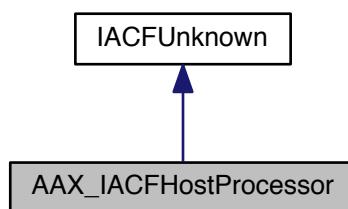
14.59 AAX_IACFHostProcessor Class Reference

```
#include <AAX_IACFHostProcessor.h>
```

Inheritance diagram for AAX_IACFHostProcessor:



Collaboration diagram for AAX_IACFHostProcessor:



14.59.1 Description

Versioned interface for an AAX host processing component.

Note

This interface gets exposed to the host application. See [AAX_CHostProcessor](#) for method documentation.

Legacy Porting Notes This interface provides offline processing features analogous to the legacy AudioSuite plug-in architecture

Public Member Functions

- virtual [AAX_Result Initialize \(IACFUnknown *iController\)=0](#)
Host Processor initialization.
- virtual [AAX_Result Uninitialize \(\)=0](#)
Host Processor teardown.
- virtual [AAX_Result InitOutputBounds \(int64_t iSrcStart, int64_t iSrcEnd, int64_t *oDstStart, int64_t *oDstEnd\)=0](#)
Sets the processing region.
- virtual [AAX_Result SetLocation \(int64_t iSample\)=0](#)
Updates the relative sample location of the current processing frame.
- virtual [AAX_Result RenderAudio \(const float *const inAudioIns\[\], int32_t inAudioInCount, float *const iAudioOuts\[\], int32_t iAudioOutCount, int32_t *ioWindowSize\)=0](#)
Perform the signal processing.
- virtual [AAX_Result PreRender \(int32_t inAudioInCount, int32_t iAudioOutCount, int32_t iWindowSize\)=0](#)
Invoked right before the start of a Preview or Render pass.
- virtual [AAX_Result PostRender \(\)=0](#)
Invoked at the end of a Render pass.
- virtual [AAX_Result AnalyzeAudio \(const float *const inAudioIns\[\], int32_t inAudioInCount, int32_t *ioWindowSize\)=0](#)
Override this method if the plug-in needs to analyze the audio prior to a Render pass.
- virtual [AAX_Result PreAnalyze \(int32_t inAudioInCount, int32_t iWindowSize\)=0](#)
Invoked right before the start of an Analysis pass.
- virtual [AAX_Result PostAnalyze \(\)=0](#)
Invoked at the end of an Analysis pass.

14.59.2 Member Function Documentation

14.59.2.1 virtual [AAX_Result AAX_IACFHostProcessor::Initialize \(IACFUnknown * iController \) \[pure virtual\]](#)

Host Processor initialization.

Parameters

in	<i>iController</i>	A versioned reference that can be resolved to both an AAX_IController interface and an AAX_IHostProcessorDelegate
----	--------------------	---

Implemented in [AAX_CHostProcessor](#).

14.59.2.2 virtual [AAX_Result AAX_IACFHostProcessor::Uninitialize \(\) \[pure virtual\]](#)

Host Processor teardown.

Implemented in [AAX_CHostProcessor](#).

14.59.2.3 virtual [AAX_Result AAX_IACFHostProcessor::InitOutputBounds \(int64_t iSrcStart, int64_t iSrcEnd, int64_t * oDstStart, int64_t * oDstEnd \) \[pure virtual\]](#)

Sets the processing region.

This method allows offline processing plug-ins to vary the length and/or start/end points of the audio processing region.

This method is called in a few different scenarios:

- Before an analyze, process or preview of data begins.

- At the end of every preview loop.
- After the user makes a new data selection on the timeline.

Plug-ins that inherit from [AAX_CHostProcessor](#) should not override this method. Instead, use the following convenience functions:

- To retrieve the length or boundaries of the processing region, use [GetInputRange\(\)](#), [GetSrcStart\(\)](#), etc.
- To change the boundaries of the processing region before processing begins, use [AAX_CHostProcessor::TranslateOutputBounds\(\)](#)

Note

Currently, a host processor may not randomly access samples outside of the boundary defined by `oDstStart` and `oDstEnd`

Legacy Porting Notes DAE no longer makes use of the `mStartBound` and `mEndBounds` member variables that existed in the legacy RTAS/TDM SDK. Use `oDstStart` and `oDstEnd` instead (preferably by overriding [TranslateOutputBounds\(\)](#).)

Parameters

in	<i>iSrcStart</i>	The selection start of the user selected region. This is will always return 0 for a given selection on the timeline.
in	<i>iSrcEnd</i>	The selection end of the user selected region. This will always return the value of the selection length on the timeline.
in	<i>oDstStart</i>	The starting sample location in the output audio region. By default, this is the same as <code>iSrcStart</code> .
in	<i>oDstEnd</i>	The ending sample location in the output audio region. By default, this is the same as <code>iSrcEnd</code> .

Implemented in [AAX_CHostProcessor](#).

14.59.2.4 virtual AAX_Result AAX_IACFHostProcessor::SetLocation(int64_t *iSample*) [pure virtual]

Updates the relative sample location of the current processing frame.

This method is called by the host to update the relative sample location of the current processing frame.

Note

Plug-ins should not override this method; instead, use [AAX_CHostProcessor::GetLocation\(\)](#) to retrieve the current relative sample location.

Parameters

in	<i>iSample</i>	The sample location of the first sample in the current processing frame relative to the beginning of the full processing buffer
----	----------------	---

Implemented in [AAX_CHostProcessor](#).

14.59.2.5 virtual AAX_Result AAX_IACFHostProcessor::RenderAudio(const float *const *inAudioIns*[], int32_t *inAudioInCount*, float *const *iAudioOuts*[], int32_t *iAudioOutCount*, int32_t * *ioWindowSize*) [pure virtual]

Perform the signal processing.

This method is called by the host to invoke the plug-in's signal processing.

Legacy Porting Notes This method is a replacement for the AudioSuite `ProcessAudio` method

Parameters

in	<i>inAudioIns</i>	Input audio buffer
in	<i>inAudioInCount</i>	The number if input channels
in	<i>iAudioOuts</i>	The number of output channels
in	<i>iAudioOutCount</i>	A user defined destination end of the ingested audio
in	<i>ioWindowSize</i>	Window buffer length of the received audio

Implemented in [AAX_CHostProcessor](#).

14.59.2.6 virtual AAX_Result AAX_IACFHostProcessor::PreRender (int32_t *inAudioInCount*, int32_t *iAudioOutCount*, int32_t *ioWindowSize*) [pure virtual]

Invoked right before the start of a Preview or Render pass.

This method is called by the host to allow a plug-in to make any initializations before processing actually begins. Upon a Preview pass, PreRender will also be called at the beginning of every "loop".

See also

[AAX_eProcessingState_StartPass](#), [AAX_eProcessingState_BeginPassGroup](#)

Parameters

in	<i>inAudioInCount</i>	The number if input channels
in	<i>iAudioOutCount</i>	The number of output channels
in	<i>ioWindowSize</i>	Window buffer length of the ingested audio

Implemented in [AAX_CHostProcessor](#).

14.59.2.7 virtual AAX_Result AAX_IACFHostProcessor::PostRender () [pure virtual]

Invoked at the end of a Render pass.

Note

Upon a Preview pass, PostRender will not be called until Preview has stopped.

See also

[AAX_eProcessingState_StopPass](#), [AAX_eProcessingState_EndPassGroup](#)

Implemented in [AAX_CHostProcessor](#).

14.59.2.8 virtual AAX_Result AAX_IACFHostProcessor::AnalyzeAudio (const float *const *inAudioIns*[], int32_t *inAudioInCount*, int32_t * *ioWindowSize*) [pure virtual]

Override this method if the plug-in needs to analyze the audio prior to a Render pass.

Use this after declaring the appropriate properties in Describe. See [AAX_eProperty_RequiresAnalysis](#) and [AAX_eProperty_OptionalAnalysis](#)

To request an analysis pass from within a plug-in, use [AAX_IHostProcessorDelegate::ForceAnalyze\(\)](#)

Legacy Porting Notes Ported from AudioSuite's `AnalyzeAudio(bool isMasterBypassed)` method

Parameters

in	<i>inAudioIns</i>	Input audio buffer
in	<i>inAudioInCount</i>	The number of input channels
in	<i>iWindowSize</i>	Window buffer length of the ingested audio

Implemented in [AAX_CHostProcessor](#).

14.59.2.9 virtual AAX_Result AAX_IACFHostProcessor::PreAnalyze (int32_t *inAudioInCount*, int32_t *iWindowSize*) [pure virtual]

Invoked right before the start of an Analysis pass.

This method is called by the host to allow a plug-in to make any initializations before an Analysis pass actually begins.

See also

[AAX_eProcessingState_StartPass](#), [AAX_eProcessingState_BeginPassGroup](#)

Parameters

in	<i>inAudioInCount</i>	The number if input channels
in	<i>iWindowSize</i>	Window buffer length of the ingested audio

Implemented in [AAX_CHostProcessor](#).

14.59.2.10 virtual AAX_Result AAX_IACFHostProcessor::PostAnalyze () [pure virtual]

Invoked at the end of an Analysis pass.

See also

[AAX_eProcessingState_StopPass](#), [AAX_eProcessingState_EndPassGroup](#)

Implemented in [AAX_CHostProcessor](#).

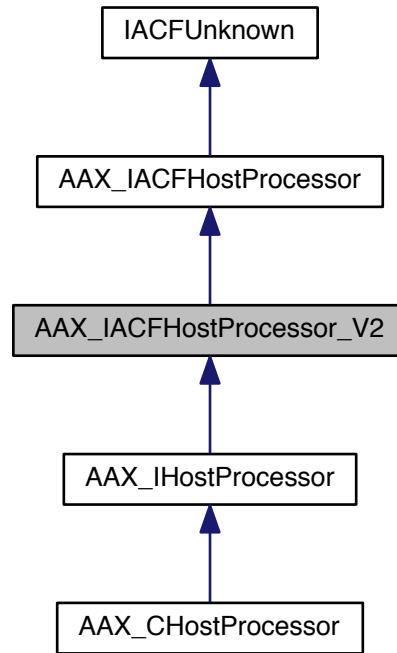
The documentation for this class was generated from the following file:

- [AAX_IACFHostProcessor.h](#)

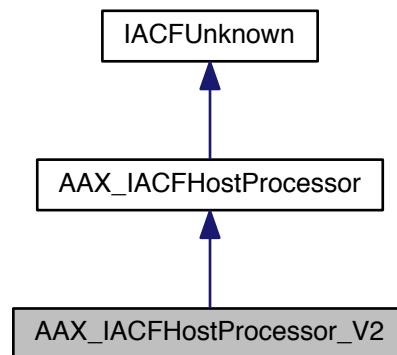
14.60 AAX_IACFHostProcessor_V2 Class Reference

```
#include <AAX_IACFHostProcessor.h>
```

Inheritance diagram for AAX_IACFHostProcessor_V2:



Collaboration diagram for AAX_IACFHostProcessor_V2:



14.60.1 Description

Supplemental interface for an AAX host processing component.

Note

This interface gets exposed to the host application. See [AAX_CHostProcessor](#) for method documentation.

Public Member Functions

- virtual [AAX_Result GetClipNameSuffix \(int32_t inMaxLength, AAX_IString *outString\) const =0](#)

Called by host application to retrieve a custom string to be appended to the clip name.

14.60.2 Member Function Documentation

- 14.60.2.1 virtual [AAX_Result AAX_IACFHostProcessor_V2::GetClipNameSuffix \(int32_t inMaxLength, AAX_IString * outString \) const \[pure virtual\]](#)

Called by host application to retrieve a custom string to be appended to the clip name.

If no string is provided then the host's default will be used.

Parameters

in	<i>inMaxLength</i>	The maximum allowed string length, not including the NULL terminating char
out	<i>outString</i>	Add a value to this string to provide a custom clip suffix

Implemented in [AAX_CHostProcessor](#).

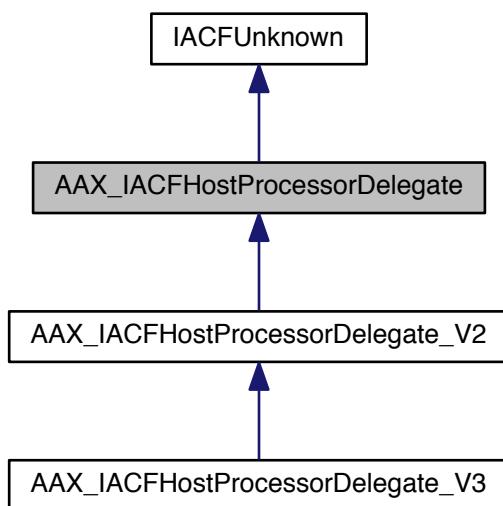
The documentation for this class was generated from the following file:

- [AAX_IACFHostProcessor.h](#)

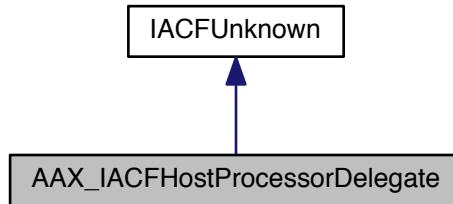
14.61 AAX_IACFHostProcessorDelegate Class Reference

```
#include <AAX_IACFHostProcessorDelegate.h>
```

Inheritance diagram for AAX_IACFHostProcessorDelegate:



Collaboration diagram for AAX_IACFHostProcessorDelegate:



14.61.1 Description

Versioned interface for host methods specific to offline processing.

Public Member Functions

- virtual [AAX_Result GetAudio](#) (const float *const *inAudioIns*[], int32_t *inAudioInCount*, int64_t *inLocation*, int32_t **ioNumSamples*)=0
CALL: Randomly access audio from the timeline.
- virtual int32_t [GetSideChainInputNum](#) ()=0
CALL: Returns the index of the side chain input buffer.

14.61.2 Member Function Documentation

14.61.2.1 virtual AAX_Result AAX_IACFHostProcessorDelegate::GetAudio (const float *const *inAudioIns*[], int32_t *inAudioInCount*, int64_t *inLocation*, int32_t * *ioNumSamples*) [pure virtual]

CALL: Randomly access audio from the timeline.

Called from within [AAX_IHostProcessor::RenderAudio\(\)](#), this method fills a buffer of samples with randomly-accessed data from the current input processing region on the timeline, including any extra samples such as processing "handles".

Note

Plug-ins that use this feature must set [AAX_eProperty_UsesRandomAccess](#) to true
It is not possible to retrieve samples from outside of the current input processing region
Always check the return value of this method before using the randomly-accessed samples

Parameters

<i>in</i>	<i>inAudioIns</i>	Timeline audio buffer(s). This must be set to <i>inAudioIns</i> from AAX_IHostProcessor::RenderAudio()
-----------	-------------------	--

Parameters

in	<i>inAudioInCount</i>	Number of buffers in <i>inAudioIns</i> . This must be set to <i>inAudioInCount</i> from AAX_IHostProcessor::RenderAudio()
----	-----------------------	---

Parameters

in	<i>inLocation</i>	A sample location relative to the beginning of the currently processed region, e.g. a value of 0 corresponds to the timeline location returned by AAX_CHostProcessor::GetSrcStart()
in, out	<i>ioNumSamples</i>	<ul style="list-style-type: none"> • Input: The maximum number of samples to read. • Output: The actual number of samples that were read from the timeline

14.61.2.2 virtual int32_t AAX_IACFHostProcessorDelegate::GetSideChainInputNum() [pure virtual]

CALL: Returns the index of the side chain input buffer.

Called from within [AAX_IHostProcessor::RenderAudio\(\)](#), this method returns the index of the side chain input sample buffer within *inAudioIns*.

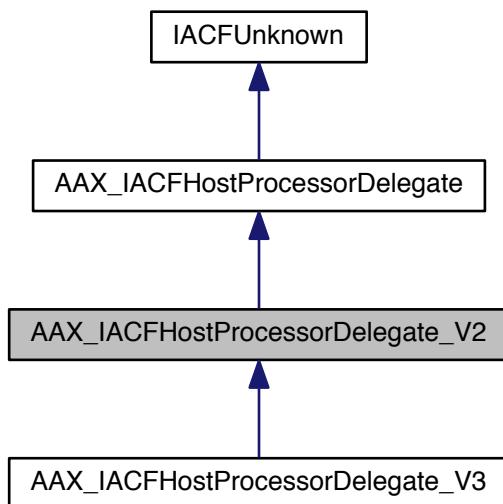
The documentation for this class was generated from the following file:

- [AAX_IACFHostProcessorDelegate.h](#)

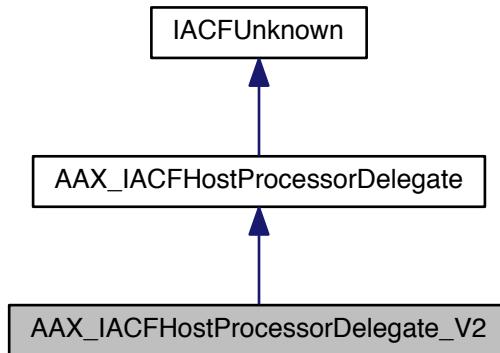
14.62 AAX_IACFHostProcessorDelegate_V2 Class Reference

```
#include <AAX_IACFHostProcessorDelegate.h>
```

Inheritance diagram for AAX_IACFHostProcessorDelegate_V2:



Collaboration diagram for AAX_IACFHostProcessorDelegate_V2:



14.62.1 Description

Versioned interface for host methods specific to offline processing.

Public Member Functions

- virtual [AAX_Result ForceAnalyze \(\)=0](#)

CALL: Request an analysis pass.

14.62.2 Member Function Documentation

14.62.2.1 virtual [AAX_Result AAX_IACFHostProcessorDelegate_V2::ForceAnalyze \(\) \[pure virtual\]](#)

CALL: Request an analysis pass.

Call this method to request an analysis pass from within the plug-in. Most plug-ins should rely on the host to trigger analysis passes when appropriate. However, plug-ins that require an analysis pass a) outside of the context of host-driven render or analysis, or b) when internal plug-in data changes may need to call [ForceAnalyze \(\)](#).

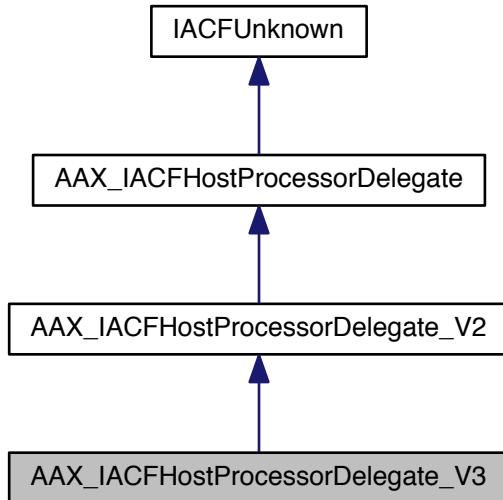
The documentation for this class was generated from the following file:

- [AAX_IACFHostProcessorDelegate.h](#)

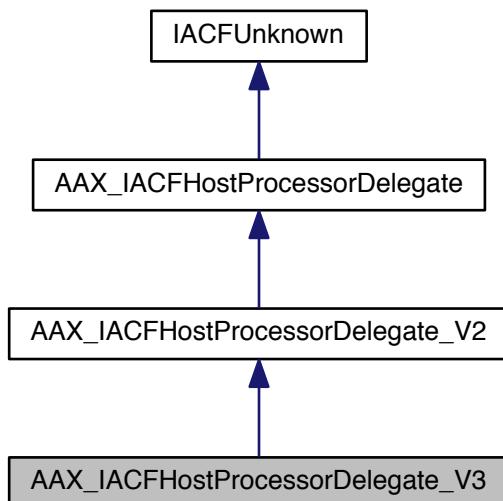
14.63 AAX_IACFHostProcessorDelegate_V3 Class Reference

```
#include <AAX_IACFHostProcessorDelegate.h>
```

Inheritance diagram for AAX_IACFHostProcessorDelegate_V3:



Collaboration diagram for AAX_IACFHostProcessorDelegate_V3:



14.63.1 Description

Versioned interface for host methods specific to offline processing.

Public Member Functions

- virtual [AAX_Result ForceProcess \(\)=0](#)

CALL: Request a process pass.

14.63.2 Member Function Documentation

14.63.2.1 virtual [AAX_Result AAX_IACFHostProcessorDelegate_V3::ForceProcess\(\) \[pure virtual\]](#)

CALL: Request a process pass.

Call this method to request a process pass from within the plug-in. If [AAX_eProperty_RequiresAnalysis](#) is defined, the resulting process pass will be preceded by an analysis pass. This method should only be used in rare circumstances by plug-ins that must launch processing outside of the normal host AudioSuite workflow.

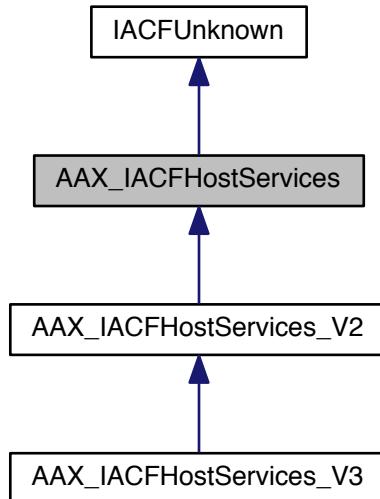
The documentation for this class was generated from the following file:

- [AAX_IACFHostProcessorDelegate.h](#)

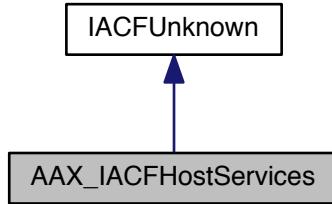
14.64 AAX_IACFHostServices Class Reference

```
#include <AAX_IACFHostServices.h>
```

Inheritance diagram for AAX_IACFHostServices:



Collaboration diagram for AAX_IACFHostServices:



14.64.1 Description

Versioned interface to diagnostic and debugging services provided by the AAX host.

Public Member Functions

- virtual [AAX_Result Assert](#) (const char *iFile, int32_t iLine, const char *iNote)=0
- virtual [AAX_Result Trace](#) (int32_t iPriority, const char *iMessage)=0

Log a trace message.

14.64.2 Member Function Documentation

14.64.2.1 virtual AAX_Result AAX_IACFHostServices::Assert (const char * iFile, int32_t iLine, const char * iNote) [pure virtual]

Deprecated Legacy version of [AAX_IACFHostServices_V3::HandleAssertFailure\(\)](#) implemented by older hosts

Prior to [AAX_IACFHostServices_V3::HandleAssertFailure\(\)](#), the [AAX_ASSERT](#) macro, a wrapper around [Assert\(\)](#), was only compiled into debug plug-in builds. [AAX_ASSERT](#) is now compiled in to all plug-in builds and the original debug-only form is available through [AAX_DEBUGASSERT](#).

Because the implementation of [Assert\(\)](#) in the host is not aware of the plug-in's build configuration, older hosts implemented this method with a warning dialog in all cases. Newer hosts - those which implement [HandleAssertFailure\(\)](#) - will log assertion failures but will not present any user dialog in shipping builds of the host software.

In order to prevent assertion failure dialogs from appearing to users who run new builds of plug-ins containing [AAX_ASSERT](#) calls in older hosts the deprecated [Assert\(\)](#) method should only be called from debug plug-in builds.

14.64.2.2 virtual AAX_Result AAX_IACFHostServices::Trace (int32_t iPriority, const char * iMessage) [pure virtual]

Log a trace message.

Parameters

in	<i>iPriority</i>	Priority of the trace, used for log filtering. One of kAAX_Trace_Priority_Low , kAAX_Trace_Priority_Normal , kAAX_Trace_Priority_High
in	<i>iMessage</i>	Message string to log

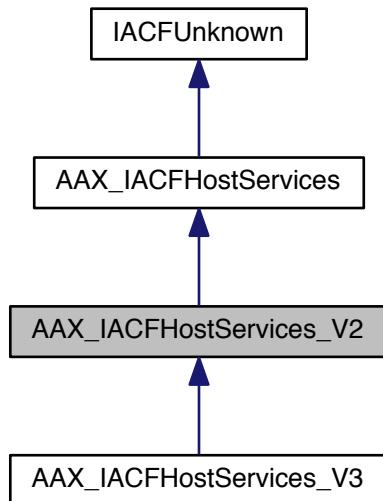
The documentation for this class was generated from the following file:

- [AAX_IACFHostServices.h](#)

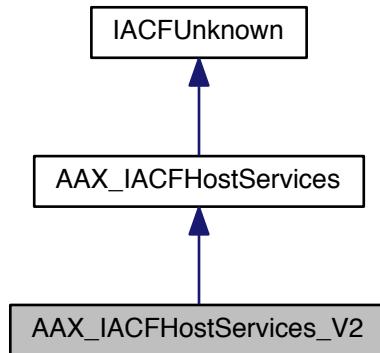
14.65 AAX_IACFHostServices_V2 Class Reference

```
#include <AAX_IACFHostServices.h>
```

Inheritance diagram for AAX_IACFHostServices_V2:



Collaboration diagram for AAX_IACFHostServices_V2:



14.65.1 Description

V2 of versioned interface to diagnostic and debugging services provided by the AAX host.

Public Member Functions

- virtual [AAX_Result StackTrace](#) (int32_t *iTracePriority*, int32_t *iStackTracePriority*, const char **iMessage*)=0
Log a trace message or a stack trace.

14.65.2 Member Function Documentation

14.65.2.1 virtual AAX_Result AAX_IACFHostServices_V2::StackTrace (int32_t *iTracePriority*, int32_t *iStackTracePriority*, const char * *iMessage*) [pure virtual]

Log a trace message or a stack trace.

If the logging output filtering is set to include logs with *iStackTracePriority* then both the logging message and a stack trace will be emitted, regardless of *iTracePriority*.

If the logging output filtering is set to include logs with *iTracePriority* but to exclude logs with *iStackTracePriority* then this will emit a normal log with no stack trace.

Parameters

in	<i>iTracePriority</i>	Priority of the trace, used for log filtering. One of kAAX_Trace_Priority_Low , kAAX_Trace_Priority_Normal , kAAX_Trace_Priority_High
in	<i>iStackTracePriority</i>	Priority of the stack trace, used for log filtering. One of kAAX_Trace_Priority_Low , kAAX_Trace_Priority_Normal , kAAX_Trace_Priority_High
in	<i>iMessage</i>	Message string to log

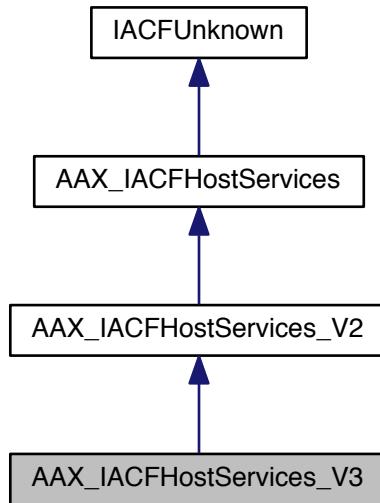
The documentation for this class was generated from the following file:

- [AAX_IACFHostServices.h](#)

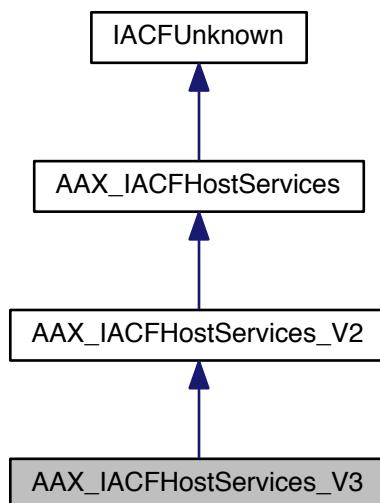
14.66 AAX_IACFHostServices_V3 Class Reference

```
#include <AAX_IACFHostServices.h>
```

Inheritance diagram for AAX_IACFHostServices_V3:



Collaboration diagram for AAX_IACFHostServices_V3:



14.66.1 Description

V3 of versioned interface to diagnostic and debugging services provided by the AAX host.

Public Member Functions

- virtual [AAX_Result HandleAssertFailure](#) (const char **iFile*, int32_t *iLine*, const char **iNote*, int32_t *iFlags*)
const =0

Handle an assertion failure.

14.66.2 Member Function Documentation

14.66.2.1 virtual [AAX_Result AAX_IACFHostServices_V3::HandleAssertFailure](#) (const char * *iFile*, int32_t *iLine*, const char * *iNote*, int32_t *iFlags*) const [pure virtual]

Handle an assertion failure.

Use this method to delegate assertion failure handling to the host

Use *iFlags* to request that specific behavior be included when handling the failure. This request may not be fulfilled by the host, and absence of a flag does not preclude the host from using that behavior when handling the failure.

Parameters

in	<i>iFile</i>	The name of the file containing the assert check. Usually <code>__FILE__</code>
in	<i>iLine</i>	The line number of the assert check. Usually <code>__LINE__</code>
in	<i>iNote</i>	Text to display related to the assert. Usually the condition which failed
in	<i>iFlags</i>	Bitfield of AAX_EAssertFlags to request specific handling behavior

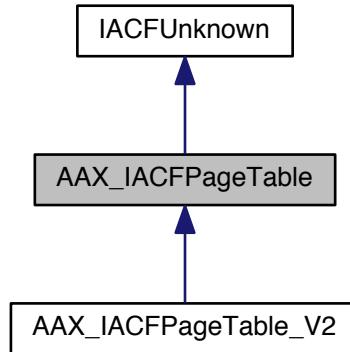
The documentation for this class was generated from the following file:

- [AAX_IACFHostServices.h](#)

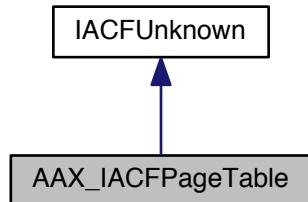
14.67 AAX_IACFPageTable Class Reference

```
#include <AAX_IACFPageTable.h>
```

Inheritance diagram for AAX_IACFPageTable:



Collaboration diagram for AAX_IACFPageTable:



14.67.1 Description

Versioned interface to the host's representation of a plug-in instance's page table.

Public Member Functions

- virtual [AAX_Result Clear \(\)=0](#)
Clears all parameter mappings from the table.
- virtual [AAX_Result Empty \(AAX_CBoolean &oEmpty\) const =0](#)
Indicates whether the table is empty.
- virtual [AAX_Result GetNumPages \(int32_t &oNumPages\) const =0](#)
Get the number of pages currently in this table.
- virtual [AAX_Result InsertPage \(int32_t iPage\)=0](#)
*Insert a new empty page before the page at index *iPage*.*
- virtual [AAX_Result RemovePage \(int32_t iPage\)=0](#)
*Remove the page at index *iPage*.*

- virtual [AAx_Result GetNumMappedParameterIDs](#) (int32_t iPage, int32_t &oNumParameterIdentifiers) const =0
Returns the total number of parameter IDs which are mapped to a page.
- virtual [AAx_Result ClearMappedParameter](#) (int32_t iPage, int32_t iIndex)=0
Clear the parameter at a particular index in this table.
- virtual [AAx_Result GetMappedParameterID](#) (int32_t iPage, int32_t iIndex, [AAx_IString &oParameterIdentifier](#)) const =0
Get the parameter identifier which is currently mapped to an index in this table.
- virtual [AAx_Result MapParameterID](#) ([AAx_CParamID iParameterIdentifier](#), int32_t iPage, int32_t iIndex)=0
Map a parameter to this table.

14.67.2 Member Function Documentation

14.67.2.1 virtual [AAx_Result AAX_IACFPageTable::Clear](#) () [pure virtual]

Clears all parameter mappings from the table.

This method does not clear any parameter name variations from the table. For that, use [AAx_IPageTable::ClearParameterNameVariations\(\)](#) or [AAx_IPageTable::ClearNameVariationsForParameter\(\)](#)

14.67.2.2 virtual [AAx_Result AAX_IACFPageTable::Empty](#) ([AAx_CBoolean & oEmpty](#)) const [pure virtual]

Indicates whether the table is empty.

A table is empty if it contains no pages. A table which contains pages but no parameter assignments is not empty. A table which has associated parameter name variations but no pages is empty.

Parameters

out	oEmpty	true if this table is empty
---------------------	------------------------	-----------------------------

14.67.2.3 virtual [AAx_Result AAX_IACFPageTable::GetNumPages](#) (int32_t & oNumPages) const [pure virtual]

Get the number of pages currently in this table.

Parameters

out	oNumPages	The number of pages which are present in the page table. Some pages might not contain any parameter assignments.
---------------------	---------------------------	--

14.67.2.4 virtual [AAx_Result AAX_IACFPageTable::InsertPage](#) (int32_t iPage) [pure virtual]

Insert a new empty page before the page at index [iPage](#).

Returns

[AAx_Error_Invalid_Argument](#) if [iPage](#) is greater than the total number of pages

Parameters

in	iPage	The insertion point page index
--------------------	-----------------------	--------------------------------

14.67.2.5 virtual AAX_Result AAX_IACFPageTable::RemovePage (int32_t *iPage*) [pure virtual]

Remove the page at index *iPage*.

Returns

[AAX_ERROR_INVALID_ARGUMENT](#) if *iPage* is greater than the index of the last existing page

Parameters

in	<i>iPage</i>	The target page index
----	--------------	-----------------------

14.67.2.6 virtual AAX_Result AAX_IACFPageTable::GetNumMappedParameterIDs (int32_t *iPage*, int32_t & *oNumParameterIdentifiers*) const [pure virtual]

Returns the total number of parameter IDs which are mapped to a page.

Note

The number of mapped parameter IDs does not correspond to the actual slot indices of the parameter assignments. For example, a page could have three total parameter assignments with parameters mapped to slots 2, 4, and 6.

Returns

[AAX_ERROR_INVALID_ARGUMENT](#) if *iPage* is greater than the index of the last existing page

Parameters

in	<i>iPage</i>	The target page index
out	<i>oNumParameterIdentifiers</i>	The number of parameter identifiers which are mapped to the target page

14.67.2.7 virtual AAX_Result AAX_IACFPageTable::ClearMappedParameter (int32_t *iPage*, int32_t *iIndex*) [pure virtual]

Clear the parameter at a particular index in this table.

Returns

[AAX_SUCCESS](#) even if no parameter was mapped at the given index (the index is still clear)

Parameters

in	<i>iPage</i>	The target page index
in	<i>iIndex</i>	The target parameter slot index within the target page

14.67.2.8 virtual AAX_Result AAX_IACFPageTable::GetMappedParameterID (int32_t *iPage*, int32_t *iIndex*, AAX_IString & *oParameterIdentifier*) const [pure virtual]

Get the parameter identifier which is currently mapped to an index in this table.

Returns

[AAX_ERROR_INVALID_ARGUMENT](#) if no parameter is mapped at the specified page/index location

Parameters

in	<i>iPage</i>	The target page index
in	<i>iIndex</i>	The target parameter slot index within the target page
out	<i>oParameterIdentifier</i>	The identifier used for the mapped parameter in the page table (may be parameter name or ID)

14.67.2.9 virtual AAX_Result AAX_IACFPageTable::MapParameterID (AAX_CParamID iParameterIdentifier, int32_t iPage, int32_t iIndex) [pure virtual]

Map a parameter to this table.

If *iParameterIdentifier* is an empty string then the parameter assignment will be cleared

Returns

[AAX_ERROR_NULL_ARGUMENT](#) if *iParameterIdentifier* is null
[AAX_ERROR_INVALID_ARGUMENT](#) if *iPage* is greater than the index of the last existing page
[AAX_ERROR_INVALID_ARGUMENT](#) if *iIndex* is negative

Parameters

in	<i>iParameterIdentifier</i>	The identifier for the parameter which will be mapped
in	<i>iPage</i>	The target page index
in	<i>iIndex</i>	The target parameter slot index within the target page

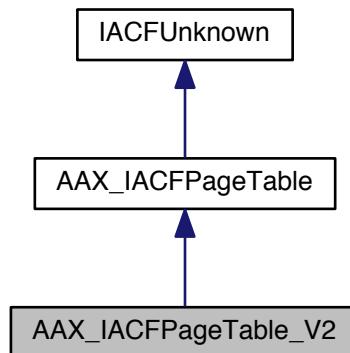
The documentation for this class was generated from the following file:

- [AAX_IACFPageTable.h](#)

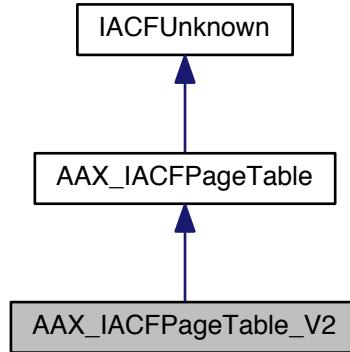
14.68 AAX_IACFPageTable_V2 Class Reference

```
#include <AAX_IACFPageTable.h>
```

Inheritance diagram for AAX_IACFPageTable_V2:



Collaboration diagram for AAX_IACFPageTable_V2:



14.68.1 Description

Versioned interface to the host's representation of a plug-in instance's page table.

Public Member Functions

- virtual [AAX_Result GetNumParametersWithNameVariations \(int32_t &oNumParameterIdentifiers\) const =0](#)
- virtual [AAX_Result GetNameVariationParameterIDAtIndex \(int32_t iIndex, AAX_IString &oParameterIdentifier\) const =0](#)
- virtual [AAX_Result GetNumNameVariationsForParameter \(AAX_CPageTableParamID iParameterIdentifier, int32_t &oNumVariations\) const =0](#)
- virtual [AAX_Result GetParameterNameVariationAtIndex \(AAX_CPageTableParamID iParameterIdentifier, int32_t iIndex, AAX_IString &oNameVariation, int32_t &oLength\) const =0](#)
- virtual [AAX_Result GetParameterNameVariationOfLength \(AAX_CPageTableParamID iParameterIdentifier, int32_t iLength, AAX_IString &oNameVariation\) const =0](#)
- virtual [AAX_Result ClearParameterNameVariations \(\)=0](#)
- virtual [AAX_Result ClearNameVariationsForParameter \(AAX_CPageTableParamID iParameterIdentifier\)=0](#)
- virtual [AAX_Result SetParameterNameVariation \(AAX_CPageTableParamID iParameterIdentifier, const AAX_IString &iNameVariation, int32_t iLength\)=0](#)

14.68.2 Member Function Documentation

14.68.2.1 virtual [AAX_Result AAX_IACFPageTable_V2::GetNumParametersWithNameVariations \(int32_t & oNumParameterIdentifiers \) const \[pure virtual\]](#)

Get the number of parameters with name variations defined for the current table type

Provides the number of parameters with `lt;ControlNameVariationslt;` which are explicitly defined for the current page table type.

Note

Normally parameter name variations are only used with the 'PgTL' table type

See also

- [AAX_IPageTable::GetNameVariationParameterIDAtIndex\(\)](#)

Parameters

out	<i>oNum</i> <i>Parameter</i> <i>Identifiers</i>	The number of parameters with name variations explicitly associated with the current table type.
------------	---	--

**14.68.2.2 virtual AAX_Result AAX_IACFPageTable_V2::GetNameVariationParameterIDAtIndex (int32_t *iIndex*,
AAX_IString & *oParameterIdentifier*) const [pure virtual]**

Get the identifier for a parameter with name variations defined for the current table type

Note

Normally parameter name variations are only used with the 'PgTL' table type

See also

- [AAX_IPageTable::GetNumParametersWithNameVariations\(\)](#)

Parameters

in	<i>iIndex</i>	The target parameter index within the list of parameters with explicit name variations defined for this table type.
out	<i>oParameter</i> <i>Identifier</i>	The identifier used for the parameter in the page table name variations list (may be parameter name or ID)

14.68.2.3 virtual AAX_Result AAX_IACFPageTable_V2::GetNumNameVariationsForParameter (AAX_CPageTableParamID *iParameterIdentifier*, int32_t & *oNumVariations*) const [pure virtual]

Get the number of name variations defined for a parameter

Provides the number of *lt;ControlNameVariations*; which are explicitly defined for *iParameterIdentifier* for the current page table type. No fallback logic is used to resolve this to the list of variations which would actually be used for an attached control surface if no explicit variations are defined for the current table type.

Note

Normally parameter name variations are only used with the 'PgTL' table type

See also

- [AAX_IPageTable::GetParameterNameVariationAtIndex\(\)](#)

Returns

AAX_SUCCESS and provides zero to *oNumVariations* if *iParameterIdentifier* is not found

Parameters

in	<i>iParameter</i> <i>Identifier</i>	The identifier for the parameter
-----------	--	----------------------------------

out	<i>oNumVariations</i>	The number of name variations which are defined for this parameter and explicitly associated with the current table type.
-----	-----------------------	---

```
14.68.2.4 virtual AAX_Result AAX_IACFPageTable_V2::GetParameterNameVariationAtIndex ( AAX_CPageTableParamID
iParameterIdentifier, int32_t iIndex, AAX_IString & oNameVariation, int32_t & oLength ) const [pure
virtual]
```

Get a parameter name variation from the page table

Only returns `lt;ControlNameVariations` which are explicitly defined for the current page table type. No fallback logic is used to resolve this to the abbreviation which would actually be shown on an attached control surface if no explicit variation is defined for the current table type.

Note

Normally parameter name variations are only used with the 'PgTL' table type

See also

- [AAX_IPageTable::GetNumNameVariationsForParameter\(\)](#)

See also

- [AAX_IPageTable::GetParameterNameVariationOfLength\(\)](#)

Returns

`AAX_ERROR_NO_ABBREVIATED_PARAMETER_NAME` if no suitable variation is defined for this table
`AAX_ERROR_ARGUMENT_OUT_OF_RANGE` if *iIndex* is out of range

Parameters

in	<i>iParameterIdentifier</i>	The identifier for the parameter
in	<i>iIndex</i>	Index of the name variation
out	<i>oNameVariation</i>	The name variation, if one is explicitly defined for this table type
out	<i>oLength</i>	The length value for this name variation. This corresponds to the variation's <code>sZ</code> attribute in the page table XML and may be different from the string length of <i>iNameVariation</i> .

```
14.68.2.5 virtual AAX_Result AAX_IACFPageTable_V2::GetParameterNameVariationOfLength ( AAX_CPageTableParamID
iParameterIdentifier, int32_t iLength, AAX_IString & oNameVariation ) const [pure
virtual]
```

Get a parameter name variation of a particular length from the page table

Only returns `lt;ControlNameVariations` which are explicitly defined of *iLength* for the current page table type. No fallback logic is used to resolve this to the abbreviation which would actually be shown on an attached control surface if no explicit variation is defined for the specified length or current table type.

Note

Normally parameter name variations are only used with the 'PgTL' table type

See also

- [AAX_IPageTable::GetParameterNameVariationAtIndex\(\)](#)

Returns

`AAX_ERROR_NO_ABBREVIATED_PARAMETER_NAME` if no suitable variation is defined for this table

Parameters

in	<i>iParameterIdentifier</i>	The identifier for the parameter
in	<i>iLength</i>	The variation length to check, i.e. the <code>sz</code> attribute for the name variation in the page table XML
out	<i>oNameVariation</i>	The name variation, if one is explicitly defined for this table type and <i>iLength</i>

14.68.2.6 virtual AAX_Result AAX_IACFPageTable_V2::ClearParameterNameVariations() [pure virtual]

Clears all name variations for the current page table type

Note

Normally parameter name variations are only used with the 'PgTL' table type

See also

[AAX_IPageTable::Clear\(\)](#)
[AAX_IPageTable::ClearNameVariationsForParameter\(\)](#)

14.68.2.7 virtual AAX_Result AAX_IACFPageTable_V2::ClearNameVariationsForParameter(AAX_CPageTableParamID iParameterIdentifier) [pure virtual]

Clears all name variations for a single parameter for the current page table type

Warning

This will invalidate the list of parameter name variations indices, i.e. the parameter identifier associated with each index by [AAX_IPageTable::GetNameVariationParameterIDAtIndex\(\)](#)

Note

Normally parameter name variations are only used with the 'PgTL' table type

See also

[AAX_IPageTable::Clear\(\)](#)
[AAX_IPageTable::ClearParameterNameVariations\(\)](#)

Returns

[AAX_SUCCESS](#) and provides zero to *oNumVariations* if *iParameterIdentifier* is not found

Parameters

in	<i>iParameterIdentifier</i>	The identifier for the parameter
----	-----------------------------	----------------------------------

14.68.2.8 virtual AAX_Result AAX_IACFPageTable_V2::SetParameterNameVariation(AAX_CPageTableParamID iParameterIdentifier, const AAX_IString & iNameVariation, int32_t iLength) [pure virtual]

Sets a name variation explicitly for the current page table type

This will add a new name variation or overwrite the existing name variation with the same length which is defined for the current table type.

Warning

If no name variation previously existed for this parameter then this will invalidate the list of parameter name variations indices, i.e. the parameter identifier associated with each index by [AAX_IPageTable::GetNameVariationParameterIDAtIndex\(\)](#)

Note

Normally parameter name variations are only used with the 'PgTL' table type

Returns

AAX_ERROR_INVALID_ARGUMENT if *iNameVariation* is empty or if *iLength* is less than zero

Parameters

in	<i>iParameterIdentifier</i>	The identifier for the parameter
in	<i>iNameVariation</i>	The new parameter name variation
in	<i>iLength</i>	The length value for this name variation. This corresponds to the variation's sz attribute in the page table XML and is not required to match the length of <i>iNameVariation</i> .

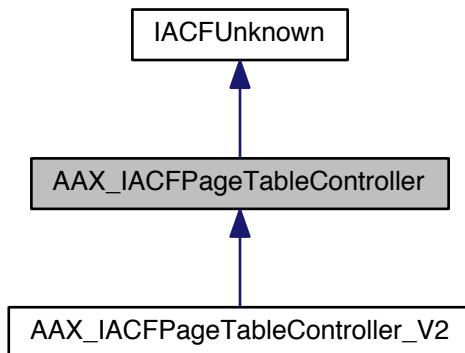
The documentation for this class was generated from the following file:

- [AAX_IACFPageTable.h](#)

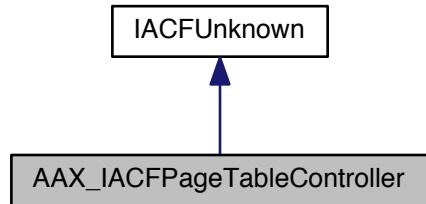
14.69 AAX_IACFPageTableController Class Reference

```
#include <AAX_IACFPageTableController.h>
```

Inheritance diagram for AAX_IACFPageTableController:



Collaboration diagram for AAX_IACFPageTableController:



14.69.1 Description

Interface for host operations related to the page tables for this plug-in.

Note

In the AAX Library, access to this interface is provided through [AAX_IController](#)

Public Member Functions

- virtual `AAX_Result CopyTableForEffect (AAX_CPropertyValue inManufacturerID, AAX_CPropertyValue inProductID, AAX_CPropertyValue inPlugInID, uint32_t inTableType, int32_t inTablePageSize, IACFUnknown *oPageTable) const =0`
- virtual `AAX_Result CopyTableOfLayoutForEffect (const char *inEffectID, const char *inLayoutName, uint32_t inTableType, int32_t inTablePageSize, IACFUnknown *oPageTable) const =0`

14.69.2 Member Function Documentation

14.69.2.1 virtual AAX_Result AAX_IACFPageTableController::CopyTableForEffect (AAX_CPropertyValue inManufacturerID, AAX_CPropertyValue inProductID, AAX_CPropertyValue inPlugInID, uint32_t inTableType, int32_t inTablePageSize, IACFUnknown * oPageTable) const [pure virtual]

Copy the current page table data for a particular plug-in type.

The host will reject the copy and return an error if the requested plug-in type is unknown, if `inTableType` is unknown or if `inTablePageSize` is not a supported size for the given table type.

The host may also restrict plug-ins to only copying page table data from certain plug-in types, such as plug-ins from the same manufacturer or plug-in types within the same effect.

See [Page Table Guide](#) for more information about page tables.

Returns

`AAX_ERROR_NULL_ARGUMENT` if `oPageTable` is null
`AAX_ERROR_INVALID_ARGUMENT` if no valid page table mapping can be created due to the specified arguments

Parameters

in	<i>inManufacturerID</i>	Manufacturer ID of the desired plug-in type
in	<i>inProductID</i>	Product ID of the desired plug-in type
in	<i>inPlugInID</i>	Type ID of the desired plug-in type (AAX_eProperty_PlugInID_Native , AAX_eProperty_PlugInID_TI)
in	<i>inTableType</i>	Four-char type identifier for the requested table type (e.g. 'PgTL', 'Av81', etc.)
in	<i>inTablePageSize</i>	Page size for the requested table. Some tables support multiple page sizes.
out	<i>oPageTable</i>	The page table object to which the page table data should be copied. <i>oPageTable</i> must support AAX_IACFPageTable

See also

[AAX_IController::CreateTableCopyForEffect\(\)](#)

```
14.69.2.2 virtual AAX_Result AAX_IACFPageTableController::CopyTableOfLayoutForEffect ( const char * inEffectID, const
char * inLayoutName, uint32_t inTableType, int32_t inTablePageSize, IACFUnknown * oPageTable ) const
[pure virtual]
```

Copy the current page table data for a particular plug-in effect and page table layout.

The host will reject the copy and return an error if the requested effect ID is unknown or if *inLayoutName* is not a valid layout name for the page tables registered for the effect.

The host may also restrict plug-ins to only copying page table data from certain effects, such as effects registered within the current [AAX](#) plug-in bundle.

See [Page Table Guide](#) for more information about page tables.

Returns

[AAX_ERROR_NULL_ARGUMENT](#) if *inEffectID*, *inLayoutName*, or *oPageTable* is null
[AAX_ERROR_INVALID_ARGUMENT](#) if no valid page table mapping can be created due to the specified arguments

Parameters

in	<i>inEffectID</i>	Effect ID for the desired effect. See AAX_ICollection::AddEffect()
in	<i>inLayoutName</i>	Page table layout name ("name" attribute of the PTLayout XML tag)
in	<i>inTableType</i>	Four-char type identifier for the requested table type (e.g. 'PgTL', 'Av81', etc.)
in	<i>inTablePageSize</i>	Page size for the requested table. Some tables support multiple page sizes.
out	<i>oPageTable</i>	The page table object to which the page table data should be copied. <i>oPageTable</i> must support AAX_IACFPageTable

See also

[AAX_IController::CreateTableCopyForLayout\(\)](#)

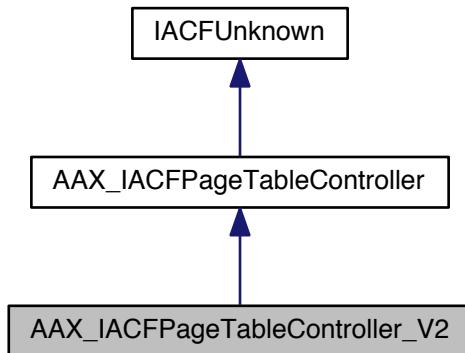
The documentation for this class was generated from the following file:

- [AAX_IACFPageTableController.h](#)

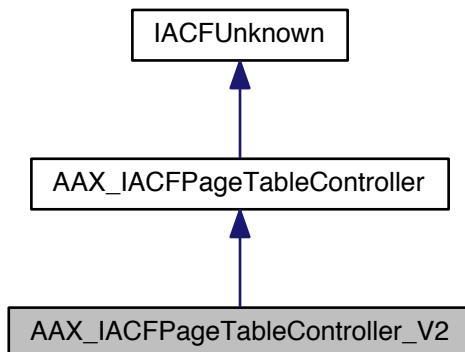
14.70 AAX_IACFPageTableController_V2 Class Reference

```
#include <AAX_IACFPageTableController.h>
```

Inheritance diagram for AAX_IACFPageTableController_V2:



Collaboration diagram for AAX_IACFPageTableController_V2:



14.70.1 Description

Interface for host operations related to the page tables for this plug-in.

Note

In the AAX Library, access to this interface is provided through [AAX_IController](#)

Public Member Functions

- virtual `AAX_Result CopyTableForEffectFromFile (const char *inPageTableFilePath, AAX_ETextEncoding inFilePathEncoding, AAX_CPropertyValue inManufacturerID, AAX_CPropertyValue inProductID, AAX_CPropertyValue inPlugInID, uint32_t inTableType, int32_t inTablePageSize, IACFUnknown *oPageTable) const =0`

- virtual [AAX_Result CopyTableOfLayoutFromFile](#) (const char *inPageTableFilePath, [AAX_ETextEncoding](#) inFilePathEncoding, const char *inLayoutName, uint32_t inTableType, int32_t inTablePageSize, [IACFUnknown](#) *oPageTable) const =0

14.70.2 Member Function Documentation

14.70.2.1 virtual [AAX_Result AAX_IACFPageTableController_V2::CopyTableForEffectFromFile](#) (const char *
inPageTableFilePath, [AAX_ETextEncoding](#) *inFilePathEncoding*, [AAX_CPropertyValue](#) *inManufacturerID*,
[AAX_CPropertyValue](#) *inProductID*, [AAX_CPropertyValue](#) *inPlugInID*, uint32_t *inTableType*, int32_t
inTablePageSize, [IACFUnknown](#) * *oPageTable*) const [pure virtual]

Returns

[AAX_ERROR_NULL_ARGUMENT](#) if *inPageTableFilePath* or *oPageTable* is null
[AAX_ERROR_UNSUPPORTED_ENCODING](#) if *inFilePathEncoding* has unsupported encoding value
[AAX_ERROR_INVALID_ARGUMENT](#) if no valid page table mapping can be created due to the specified arguments

Parameters

in	<i>inPageTableFilePath</i>	Path to XML page table file.
in	<i>inFilePathEncoding</i>	File path text encoding.
in	<i>inManufacturerID</i>	Manufacturer ID of the desired plug-in type
in	<i>inProductID</i>	Product ID of the desired plug-in type
in	<i>inPlugInID</i>	Type ID of the desired plug-in type (AAX_eProperty_PlugInID_Native , AAX_eProperty_PlugInID_TI)
in	<i>inTableType</i>	Four-char type identifier for the requested table type (e.g. 'PgTL', 'Av81', etc.)
in	<i>inTablePageSize</i>	Page size for the requested table. Some tables support multiple page sizes.
out	<i>oPageTable</i>	The page table object to which the page table data should be copied. <i>oPageTable</i> must support AAX_IACFPageTable

See also

[AAX_IController::CreateTableCopyForEffect\(\)](#)

14.70.2.2 virtual [AAX_Result AAX_IACFPageTableController_V2::CopyTableOfLayoutFromFile](#) (const char *
inPageTableFilePath, [AAX_ETextEncoding](#) *inFilePathEncoding*, const char * *inLayoutName*, uint32_t
inTableType, int32_t *inTablePageSize*, [IACFUnknown](#) * *oPageTable*) const [pure virtual]

Returns

[AAX_ERROR_NULL_ARGUMENT](#) if *inPageTableFilePath*, *inLayoutName*, or *oPageTable* is null
[AAX_ERROR_UNSUPPORTED_ENCODING](#) if *inFilePathEncoding* has unsupported encoding value
[AAX_ERROR_INVALID_ARGUMENT](#) if no valid page table mapping can be created due to the specified arguments

Parameters

in	<i>inPageTable</i> ↪ <i>FilePath</i>	Path to XML page table file.
in	<i>inFilePath</i> ↪ <i>Encoding</i>	File path text encoding.
in	<i>inLayoutName</i>	Page table layout name ("name" attribute of the <code>PTLayout</code> XML tag)
in	<i>inTableType</i>	Four-char type identifier for the requested table type (e.g. 'PgTL', 'Av81', etc.)
in	<i>inTablePageSize</i>	Page size for the requested table. Some tables support multiple page sizes.
out	<i>oPageTable</i>	The page table object to which the page table data should be copied. $\circ \leftarrow$ PageTable must support AAX_IACFPageTable

See also

[AAX_IController::CreateTableCopyForLayout\(\)](#)

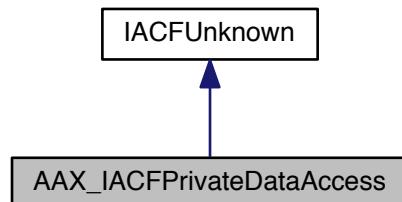
The documentation for this class was generated from the following file:

- [AAX_IACFPageTableController.h](#)

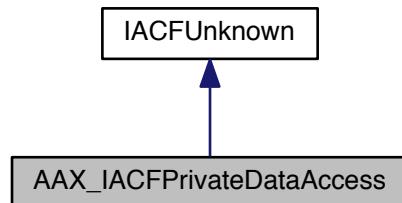
14.71 AAX_IACFPrivateDataAccess Class Reference

```
#include <AAX_IACFPrivateDataAccess.h>
```

Inheritance diagram for AAX_IACFPrivateDataAccess:



Collaboration diagram for AAX_IACFPrivateDataAccess:



14.71.1 Description

Interface for the AAX host's data access functionality.

Public Member Functions

- virtual `AAX_Result ReadPortDirect (AAX_CFieldIndex inFieldIndex, const uint32_t inOffset, const uint32_t inSize, void *outBuffer)=0`
Read data directly from DSP at the given port.
- virtual `AAX_Result WritePortDirect (AAX_CFieldIndex inFieldIndex, const uint32_t inOffset, const uint32_t inSize, const void *inBuffer)=0`
Write data directly to DSP at the given port.

14.71.2 Member Function Documentation

14.71.2.1 virtual AAX_Result AAX_IACFPrivateDataAccess::ReadPortDirect (`AAX_CFieldIndex inFieldIndex, const uint32_t inOffset, const uint32_t inSize, void * outBuffer`) [pure virtual]

Read data directly from DSP at the given port.

Note

Blocking

Parameters

in	<code>inFieldIndex</code>	The port to read from.
in	<code>inOffset</code>	Offset into data to start reading.
in	<code>inSize</code>	Amount of data to read (in bytes).
out	<code>outBuffer</code>	Pointer to storage for data to be read into.

14.71.2.2 virtual AAX_Result AAX_IACFPrivateDataAccess::WritePortDirect (`AAX_CFieldIndex inFieldIndex, const uint32_t inOffset, const uint32_t inSize, const void * inBuffer`) [pure virtual]

Write data directly to DSP at the given port.

Note

Blocking

Parameters

in	<code>inFieldIndex</code>	The port to write to.
in	<code>inOffset</code>	Offset into data to begin writing.
in	<code>inSize</code>	Amount of data to write (in bytes).
in	<code>inBuffer</code>	Pointer to data being written.

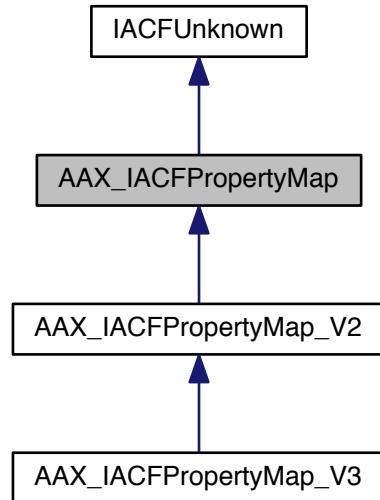
The documentation for this class was generated from the following file:

- `AAX_IACFPrivateDataAccess.h`

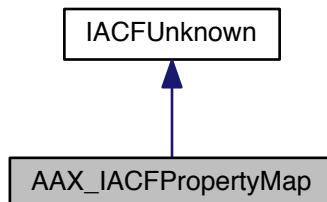
14.72 AAX_IACFPropertyMap Class Reference

```
#include <AAX_IACFPropertyMap.h>
```

Inheritance diagram for AAX_IACFPropertyMap:



Collaboration diagram for AAX_IACFPropertyMap:



14.72.1 Description

Versioned interface for an [AAX_IPropertyMap](#).

Public Member Functions

- virtual [AAX_CBoolean GetProperty \(AAX_EProperty inProperty, AAX_CPropertyValue *outValue\) const =0](#)
Get a property value from a property map.
- virtual [AAX_Result AddProperty \(AAX_EProperty inProperty, AAX_CPropertyValue inValue\)=0](#)
Add a property to a property map.
- virtual [AAX_Result RemoveProperty \(AAX_EProperty inProperty\)=0](#)
Remove a property from a property map.

14.72.2 Member Function Documentation

14.72.2.1 virtual **AAX_CBoolean** AAX_IACFPropertyMap::GetProperty (**AAX_EProperty** *inProperty*, **AAX_CPropertyValue** * *outValue*) const [pure virtual]

Get a property value from a property map.

Returns true if the selected property is supported, false if it is not

Parameters

in	<i>inProperty</i>	The property ID
out	<i>outValue</i>	The property value

14.72.2.2 virtual **AAX_Result** AAX_IACFPropertyMap::AddProperty (**AAX_EProperty** *inProperty*, **AAX_CPropertyValue** *inValue*) [pure virtual]

Add a property to a property map.

Note

This method may return an error if adding the property was unsuccessful. If there is a failure when adding a required property then registration of the relevant description element must be abandoned and the plug-in's description logic should proceed to the next element.

Parameters

in	<i>inProperty</i>	The property ID.
in	<i>inValue</i>	

14.72.2.3 virtual **AAX_Result** AAX_IACFPropertyMap::RemoveProperty (**AAX_EProperty** *inProperty*) [pure virtual]

Remove a property from a property map.

Parameters

in	<i>inProperty</i>	The property ID.
----	-------------------	------------------

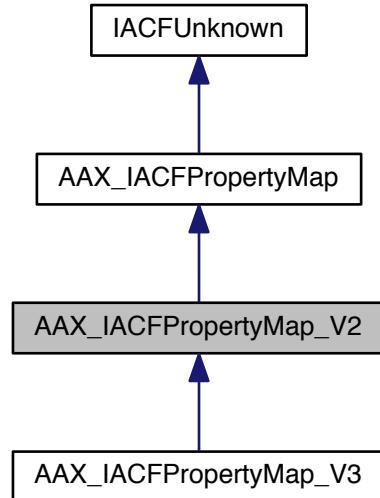
The documentation for this class was generated from the following file:

- [AAX_IACFPropertyMap.h](#)

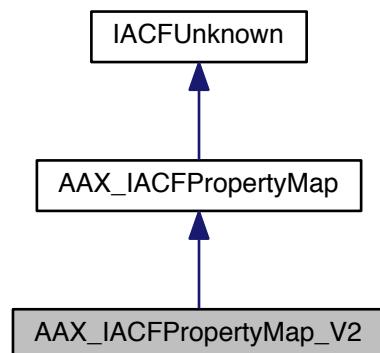
14.73 AAX_IACFPropertyMap_V2 Class Reference

```
#include <AAX_IACFPropertyMap.h>
```

Inheritance diagram for AAX_IACFPropertyMap_V2:



Collaboration diagram for AAX_IACFPropertyMap_V2:



14.73.1 Description

Versioned interface for an [AAX_IPropertyMap](#).

Public Member Functions

- virtual [AAX_Result AddPropertyWithIDArray \(AAX_EProperty inProperty, const AAX_SPlugInIdentifierTriad *inPluginIDs, uint32_t inNumPluginIDs\)=0](#)

Add an array of plug-in IDs to a property map.

- virtual `AAX_CBoolean GetPropertyWithIDArray (AAX_EProperty inProperty, const AAX_SPlugInIdentifierTriad **outPluginIDs, uint32_t *outNumPluginIDs) const =0`

Get an array of plug-in IDs from a property map.

14.73.2 Member Function Documentation

14.73.2.1 virtual AAX_Result AAX_IACFPropertyMap_V2::AddPropertyWithIDArray (`AAX_EProperty inProperty, const AAX_SPlugInIdentifierTriad *inPluginIDs, uint32_t inNumPluginIDs`) [pure virtual]

Add an array of plug-in IDs to a property map.

Parameters

in	<code>inProperty</code>	The property ID.
in	<code>inPluginIDs</code>	An array of <code>AAX_SPlugInIdentifierTriad</code>
in	<code>inNumPluginIDs</code>	The length of <code>inPluginIDs</code>

14.73.2.2 virtual AAX_CBoolean AAX_IACFPropertyMap_V2::GetPropertyWithIDArray (`AAX_EProperty inProperty, const AAX_SPlugInIdentifierTriad **outPluginIDs, uint32_t *outNumPluginIDs`) const [pure virtual]

Get an array of plug-in IDs from a property map.

Parameters

in	<code>inProperty</code>	The property ID.
out	<code>outPluginIDs</code>	A pointer that will be set to reference an array of <code>AAX_SPlugInIdentifierTriad</code>
in	<code>outNumPluginIDs</code>	The length of <code>outPluginIDs</code>

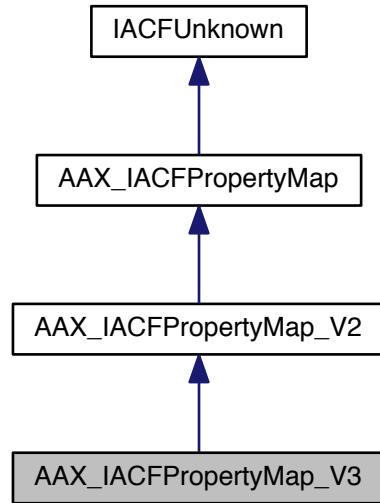
The documentation for this class was generated from the following file:

- `AAX_IACFPropertyMap.h`

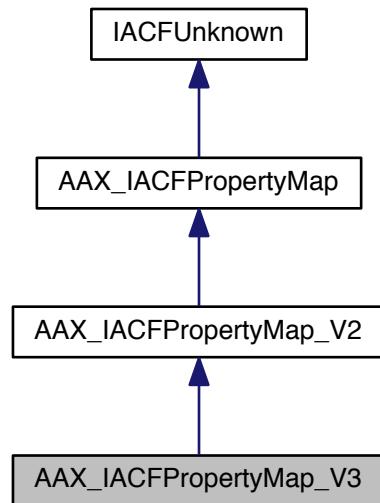
14.74 AAX_IACFPropertyMap_V3 Class Reference

```
#include <AAX_IACFPropertyMap.h>
```

Inheritance diagram for AAX_IACFPropertyMap_V3:



Collaboration diagram for AAX_IACFPropertyMap_V3:



14.74.1 Description

Versioned interface for an [AAX_IPropertyMap](#).

Public Member Functions

- virtual [AAX_CBoolean GetProperty64 \(AAX_EProperty inProperty, AAX_CPropertyValue64 *outValue\) const =0](#)

Get a property value from a property map.

- virtual [AAX_Result AddProperty64 \(AAX_EProperty inProperty, AAX_CPropertyValue64 inValue\)=0](#)

Add a property to a property map.

14.74.2 Member Function Documentation

14.74.2.1 virtual AAX_CBoolean AAX_IACFPropertyMap_V3::GetProperty64 (AAX_EProperty *inProperty*, AAX_CPropertyValue64 * *outValue*) const [pure virtual]

Get a property value from a property map.

Returns true if the selected property is supported, false if it is not

Parameters

in	<i>inProperty</i>	The property ID
out	<i>outValue</i>	The property value

14.74.2.2 virtual AAX_Result AAX_IACFPropertyMap_V3::AddProperty64 (AAX_EProperty *inProperty*, AAX_CPropertyValue64 *inValue*) [pure virtual]

Add a property to a property map.

Note

This method may return an error if adding the property was unsuccessful. If there is a failure when adding a required property then registration of the relevant description element must be abandoned and the plug-in's description logic should proceed to the next element.

Parameters

in	<i>inProperty</i>	The property ID.
in	<i>inValue</i>	

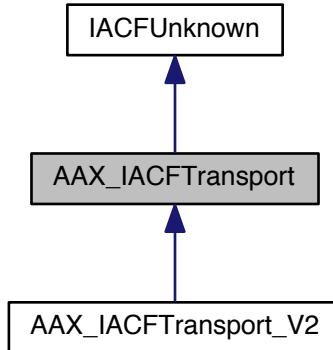
The documentation for this class was generated from the following file:

- [AAX_IACFPropertyMap.h](#)

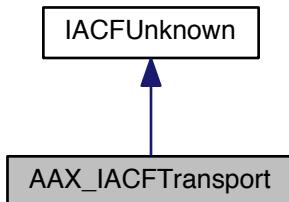
14.75 AAX_IACFTransport Class Reference

```
#include <AAX_IACFTransport.h>
```

Inheritance diagram for AAX_IACFTransport:



Collaboration diagram for AAX_IACFTransport:



14.75.1 Description

Versioned interface to information about the host's transport state.

Public Member Functions

- virtual [AAX_Result GetCurrentTempo](#) (double *TempoBPM) const =0
CALL: Gets the current tempo.
- virtual [AAX_Result GetCurrentMeter](#) (int32_t *MeterNumerator, int32_t *MeterDenominator) const =0
CALL: Gets the current meter.
- virtual [AAX_Result IsTransportPlaying](#) (bool *isPlaying) const =0
CALL: Indicates whether or not the transport is playing back.
- virtual [AAX_Result GetCurrentTickPosition](#) (int64_t *TickPosition) const =0
CALL: Gets the current tick position.
- virtual [AAX_Result GetCurrentLoopPosition](#) (bool *bLooping, int64_t *LoopStartTick, int64_t *LoopEndTick) const =0

CALL: Gets current information on loop playback.

- virtual [AAX_Result GetCurrentNativeSampleLocation](#) (int64_t *SampleLocation) const =0

CALL: Gets the current playback location of the native audio engine.

- virtual [AAX_Result GetCustomTickPosition](#) (int64_t *oTickPosition, int64_t iSampleLocation) const =0

CALL: Given an absolute sample position, gets the corresponding tick position.

- virtual [AAX_Result GetBarBeatPosition](#) (int32_t *Bars, int32_t *Beats, int64_t *DisplayTicks, int64_t SampleLocation) const =0

CALL: Given an absolute sample position, gets the corresponding bar and beat position.

- virtual [AAX_Result GetTicksPerQuarter](#) (uint32_t *ticks) const =0

CALL: Retrieves the number of ticks per quarter note.

- virtual [AAX_Result GetCurrentTicksPerBeat](#) (uint32_t *ticks) const =0

CALL: Retrieves the number of ticks per beat.

14.75.2 Member Function Documentation

14.75.2.1 virtual [AAX_Result AAX_IACFTransport::GetCurrentTempo](#) (double * *TempoBPM*) const [pure virtual]

CALL: Gets the current tempo.

Returns the tempo corresponding to the current position of the transport counter

Note

The resolution of the tempo returned here is based on the host's tempo resolution, so it will match the tempo displayed in the host. Use [GetCurrentTicksPerBeat\(\)](#) to calculate the tempo resolution note.

Parameters

out	<i>TempoBPM</i>	The current tempo in beats per minute
-----	-----------------	---------------------------------------

14.75.2.2 virtual [AAX_Result AAX_IACFTransport::GetCurrentMeter](#) (int32_t * *MeterNumerator*, int32_t * *MeterDenominator*) const [pure virtual]

CALL: Gets the current meter.

Returns the meter corresponding to the current position of the transport counter

Parameters

out	<i>MeterNumerator</i>	The numerator portion of the meter
out	<i>MeterDenominator</i>	The denominator portion of the meter

14.75.2.3 virtual [AAX_Result AAX_IACFTransport::IsTransportPlaying](#) (bool * *isPlaying*) const [pure virtual]

CALL: Indicates whether or not the transport is playing back.

Parameters

out	<i>isPlaying</i>	true if the transport is currently in playback
-----	------------------	--

```
14.75.2.4 virtual AAX_Result AAX_IACFTransport::GetCurrentTickPosition( int64_t * TickPosition ) const [pure virtual]
```

CALL: Gets the current tick position.

Returns the current tick position corresponding to the current transport position. One "Tick" is represented here as 1/960000 of a quarter note. That is, there are 960,000 of these ticks in a quarter note.

Host Compatibility Notes The tick resolution here is different than that of the tick displayed in Pro Tools. "Display ticks" (as they are called) are 1/960 of a quarter note.

Parameters

out	<i>TickPosition</i>	The tick position value
-----	---------------------	-------------------------

```
14.75.2.5 virtual AAX_Result AAX_IACFTransport::GetCurrentLoopPosition( bool * bLooping, int64_t * LoopStartTick, int64_t * LoopEndTick ) const [pure virtual]
```

CALL: Gets current information on loop playback.

Host Compatibility Notes This does not indicate anything about the status of the "Loop Record" option. Even when the host is configured to loop playback, looping may not occur if certain conditions are not met (i.e. the length of the selection is too short)

Parameters

out	<i>bLooping</i>	true if the host is configured to loop playback
out	<i>LoopStartTick</i>	The starting tick position of the selection being looped (see GetCurrentTickPosition())
out	<i>LoopEndTick</i>	The ending tick position of the selection being looped (see GetCurrentTickPosition())

```
14.75.2.6 virtual AAX_Result AAX_IACFTransport::GetCurrentNativeSampleLocation( int64_t * SampleLocation ) const [pure virtual]
```

CALL: Gets the current playback location of the native audio engine.

When called from a ProcessProc render callback, this method will provide the absolute sample location at the beginning of the callback's audio buffers.

When called from [AAX_IEffectParameters::RenderAudio_Hybrid\(\)](#), this method will provide the absolute sample location for the samples in the method's **output** audio buffers. To calculate the absolute sample location for the samples in the method's input buffers (i.e. the timeline location where the samples originated) subtract the value provided by [AAX_IController::GetHybridSignalLatency\(\)](#) from this value.

When called from a non-real-time thread, this method will provide the current location of the samples being processed by the plug-in's ProcessProc on its real-time processing thread.

Note

This method only returns a value during playback. It cannot be used to determine, e.g., the location of the timeline selector while the host is not in playback.

Parameters

out	<i>SampleLocation</i>	Absolute sample location of the first sample in the current native processing buffer
-----	-----------------------	--

14.75.2.7 virtual AAX_Result AAX_IACFTransport::GetCustomTickPosition (*int64_t* * *oTickPosition*, *int64_t* *iSampleLocation*) const [pure virtual]

CALL: Given an absolute sample position, gets the corresponding tick position.

Host Compatibility Notes There is a minor performance cost associated with using this API in Pro Tools. It should not be used excessively without need.

Parameters

out	<i>oTickPosition</i>	the timeline tick position corresponding to <i>iSampleLocation</i>
in	<i>iSampleLocation</i>	An absolute sample location (see GetCurrentNativeSampleLocation())

14.75.2.8 virtual AAX_Result AAX_IACFTransport::GetBarBeatPosition (*int32_t* * *Bars*, *int32_t* * *Beats*, *int64_t* * *DisplayTicks*, *int64_t* *SampleLocation*) const [pure virtual]

CALL: Given an absolute sample position, gets the corresponding bar and beat position.

Host Compatibility Notes There is a minor performance cost associated with using this API in Pro Tools. It should not be used excessively without need.

Parameters

out	<i>Bars</i>	The bar corresponding to <i>SampleLocation</i>
out	<i>Beats</i>	The beat corresponding to <i>SampleLocation</i>
out	<i>DisplayTicks</i>	The ticks corresponding to <i>SampleLocation</i>
in	<i>SampleLocation</i>	An absolute sample location (see GetCurrentNativeSampleLocation())

14.75.2.9 virtual AAX_Result AAX_IACFTransport::GetTicksPerQuarter (*uint32_t* * *ticks*) const [pure virtual]

CALL: Retrieves the number of ticks per quarter note.

Parameters

out	<i>ticks</i>	
-----	--------------	--

14.75.2.10 virtual AAX_Result AAX_IACFTransport::GetCurrentTicksPerBeat (*uint32_t* * *ticks*) const [pure virtual]

CALL: Retrieves the number of ticks per beat.

Parameters

out	<i>ticks</i>	
-----	--------------	--

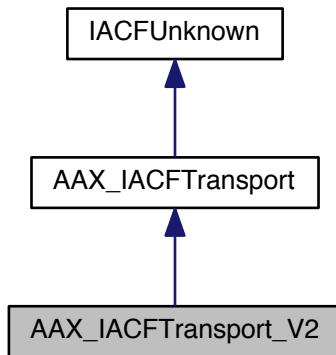
The documentation for this class was generated from the following file:

- [AAX_IACFTransport.h](#)

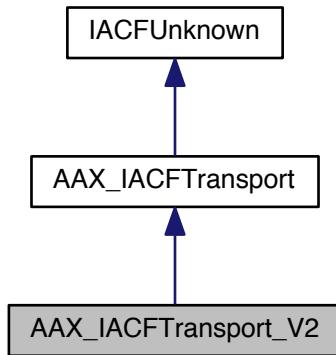
14.76 AAX_IACFTransport_V2 Class Reference

```
#include <AAX_IACFTransport.h>
```

Inheritance diagram for AAX_IACFTransport_V2:



Collaboration diagram for AAX_IACFTransport_V2:



14.76.1 Description

Versioned interface to information about the host's transport state.

Public Member Functions

- virtual [AAX_Result GetTimelineSelectionStartPosition](#) (int64_t *oSampleLocation) const =0
CALL: Retrieves the current absolute sample position of the beginning of the current transport selection.
- virtual [AAX_Result GetTimeCodeInfo](#) ([AAX_EFrameRate](#) *oFrameRate, int32_t *oOffset) const =0

CALL: Retrieves the current time code frame rate and offset.

- virtual `AAX_Result GetFeetFramesInfo (AAX_EFootFramesRate *oFootFramesRate, int64_t *oOffset) const =0`

CALL: Retrieves the current timecode feet/frames rate and offset.

- virtual `AAX_Result IsMetronomeEnabled (int32_t *isEnabled) const =0`

Sets isEnabled to true if the metronome is enabled.

14.76.2 Member Function Documentation

14.76.2.1 virtual AAX_Result AAX_IACFTransport_V2::GetTimelineSelectionStartPosition (int64_t * oSampleLocation) const [pure virtual]

CALL: Retrieves the current absolute sample position of the beginning of the current transport selection.

Note

This method is part of the [version 2 transport interface](#)

Parameters

out	<i>oSample← Location</i>	
-----	------------------------------	--

14.76.2.2 virtual AAX_Result AAX_IACFTransport_V2::GetTimeCodeInfo (AAX_EFrameRate * oFrameRate, int32_t * oOffset) const [pure virtual]

CALL: Retrieves the current time code frame rate and offset.

Note

This method is part of the [version 2 transport interface](#)

Parameters

out	<i>oFrameRate</i>	
out	<i>oOffset</i>	

14.76.2.3 virtual AAX_Result AAX_IACFTransport_V2::GetFeetFramesInfo (AAX_EFootFramesRate * oFootFramesRate, int64_t * oOffset) const [pure virtual]

CALL: Retrieves the current timecode feet/frames rate and offset.

Note

This method is part of the [version 2 transport interface](#)

Parameters

out	<i>oFootFrames← Rate</i>	
-----	------------------------------	--

out	<i>oOffset</i>	
-----	----------------	--

14.76.2.4 virtual AAX_Result AAX_IACFTransport_V2::IsMetronomeEnabled (int32_t * *isEnabled*) const [pure virtual]

Sets *isEnabled* to true if the metronome is enabled.

Note

This method is part of the [version 2 transport interface](#)

Parameters

out	<i>isEnabled</i>	
-----	------------------	--

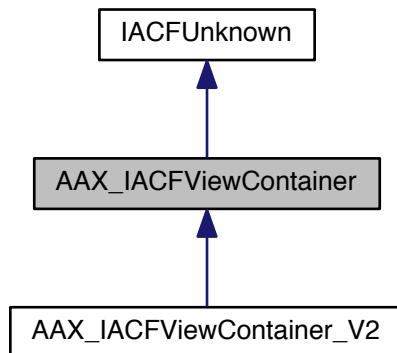
The documentation for this class was generated from the following file:

- [AAX_IACFTransport.h](#)

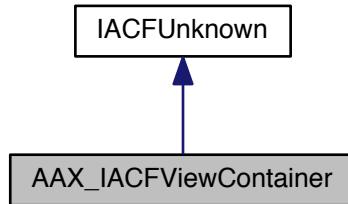
14.77 AAX_IACFViewContainer Class Reference

```
#include <AAX_IACFViewContainer.h>
```

Inheritance diagram for AAX_IACFViewContainer:



Collaboration diagram for AAX_IACFViewContainer:



14.77.1 Description

Interface for the AAX host's view of a single instance of an effect. Used by both clients of the host app and by effect components.

See also

[AAX_IViewContainer](#)

Public Member Functions

View and GUI state queries

- virtual int32_t [GetType \(\)=0](#)
Returns the raw view type as one of [AAX_EViewContainer_Type](#).
- virtual void * [GetPtr \(\)=0](#)
Returns a pointer to the raw view.
- virtual [AAX_Result GetModifiers \(uint32_t *outModifiers\)=0](#)
Queries the host for the current [modifier keys](#).

View change requests

- virtual [AAX_Result SetViewSize \(AAX_Point &inSize\)=0](#)
Request a change to the main view size.

Host event handlers

- virtual [AAX_Result HandleParameterMouseDown \(AAX_CParamID inParamID, uint32_t inModifiers\)=0](#)
Alert the host to a mouse down event.
- virtual [AAX_Result HandleParameterMouseDrag \(AAX_CParamID inParamID, uint32_t inModifiers\)=0](#)
Alert the host to a mouse drag event.
- virtual [AAX_Result HandleParameterMouseUp \(AAX_CParamID inParamID, uint32_t inModifiers\)=0](#)
Alert the host to a mouse up event.

14.77.2 Member Function Documentation

14.77.2.1 virtual int32_t AAX_IACFViewContainer::GetType () [pure virtual]

Returns the raw view type as one of [AAX_EViewContainer_Type](#).

14.77.2.2 virtual void* AAX_IACFViewContainer::GetPtr() [pure virtual]

Returns a pointer to the raw view.

14.77.2.3 virtual AAX_Result AAX_IACFViewContainer::GetModifiers(uint32_t * *outModifiers*) [pure virtual]

Queries the host for the current modifier keys.

This method returns a bit mask with bits set for each of the currently active modifier keys. This method does not return the state of the [AAX_eModifiers_SecondaryButton](#).

Host Compatibility Notes Although this method allows plug-ins to acquire the current state of the Windows key (normally blocked by Pro Tools), plug-ins should not use key combinations that require this key.

Parameters

out	<i>outModifiers</i>	Current modifiers as a bitmask of AAX_EModifiers
-----	---------------------	--

14.77.2.4 virtual AAX_Result AAX_IACFViewContainer::SetViewSize(AAX_Point & *inSize*) [pure virtual]

Request a change to the main view size.

Note

- For compatibility with the smallest supported displays, plug-in GUI dimensions should not exceed 749x617 pixels, or 749x565 pixels for plug-ins with sidechain support.

Parameters

in	<i>inSize</i>	The new size to which the plug-in view should be set
----	---------------	--

14.77.2.5 virtual AAX_Result AAX_IACFViewContainer::HandleParameterMouseDown(AAX_CParamID *inParamID*, uint32_t *inModifiers*) [pure virtual]

Alert the host to a mouse down event.

Parameters

in	<i>inParamID</i>	ID of the parameter whose control is being edited
in	<i>inModifiers</i>	A bitmask of AAX_EModifiers values

14.77.2.6 virtual AAX_Result AAX_IACFViewContainer::HandleParameterMouseDrag(AAX_CParamID *inParamID*, uint32_t *inModifiers*) [pure virtual]

Alert the host to a mouse drag event.

Warning

The host may return [AAX_ERROR_UNIMPLEMENTED](#) for this event even if the host did handle the corresponding mouse down event. A plug-in should ignore any following mouse drag and mouse up events that correspond to a host-managed mouse down event. ([PTSW-195209](#) / [PT-218474](#))

Parameters

in	<i>inParamID</i>	ID of the parameter whose control is being edited
in	<i>inModifiers</i>	A bitmask of AAX_EModifiers values

14.77.2.7 virtual **AAX_Result** **AAX_IACFViewContainer::HandleParameterMouseUp** (**AAX_CParamID** *inParamID*, **uint32_t** *inModifiers*) [pure virtual]

Alert the host to a mouse up event.

Warning

The host may return [AAX_ERROR_UNIMPLEMENTED](#) for this event even if the host did handle the corresponding mouse down event. A plug-in should ignore any following mouse drag and mouse up events that correspond to a host-managed mouse down event. ([PTSW-195209 / PT-218474](#))

Parameters

in	<i>inParamID</i>	ID of the parameter whose control is being edited
in	<i>inModifiers</i>	A bitmask of AAX_EModifiers values

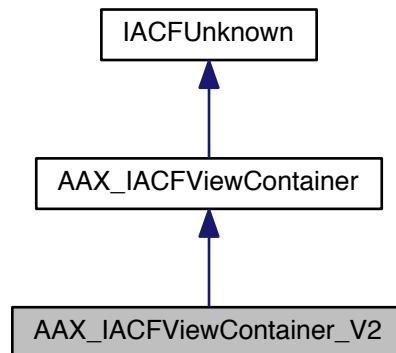
The documentation for this class was generated from the following file:

- [AAX_IACFViewContainer.h](#)

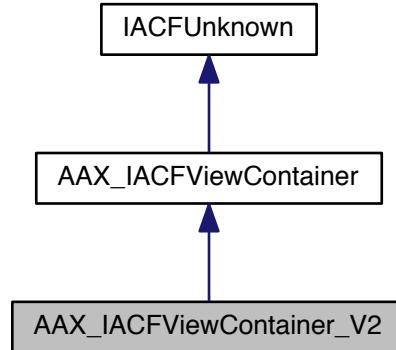
14.78 AAX_IACFViewContainer_V2 Class Reference

```
#include <AAX_IACFViewContainer.h>
```

Inheritance diagram for AAX_IACFViewContainer_V2:



Collaboration diagram for AAX_IACFViewContainer_V2:



14.78.1 Description

Supplemental interface for the AAX host's view of a single instance of an effect. Used by both clients of the host app and by effect components.

See also

[AAX_IViewContainer](#)

Public Member Functions

Host event handlers

- virtual `AAX_Result HandleMultipleParametersMouseDown (const AAX_CParamID *inParamIDs, uint32_t inNumOfParams, uint32_t inModifiers)=0`
Alert the host to a mouse down event.
- virtual `AAX_Result HandleMultipleParametersMouseDrag (const AAX_CParamID *inParamIDs, uint32_t inNumOfParams, uint32_t inModifiers)=0`
Alert the host to a mouse drag event.
- virtual `AAX_Result HandleMultipleParametersMouseUp (const AAX_CParamID *inParamIDs, uint32_t inNumOfParams, uint32_t inModifiers)=0`
Alert the host to a mouse up event.

14.78.2 Member Function Documentation

14.78.2.1 virtual `AAX_Result AAX_IACFViewContainer_V2::HandleMultipleParametersMouseDown (const AAX_CParamID * inParamIDs, uint32_t inNumOfParams, uint32_t inModifiers) [pure virtual]`

Alert the host to a mouse down event.

Parameters

in	<i>inParamIDs</i>	IDs of the parameters that belong to the same GUI element whose controls are being edited
in	<i>inNumParams</i>	Number of parameter IDs
in	<i>inModifiers</i>	A bitmask of AAX_EModifiers values

14.78.2.2 virtual AAX_Result AAX_IACFViewContainer_V2::HandleMultipleParametersMouseDrag (const AAX_CParamID * *inParamIDs*, uint32_t *inNumParams*, uint32_t *inModifiers*) [pure virtual]

Alert the host to a mouse drag event.

Warning

The host may return [AAX_ERROR_UNIMPLEMENTED](#) for this event even if the host did handle the corresponding mouse down event. A plug-in should ignore any following mouse drag and mouse up events that correspond to a host-managed mouse down event. ([PTSW-195209](#) / [PT-218474](#))

Parameters

in	<i>inParamIDs</i>	IDs of the parameters that belong to the same GUI element whose controls are being edited
in	<i>inNumParams</i>	Number of parameter IDs
in	<i>inModifiers</i>	A bitmask of AAX_EModifiers values

14.78.2.3 virtual AAX_Result AAX_IACFViewContainer_V2::HandleMultipleParametersMouseUp (const AAX_CParamID * *inParamIDs*, uint32_t *inNumParams*, uint32_t *inModifiers*) [pure virtual]

Alert the host to a mouse up event.

Warning

The host may return [AAX_ERROR_UNIMPLEMENTED](#) for this event even if the host did handle the corresponding mouse down event. A plug-in should ignore any following mouse drag and mouse up events that correspond to a host-managed mouse down event. ([PTSW-195209](#) / [PT-218474](#))

Parameters

in	<i>inParamIDs</i>	IDs of the parameters that belong to the same GUI element whose controls are being edited
in	<i>inNumParams</i>	Number of parameter IDs
in	<i>inModifiers</i>	A bitmask of AAX_EModifiers values

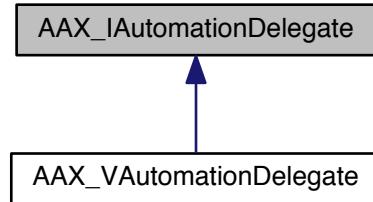
The documentation for this class was generated from the following file:

- [AAX_IACFViewContainer.h](#)

14.79 AAX_IAutomationDelegate Class Reference

```
#include <AAX_IAutomationDelegate.h>
```

Inheritance diagram for AAX_IAutomationDelegate:



14.79.1 Description

Interface allowing an AAX plug-in to interact with the host's event system.

:Implemented by the AAX Host

This delegate provides a means of interacting with the host's event system in order to ensure that events such as parameter updates are properly arbitrated and broadcast to all listeners. The automation delegate is used regardless of whether an individual parameter is "automatable" or "automation-enabled".

A parameter must be registered with the automation delegate in order for updates to the parameter's control in the plug-in's GUI or other controller (control surface, etc.) to be successfully processed by the host and sent to the [AAX_IEffectParameters](#) object.

The parameter identifiers used by this interface correspond to the control IDs used to identify parameters in the [Parameter Manager](#).

Public Member Functions

- virtual [~AAX_IAutomationDelegate \(\)](#)
- virtual [AAX_Result RegisterParameter \(AAX_CParamID iParameterID\)=0](#)
- virtual [AAX_Result UnregisterParameter \(AAX_CParamID iParameterID\)=0](#)
- virtual [AAX_Result PostSetValueRequest \(AAX_CParamID iParameterID, double normalizedValue\) const =0](#)
- virtual [AAX_Result PostCurrentValue \(AAX_CParamID iParameterID, double normalizedValue\) const =0](#)
- virtual [AAX_Result PostTouchRequest \(AAX_CParamID iParameterID\)=0](#)
- virtual [AAX_Result PostReleaseRequest \(AAX_CParamID iParameterID\)=0](#)
- virtual [AAX_Result GetTouchState \(AAX_CParamID iParameterID, AAX_CBoolean *oTouched\)=0](#)

14.79.2 Constructor & Destructor Documentation

14.79.2.1 virtual AAX_IAutomationDelegate::[~AAX_IAutomationDelegate\(\)](#) [inline], [virtual]

14.79.3 Member Function Documentation

14.79.3.1 virtual [AAX_Result AAX_IAutomationDelegate::RegisterParameter \(AAX_CParamID iParameterID \)](#) [pure virtual]

Register a control with the automation system using a char* based control identifier

The automation delegate owns a list of the IDs of all of the parameters that have been registered with it. This list is used to set up listeners for all of the registered parameters such that the automation delegate may update the plug-in when the state of any of the registered parameters have been modified.

See also

[AAX_IAutomationDelegate::UnregisterParameter\(\)](#)

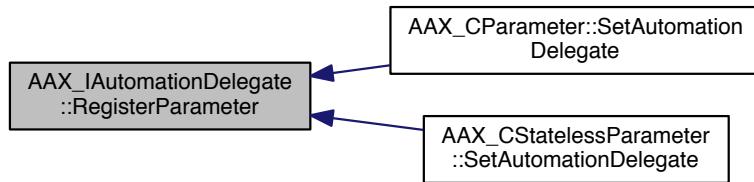
Parameters

in	<i>iParameterID</i>	Parameter ID that is being registered
----	---------------------	---------------------------------------

Implemented in [AAX_VAutomationDelegate](#).

Referenced by [AAX_CParameter< T >::SetAutomationDelegate\(\)](#), and [AAX_CStatelessParameter::SetAutomationDelegate\(\)](#).

Here is the caller graph for this function:



14.79.3.2 virtual AAX_Result AAX_IAutomationDelegate::UnregisterParameter (AAX_CParamID iParameterID) [pure virtual]

Unregister a control with the automation system using a char* based control identifier

Note

All registered controls should be unregistered or the system might leak.

See also

[AAX_IAutomationDelegate::RegisterParameter\(\)](#)

Parameters

in	<i>iParameterID</i>	Parameter ID that is being registered
----	---------------------	---------------------------------------

Implemented in [AAX_VAutomationDelegate](#).

Referenced by [AAX_CStatelessParameter::SetAutomationDelegate\(\)](#).

Here is the caller graph for this function:



14.79.3.3 virtual AAX_Result AAX_IAutomationDelegate::PostSetValueRequest (AAX_CParamID *iParameterID*, double *normalizedValue*) const [pure virtual]

Submits a request for the given parameter's value to be changed

Parameters

in	<i>iParameterID</i>	ID of the parameter for which a change is requested
in	<i>normalizedValue</i>	The requested new parameter value, formatted as a double and normalized to [0 1]

Implemented in [AAX_VAutomationDelegate](#).

14.79.3.4 virtual AAX_Result AAX_IAutomationDelegate::PostCurrentValue (AAX_CParamID *iParameterID*, double *normalizedValue*) const [pure virtual]

Notifies listeners that a parameter's value has changed

Parameters

in	<i>iParameterID</i>	ID of the parameter that has been updated
in	<i>normalizedValue</i>	The current parameter value, formatted as a double and normalized to [0 1]

Implemented in [AAX_VAutomationDelegate](#).

14.79.3.5 virtual AAX_Result AAX_IAutomationDelegate::PostTouchRequest (AAX_CParamID *iParameterID*) [pure virtual]

Requests that the given parameter be "touched", i.e. locked for updates by the current client

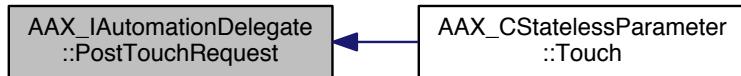
Parameters

in	<i>iParameterID</i>	ID of the parameter that will be touched
----	---------------------	--

Implemented in [AAX_VAutomationDelegate](#).

Referenced by [AAX_CStatelessParameter::Touch\(\)](#).

Here is the caller graph for this function:



14.79.3.6 virtual AAX_Result AAX_IAutomationDelegate::PostReleaseRequest (AAX_CParamID *iParameterID*) [pure virtual]

Requests that the given parameter be "released", i.e. available for updates from any client

Parameters

in	<i>iParameterID</i>	ID of the parameter that will be released
----	---------------------	---

Implemented in [AAX_VAutomationDelegate](#).

Referenced by [AAX_CStatelessParameter::Release\(\)](#).

Here is the caller graph for this function:



14.79.3.7 virtual AAX_Result AAX_IAutomationDelegate::GetTouchState (AAX_CParamID *iParameterID*, AAX_CBoolean * *oTouched*) [pure virtual]

Gets the current touched state of a parameter

Parameters

in	<i>iParameterID</i>	ID of the parameter that is being queried
out	<i>oTouched</i>	The current touch state of the parameter

Implemented in [AAX_VAutomationDelegate](#).

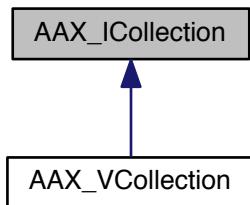
The documentation for this class was generated from the following file:

- [AAX_IAutomationDelegate.h](#)

14.80 AAX_ICollection Class Reference

```
#include <AAX_ICollection.h>
```

Inheritance diagram for AAX_ICollection:



14.80.1 Description

Interface to represent a plug-in binary's static description.

:Implemented by the AAX Host

The [AAX_ICollection](#) interface provides a creation function for new plug-in descriptors, which in turn provides access to the various interfaces necessary for describing a plug-in. When a plug-in description is complete, it is added to the collection via the [AddEffect](#) method. The [AAX_ICollection](#) interface also provides some additional description methods that are used to describe the overall plug-in package. These methods can be used to describe the plug-in package's name, the name of the plug-in's manufacturer, and the plug-in package version.

Legacy Porting Notes The information in [AAX_ICollection](#) is roughly analogous to the information provided by CProcessGroup in the legacy plug-in library

Public Member Functions

- virtual ~[AAX_ICollection](#) ()
- virtual [AAX_IEffectDescriptor](#) * [NewDescriptor](#) ()=0

Create a new Effect descriptor.
- virtual [AAX_Result](#) [AddEffect](#) (const char *inEffectID, [AAX_IEffectDescriptor](#) *inEffectDescriptor)=0

Add an Effect description to the collection.
- virtual [AAX_Result](#) [SetManufacturerName](#) (const char *inPackageName)=0

Set the plug-in manufacturer name.
- virtual [AAX_Result](#) [AddPackageName](#) (const char *inPackageName)=0

Set the plug-in package name.
- virtual [AAX_Result](#) [SetPackageVersion](#) (uint32_t inVersion)=0

Set the plug-in package version number.
- virtual [AAX_IPropertyMap](#) * [NewPropertyMap](#) ()=0

Create a new property map.
- virtual [AAX_Result](#) [SetProperties](#) ([AAX_IPropertyMap](#) *inProperties)=0

Set the properties of the collection.
- virtual [AAX_IDescriptionHost](#) * [DescriptionHost](#) ()=0
- virtual const [AAX_IDescriptionHost](#) * [DescriptionHost](#) () const =0
- virtual [IACFDefinition](#) * [HostDefinition](#) () const =0

14.80.2 Constructor & Destructor Documentation

14.80.2.1 virtual AAX_ICollection::~AAX_ICollection() [inline], [virtual]

14.80.3 Member Function Documentation

14.80.3.1 virtual AAX_IEffectDescriptor* AAX_ICollection::NewDescriptor() [pure virtual]

Create a new Effect descriptor.

Implemented in [AAX_VCollection](#).

14.80.3.2 virtual AAX_Result AAX_ICollection::AddEffect(const char * *inEffectID*, AAX_IEffectDescriptor * *inEffectDescriptor*) [pure virtual]

Add an Effect description to the collection.

Each Effect that a plug-in registers with [AAX_ICollection::AddEffect\(\)](#) is considered a completely different user-facing product. For example, in Avid's Dynamics III plug-in the Expander, Compressor, and DeEsser are each registered as separate Effects. All stem format variations within each Effect are registered within that Effect's [AA←X_IEffectDescriptor](#) using [AddComponent\(\)](#).

The [AAX_eProperty_ProductID](#) value for all ProcessProcs within a single Effect must be identical.

This method passes ownership of an [AAX_IEffectDescriptor](#) object to the [AAX_ICollection](#). The [AAX_IEffectDescriptor](#) must not be deleted by the AAX plug-in, nor should it be edited in any way after it is passed to the [AAX_ICollection](#).

Parameters

in	<i>inEffectID</i>	The effect ID.
in	<i>inEffectDescriptor</i>	The Effect descriptor.

Implemented in [AAX_VCollection](#).

14.80.3.3 virtual AAX_Result AAX_ICollection::SetManufacturerName(const char * *inPackageName*) [pure virtual]

Set the plug-in manufacturer name.

Parameters

in	<i>inPackageName</i>	The name of the manufacturer.
----	----------------------	-------------------------------

Implemented in [AAX_VCollection](#).

14.80.3.4 virtual AAX_Result AAX_ICollection::AddPackageName(const char * *inPackageName*) [pure virtual]

Set the plug-in package name.

May be called multiple times to add abbreviated package names.

Note

Every plug-in must include at least one name variant with 16 or fewer characters, plus a null terminating character. Used for Presets folder.

Parameters

in	<i>inPackageName</i>	The name of the package.
----	----------------------	--------------------------

Implemented in [AAX_VCollection](#).

14.80.3.5 virtual AAX_Result AAX_ICollection::SetPackageVersion(uint32_t *inVersion*) [pure virtual]

Set the plug-in package version number.

Parameters

in	<i>inVersion</i>	The package version numner.
----	------------------	-----------------------------

Implemented in [AAX_VCollection](#).

14.80.3.6 virtual AAX_IPropertyMap* AAX_ICollection::NewPropertyMap() [pure virtual]

Create a new property map.

Implemented in [AAX_VCollection](#).

14.80.3.7 virtual AAX_Result AAX_ICollection::SetProperties(AAX_IPropertyMap * *inProperties*) [pure virtual]

Set the properties of the collection.

Parameters

in	<i>inProperties</i>	Collection properties
----	---------------------	-----------------------

Implemented in [AAX_VCollection](#).

14.80.3.8 virtual AAX_IDescriptionHost* AAX_ICollection::DescriptionHost() [pure virtual]

Get a pointer to an [AAX_IDescriptionHost](#), if supported by the host

This interface is served by the [AAX_ICollection](#) in order to avoid requiring a new method prototype for the [GetEffectDescriptions\(\)](#) method called from the AAX Library.

See also

[AAX_UIDs.h](#) for available feature UIDs, e.g. [AAXATTR_ClientFeature_AuxOutputStem](#)

Implemented in [AAX_VCollection](#).

14.80.3.9 virtual const AAX_IDescriptionHost* AAX_ICollection::DescriptionHost() const [pure virtual]

Get a pointer to an [AAX_IDescriptionHost](#), if supported by the host

This interface is served by the [AAX_ICollection](#) in order to avoid requiring a new method prototype for the [GetEffectDescriptions\(\)](#) method called from the AAX Library.

See also

[AAX_UIDs.h](#) for available feature UIDs, e.g. [AAXATTR_ClientFeature_AuxOutputStem](#)

Implemented in [AAX_VCollection](#).

14.80.3.10 virtual IACFDefinition* AAX_ICollection::HostDefinition() const [pure virtual]

Get a pointer to an [IACFDefinition](#), if supported by the host

This interface is served by the [AAX_ICollection](#) in order to avoid requiring a new method prototype for the [GetEffectDescriptions\(\)](#) method called from the AAX Library.

See also

[AAX_UIDs.h](#) for available host attribute UIDs, e.g. [AAXATTR_Client_Level](#)

The implementation of [AAX_ICollection](#) owns the referenced object. No AddRef occurs.

[IACFDefinition::DefineAttribute\(\)](#) is not supported on this object

Implemented in [AAX_VCollection](#).

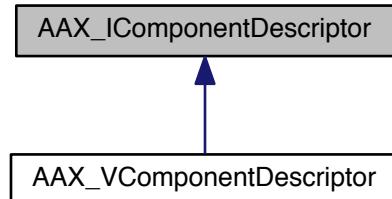
The documentation for this class was generated from the following file:

- [AAX_ICollection.h](#)

14.81 AAX_IComponentDescriptor Class Reference

```
#include <AAX_IComponentDescriptor.h>
```

Inheritance diagram for AAX_IComponentDescriptor:



14.81.1 Description

Description interface for an AAX plug-in component.

:Implemented by the AAX Host

This is an abstract interface containing everything needed to describe a single algorithm of an Effect. For more information about algorithm processing in AAX plug-ins, see [Real-time algorithm callback](#).

Public Member Functions

- virtual [~AAX_IComponentDescriptor\(\)](#)
- virtual [AAX_Result Clear\(\)=0](#)
Clears the descriptor.
- virtual [AAX_Result AddAudioIn \(AAX_CFieldIndex inFieldIndex\)=0](#)

- *Subscribes an audio input context field.*
- virtual `AAX_Result AddAudioOut (AAX_CFieldIndex inFieldIndex)=0`
Subscribes an audio output context field.
- virtual `AAX_Result AddAudioBufferLength (AAX_CFieldIndex inFieldIndex)=0`
Subscribes a buffer length context field.
- virtual `AAX_Result AddSampleRate (AAX_CFieldIndex inFieldIndex)=0`
Subscribes a sample rate context field.
- virtual `AAX_Result AddClock (AAX_CFieldIndex inFieldIndex)=0`
Subscribes a clock context field.
- virtual `AAX_Result AddSideChainIn (AAX_CFieldIndex inFieldIndex)=0`
Subscribes a side-chain input context field.
- virtual `AAX_Result AddDataInPort (AAX_CFieldIndex inFieldIndex, uint32_t inPacketSize, AAX_EDataInPortType inPortType=AAX_eDataInPortType_Buffered)=0`
Adds a custom data port to the algorithm context.
- virtual `AAX_Result AddAuxOutputStem (AAX_CFieldIndex inFieldIndex, int32_t inStemFormat, const char inNameUTF8[])=0`
Adds an auxiliary output stem for a plug-in.
- virtual `AAX_Result AddPrivateData (AAX_CFieldIndex inFieldIndex, int32_t inDataSize, uint32_t inOptions=AAX_ePrivateDataOptions_DefaultOptions)=0`
Adds a private data port to the algorithm context.
- virtual `AAX_Result AddTemporaryData (AAX_CFieldIndex inFieldIndex, uint32_t inDataElementSize)=0`
Adds a block of data to a context that is not saved between callbacks and is scaled by the system buffer size.
- virtual `AAX_Result AddDmaInstance (AAX_CFieldIndex inFieldIndex, AAX_IDma::EMode inDmaMode)=0`
Adds a DMA field to the plug-in's context.
- virtual `AAX_Result AddMIDI (AAX_CFieldIndex inFieldIndex, const AAX_CTypeID *inMeterIDs, const uint32_t inMeterCount)=0`
Adds a meter field to the plug-in's context.
- virtual `AAX_Result AddMIDINode (AAX_CFieldIndex inFieldIndex, AAX_EMIDINodeType inNodeType, const char inNodeName[], uint32_t channelMask)=0`
Adds a MIDI node field to the plug-in's context.
- virtual `AAX_Result AddReservedField (AAX_CFieldIndex inFieldIndex, uint32_t inFieldType)=0`
Subscribes a context field to host-provided services or information.
- virtual `AAX_IPropertyMap * NewPropertyMap () const =0`
Creates a new, empty property map.
- virtual `AAX_IPropertyMap * DuplicatePropertyMap (AAX_IPropertyMap *inPropertyMap) const =0`
Creates a new property map using an existing property map.
- virtual `AAX_Result AddProcessProc_Native (AAX_CProcessProc inProcessProc, AAX_IPropertyMap *inProperties=NULL, AAX_CInstanceInitProc inInstanceInitProc=NULL, AAX_CBackgroundProc inBackgroundProc=NULL, AAX_CSelector *outProcID=NULL)=0`
Registers an algorithm processing entrypoint (process procedure) for the native architecture.
- virtual `AAX_Result AddProcessProc_TI (const char inDLLFileNameUTF8[], const char inProcessProcSymbol[], AAX_IPropertyMap *inProperties, const char inInstanceInitProcSymbol[]=NULL, const char inBackgroundProcSymbol[]=NULL, AAX_CSelector *outProcID=NULL)=0`
Registers an algorithm processing entrypoint (process procedure) for the native architecture.
- virtual `AAX_Result AddProcessProc (AAX_IPropertyMap *inProperties, AAX_CSelector *outProcIDs=NULL, int32_t inProcIDsSize=0)=0`
Registers one or more algorithm processing entrypoints (process procedures)
- template<typename aContextType > `AAX_Result AddProcessProc_Native (void(AAX_CALLBACK *inProcessProc)(aContextType *const inInstancesBegin[], const void *inInstancesEnd), AAX_IPropertyMap *inProperties=NULL, int32_t(AAX_CALLBACK *inInstanceInitProc)(const aContextType *inInstanceContextPtr, AAX_EComponentInstanceInitAction inAction)=NULL, int32_t(AAX_CALLBACK *inBackgroundProc)(void)=NULL)`
Registers an algorithm processing entrypoint (process procedure) for the native architecture.

14.81.2 Constructor & Destructor Documentation

14.81.2.1 virtual AAX_IComponentDescriptor::~AAX_IComponentDescriptor() [inline], [virtual]

14.81.3 Member Function Documentation

14.81.3.1 virtual AAX_Result AAX_IComponentDescriptor::Clear() [pure virtual]

Clears the descriptor.

Clears the descriptor and readies it for the next algorithm description

Implemented in [AAX_VComponentDescriptor](#).

14.81.3.2 virtual AAX_Result AAX_IComponentDescriptor::AddAudioIn(AAX_CFieldIndex *inFieldIndex*) [pure virtual]

Subscribes an audio input context field.

Defines an audio in port for host-provided information in the algorithm's context structure.

- Data type: float**
- Data kind: An array of float arrays, one for each input channel

Parameters

in	<i>inFieldIndex</i>	Unique identifier for the field, generated using AAX_FIELD_INDEX
----	---------------------	--

Implemented in [AAX_VComponentDescriptor](#).

Referenced by [AAX_CMonolithicParameters::StaticDescribe\(\)](#).

Here is the caller graph for this function:



14.81.3.3 virtual AAX_Result AAX_IComponentDescriptor::AddAudioOut(AAX_CFieldIndex *inFieldIndex*) [pure virtual]

Subscribes an audio output context field.

Defines an audio out port for host-provided information in the algorithm's context structure.

- Data type: float**
- Data kind: An array of float arrays, one for each output channel

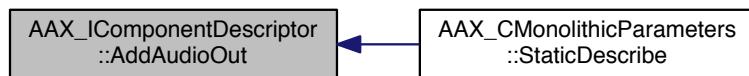
Parameters

in	<i>inFieldIndex</i>	Unique identifier for the field, generated using AAX_FIELD_INDEX
----	---------------------	--

Implemented in [AAX_VComponentDescriptor](#).

Referenced by [AAX_CMonolithicParameters::StaticDescribe\(\)](#).

Here is the caller graph for this function:



14.81.3.4 virtual AAX_Result AAX_IComponentDescriptor::AddAudioBufferLength (AAX_CFieldIndex *inFieldIndex*) [pure virtual]

Subscribes a buffer length context field.

Defines a buffer length port for host-provided information in the algorithm's context structure.

- Data type: `int32_t*`
- Data kind: The number of samples in the current audio buffer

Parameters

in	<i>inFieldIndex</i>	Unique identifier for the field, generated using AAX_FIELD_INDEX
----	---------------------	--

Implemented in [AAX_VComponentDescriptor](#).

Referenced by [AAX_CMonolithicParameters::StaticDescribe\(\)](#).

Here is the caller graph for this function:



14.81.3.5 virtual AAX_Result AAX_IComponentDescriptor::AddSampleRate (AAX_CFieldIndex *inFieldIndex*) [pure virtual]

Subscribes a sample rate context field.

Defines a sample rate port for host-provided information in the algorithm's context structure.

- Data type: [AAX_CSsampleRate](#) *
- Data kind: The current sample rate

Parameters

in	<i>inFieldIndex</i>	Unique identifier for the field, generated using AAX_FIELD_INDEX
----	---------------------	--

Implemented in [AAX_VComponentDescriptor](#).

14.81.3.6 virtual AAX_Result AAX_IComponentDescriptor::AddClock (AAX_CFieldIndex *inFieldIndex*) [pure virtual]

Subscribes a clock context field.

Defines a clock port for host-provided information in the algorithm's context structure.

- Data type: [AAX_CTimestamp](#) *
- Data kind: A running counter which increments even when the transport is not playing. The counter increments exactly once per sample quantum.

Host Compatibility Notes As of Pro Tools 11.1, this field may be used in both Native and DSP plug-ins. The DSP clock data is a 16-bit cycling counter. This field was only available for Native plug-ins in previous Pro Tools versions.

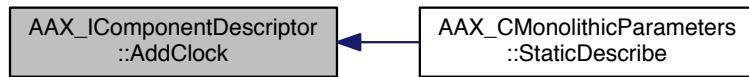
Parameters

in	<i>inFieldIndex</i>	Unique identifier for the field, generated using AAX_FIELD_INDEX
----	---------------------	--

Implemented in [AAX_VComponentDescriptor](#).

Referenced by [AAX_CMonolithicParameters::StaticDescribe\(\)](#).

Here is the caller graph for this function:



14.81.3.7 virtual AAX_Result AAX_IComponentDescriptor::AddSideChainIn (AAX_CFieldIndex *inFieldIndex*) [pure virtual]

Subscribes a side-chain input context field.

Defines a side-chain input port for host-provided information in the algorithm's context structure.

- Data type: int32_t*
- Data kind: The index of the plug-in's first side-chain input channel within the array of input audio buffers

Parameters

in	<i>inFieldIndex</i>	Unique identifier for the field, generated using AAX_FIELD_INDEX
----	---------------------	--

Implemented in [AAX_VComponentDescriptor](#).

```
14.81.3.8 virtual AAX_Result AAX_IComponentDescriptor::AddDataInPort( AAX_CFieldIndex inFieldIndex, uint32_t
inPacketSize, AAX_EDataInPortType inPortType = AAX_eDataInPortType_Buffed ) [pure
virtual]
```

Adds a custom data port to the algorithm context.

Defines a read-only data port for plug-in information in the algorithm's context structure. The plug-in can send information to this port using [AAX_IController::PostPacket\(\)](#).

The host guarantees that all packets will be delivered to this port in the order in which they were posted, up to the point of a packet buffer overflow, though some packets may be dropped depending on the *inPortType* and host implementation.

Note

When a plug-in is operating in offline (AudioSuite) mode, all data ports operate as [AAX_eDataInPortType_Unbuffered](#) ports

Parameters

in	<i>inFieldIndex</i>	Unique identifier for the port, generated using AAX_FIELD_INDEX
in	<i>inPacketSize</i>	Size of the data packets that will be sent to this port
in	<i>inPortType</i>	The requested packet delivery behavior for this port

Implemented in [AAX_VComponentDescriptor](#).

Referenced by [AAX_CMonolithicParameters::StaticDescribe\(\)](#).

Here is the caller graph for this function:



```
14.81.3.9 virtual AAX_Result AAX_IComponentDescriptor::AddAuxOutputStem( AAX_CFieldIndex inFieldIndex, int32_t
inStemFormat, const char inNameUTF8[] ) [pure virtual]
```

Adds an auxiliary output stem for a plug-in.

Use this method to add additional output channels to the algorithm context.

The aux output stem audio buffers will be added to the end of the audio outputs array in the order in which they are described. When writing audio data to a specific aux output, find the proper starting channel by accumulating all of the channels of the main output stem format and any previously-described aux output stems.

The plug-in is responsible for providing a meaningful name for each aux outputs. At the very least, individual outputs should be labeled "Output xx", where "xx" is the aux output number as it is defined in the plug-in. The output name should also include the words "mono" and "stereo" to support when users are looking for an output with a specific stem format.

Host Compatibility Notes There is a hard limit to the number of outputs that Pro Tools supports for a single plug-in instance. This limit is currently set at 256 channels, which includes all of the plug-in's output channels in addition to the sum total of all of its aux output stem channels.

Host Compatibility Notes Pro Tools supports only mono and stereo auxiliary output stem formats

Warning

This method will return an error code on hosts which do not support auxiliary output stems. This indicates that the host will not provide audio buffers for auxiliary output stems during processing. A plug-in must not attempt to write data into auxiliary output stem buffers which have not been provided by the host!

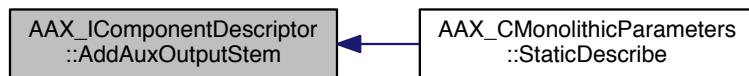
Parameters

in	<i>inFieldIndex</i>	DEPRECATED: This parameter is no longer needed by the host, but is included in the interface for binary compatibility
in	<i>inStemFormat</i>	The stem format of the new aux output
in	<i>inNameUTF8</i>	The name of the aux output. This name is static and cannot be changed after the descriptor is submitted to the host

Implemented in [AAX_VComponentDescriptor](#).

Referenced by [AAX_CMonolithicParameters::StaticDescribe\(\)](#).

Here is the caller graph for this function:



14.81.3.10 virtual AAX_Result AAX_IComponentDescriptor::AddPrivateData (AAX_CFieldIndex *inFieldIndex*, int32_t *inDataSize*, uint32_t *inOptions* = AAX_ePrivateDataOptions_DefaultOptions) [pure virtual]

Adds a private data port to the algorithm context.

Defines a read/write data port for private state data. Data written to this port will be maintained by the host between calls to the algorithm context.

See also

[alg_pd_registration](#)

Parameters

in	<i>inFieldIndex</i>	Unique identifier for the port, generated using AAX_FIELD_INDEX
in	<i>inDataSize</i>	Size of the data packets that will be sent to this port
in	<i>inOptions</i>	Options that define the private data port's behavior

Implemented in [AAX_VComponentDescriptor](#).

Referenced by [AAX_CMonolithicParameters::StaticDescribe\(\)](#).

Here is the caller graph for this function:



14.81.3.11 virtual [AAX_Result](#) [AAX_IComponentDescriptor::AddTemporaryData](#) ([AAX_CFieldIndex](#) *inFieldIndex*,
uint32_t *inDataElementSize*) [pure virtual]

Adds a block of data to a context that is not saved between callbacks and is scaled by the system buffer size.

This can be very useful if you use block processing and need to store intermediate results. Just specify your base element size and the system will scale the overall block size by the buffer size. For example, to create a buffer of floats that is the length of the block, specify 4 bytes as the elementsize.

This data block does not retain state across callback and can also be reused across instances on memory constrained systems.

Parameters

in	<i>inFieldIndex</i>	Unique identifier for the port, generated using AAX_FIELD_INDEX
in	<i>inDataElementSize</i>	The size of a single piece of data in the block. This number will be multiplied by the processing block size to determine total block size.

Implemented in [AAX_VComponentDescriptor](#).

14.81.3.12 virtual [AAX_Result](#) [AAX_IComponentDescriptor::AddDmaInstance](#) ([AAX_CFieldIndex](#) *inFieldIndex*,
[AAX_IDma::EMode](#) *inDmaMode*) [pure virtual]

Adds a DMA field to the plug-in's context.

DMA (direct memory access) provides efficient reads from and writes to external memory on the DSP. DMA behavior is emulated in host-based plug-ins for cross-platform portability.

Note

The order in which DMA instances are added defines their priority and therefore order of execution of DMA operations. In most plug-ins, Scatter fields should be placed first in order to achieve the lowest possible access latency.

For more information, see [Direct Memory Access](#).

Todo Update the DMA system management such that operation priority can be set arbitrarily

Parameters

in	<i>inFieldIndex</i>	Unique identifier for the field, generated using AAX_FIELD_INDEX
in	<i>inDmaMode</i>	AAX_IDma::EMode that will apply to this field

Implemented in [AAX_VComponentDescriptor](#).

14.81.3.13 virtual AAX_Result AAX_IComponentDescriptor::AddMeters (AAX_CFieldIndex *inFieldIndex*, const AAX_CTypeID * *inMeterIDs*, const uint32_t *inMeterCount*) [pure virtual]

Adds a meter field to the plug-in's context.

Meter fields include an array of meter tap values, with one tap per meter per context. Only one meter field should be added per Component. Individual meter behaviors can be described at the Effect level.

For more information, see [Plug-in meters](#).

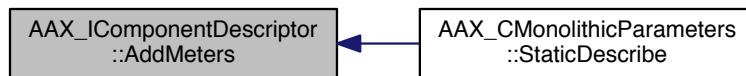
Parameters

in	<i>inFieldIndex</i>	Unique identifier for the field, generated using AAX_FIELD_INDEX
in	<i>inMeterIDs</i>	Array of 32-bit IDs, one for each meter. Meter IDs must be unique within the Effect.
in	<i>inMeterCount</i>	The number of meters included in this field

Implemented in [AAX_VComponentDescriptor](#).

Referenced by [AAX_CMonolithicParameters::StaticDescribe\(\)](#).

Here is the caller graph for this function:



14.81.3.14 virtual AAX_Result AAX_IComponentDescriptor::AddMIDINode (AAX_CFieldIndex *inFieldIndex*, AAX_EMIDINodeType *inNodeType*, const char *innodeName*[], uint32_t *channelMask*) [pure virtual]

Adds a MIDI node field to the plug-in's context.

- Data type: [AAX_IMIDINode](#) *

The resulting MIDI node data will be available both in the algorithm context and in the plug-in's [data model](#) via [UpdateMIDINodes\(\)](#).

To add a MIDI node that is only accessible to the plug-in's data model, use [AAX_IEffectDescriptor::AddControlMIDINode\(\)](#)

Host Compatibility Notes Due to current restrictions MIDI data won't be delivered to DSP algorithms, only to AAX Native.

Parameters

in	<i>inFieldIndex</i>	The ID of the port. MIDI node ports should be formatted as a pointer to an AA_X_IMIDINode .
in	<i>inNodeType</i>	The type of MIDI node, as AAX_EMIDINodeType
in	<i>innodeName</i>	The name of the MIDI node as it should appear in the host's UI
in	<i>channelMask</i>	The channel mask for the MIDI node. This parameter specifies used MIDI channels. For Global MIDI nodes, use a mask of AAX_EMidiGlobalNodeSelectors

Implemented in [AAX_VComponentDescriptor](#).

Referenced by [AAX_CMonolithicParameters::StaticDescribe\(\)](#).

Here is the caller graph for this function:



14.81.3.15 virtual [AAX_Result](#) [AAX_IComponentDescriptor::AddReservedField](#) ([AAX_CFieldIndex](#) *inFieldIndex*,
[uint32_t](#) *inFieldType*) [pure virtual]

Subscribes a context field to host-provided services or information.

Note

Currently for internal use only.

Parameters

in	<i>inFieldIndex</i>	Unique identifier for the field, generated using AAX_FIELD_INDEX
in	<i>inFieldType</i>	Type of field that is being added

Implemented in [AAX_VComponentDescriptor](#).

14.81.3.16 virtual [AAX_IPropertyMap*](#) [AAX_IComponentDescriptor::NewPropertyMap](#) () const [pure virtual]

Creates a new, empty property map.

The component descriptor owns the reference to the resulting property map, and the underlying property map is destroyed when the component descriptor is released.

Implemented in [AAX_VComponentDescriptor](#).

Referenced by [AAX_CMonolithicParameters::StaticDescribe\(\)](#).

Here is the caller graph for this function:



14.81.3.17 virtual **AAX_IPropertyMap*** **AAX_IComponentDescriptor::DuplicatePropertyMap** (**AAX_IPropertyMap *** *inPropertyMap*) const [pure virtual]

Creates a new property map using an existing property map.

The component descriptor owns the reference to the resulting property map, and the underlying property map is destroyed when the component descriptor is released.

Parameters

in	<i>inPropertyMap</i>	The property values in this map will be copied into the new map
----	----------------------	---

Implemented in [AAX_VComponentDescriptor](#).

14.81.3.18 virtual **AAX_Result** **AAX_IComponentDescriptor::AddProcessProc_Native** (**AAX_CProcessProc** *inProcessProc*, **AAX_IPropertyMap *** *inProperties* = NULL, **AAX_CInstanceInitProc** *inInstanceInitProc* = NULL, **AAX_CBackgroundProc** *inBackgroundProc* = NULL, **AAX_CSelector *** *outProcID* = NULL) [pure virtual]

Registers an algorithm processing entrypoint (process procedure) for the native architecture.

Parameters

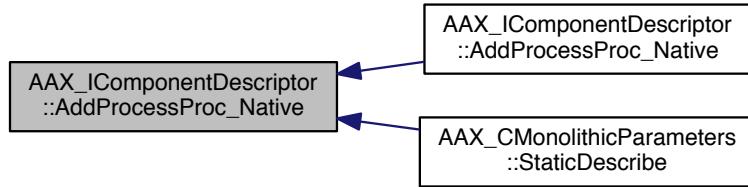
in	<i>inProcessProc</i>	Symbol for this processing callback
in	<i>inProperties</i>	A property map for this processing callback. The property map's values are copied by the host and associated with the new ProcessProc. The property map contents are unchanged and the map may be re-used when registering additional ProcessProcs.
in	<i>inInstanceInitProc</i>	Initialization routine that will be called when a new instance of the Effect is created. See Algorithm initialization .
in	<i>inBackgroundProc</i>	Background routine that will be called in an idle context within the same address space as the associated process procedure. See Background processing callback
out	<i>outProcID</i>	

Todo document this parameter

Implemented in [AAX_VComponentDescriptor](#).

Referenced by [AddProcessProc_Native\(\)](#), and [AAX_CMonolithicParameters::StaticDescribe\(\)](#).

Here is the caller graph for this function:



14.81.3.19 virtual **AAX_Result** **AAX_IComponentDescriptor::AddProcessProc_TI** (**const char** *inDLLFileNameUTF8*[], **const char** *inProcessProcSymbol*[], **AAX_IPropertyMap** * *inProperties*, **const char** *inInstanceInitProcSymbol*[] = **NULL**, **const char** *inBackgroundProcSymbol*[] = **NULL**, **AAX_CSelector** * *outProcID* = **NULL**) [pure virtual]

Registers an algorithm processing entrypoint (process procedure) for the native architecture.

Parameters

in	<i>inDLLFileNameUTF8</i>	UTF-8 encoded filename for the ELF DLL containing the algorithm code fragment
in	<i>inProcessProcSymbol</i>	Symbol for this processing callback
in	<i>inProperties</i>	A property map for this processing callback. The property map's values are copied by the host and associated with the new ProcessProc. The property map contents are unchanged and the map may be re-used when registering additional ProcessProcs.
in	<i>inInstanceInitProcSymbol</i>	Initialization routine that will be called when a new instance of the Effect is created. Must be included in the same DLL as the main algorithm entrypoint. See Algorithm initialization .
in	<i>inBackgroundProcSymbol</i>	Background routine that will be called in an idle context within the same address space as the associated process procedure. Must be included in the same DLL as the main algorithm entrypoint. See Background processing callback
out	<i>outProcID</i>	

Todo document this parameter

Implemented in [AAX_VComponentDescriptor](#).

14.81.3.20 virtual **AAX_Result** **AAX_IComponentDescriptor::AddProcessProc** (**AAX_IPropertyMap** * *inProperties*, **AAX_CSelector** * *outProcIDs* = **NULL**, **int32_t** *inProcIDsSize* = 0) [pure virtual]

Registers one or more algorithm processing entrypoints (process procedures)

Any non-overlapping set of processing entrypoints may be specified. Typically this can be used to specify both Native and TI entrypoints using the same call.

The AAX Library implementation of this method includes backwards compatibility logic to complete the Process→ Proc registration on hosts which do not support this method. Therefore plug-in code may use this single registration routine instead of separate calls to [AddProcessProc_Native\(\)](#), [AddProcessProc_TI\(\)](#), etc. regardless of the host version.

The following properties replace the input arguments to the platform-specific registration methods:

[AddProcessProc_Native\(\)](#) (`AAX_eProperty_PluginID_Native`, `AAX_eProperty_PluginID_AudioSuite`)

- `AAX_CProcessProc` `iProcessProc`: `AAX_eProperty_NativeProcessProc` (required)
- `AAX_CInstanceInitProc` `iInstanceInitProc`: `AAX_eProperty_NativeInstanceInitProc` (optional)
- `AAX_CBackgroundProc` `iBackgroundProc`: `AAX_eProperty_NativeBackgroundProc` (optional)

[AddProcessProc_TI\(\)](#) (`AAX_eProperty_PluginID_TI`)

- `const char inDLLFileNameUTF8[]`: `AAX_eProperty_TIDLLFileName` (required)
- `const char iProcessProcSymbol[]`: `AAX_eProperty_TIProcessProc` (required)
- `const char iInstanceInitProcSymbol[]`: `AAX_eProperty_TIInstanceInitProc` (optional)
- `const char iBackgroundProcSymbol[]`: `AAX_eProperty_TIBackgroundProc` (optional)

If any platform-specific plug-in ID property is present in `iProperties` then [AddProcessProc\(\)](#) will check for the required properties for that platform.

Note

`AAX_eProperty_AudioBufferLength` will be ignored for the Native and AudioSuite ProcessProcs since it should only be used for AAX DSP.

Parameters

in	<code>inProperties</code>	A property map for this processing callback. The property map's values are copied by the host and associated with the new ProcessProc. The property map contents are unchanged and the map may be re-used when registering additional ProcessProcs.
out	<code>outProcIDs</code>	

Todo document this parameter Returned array will be NULL-terminated

Parameters

in	<code>inProcIDsSize</code>	The size of the array provided to <code>oProcIDs</code> . If <code>oProcIDs</code> is non-NULL but <code>iProcIDsSize</code> is not large enough for all of the registered ProcessProcs (plus one for NULL termination) then this method will fail with <code>AAX_ERROR_ARGUMENT_OVERFLOW</code>
----	----------------------------	--

Implemented in [AAX_VComponentDescriptor](#).

```
14.81.3.21 template<typename aContextType > AAX_Result AAX_IComponentDescriptor::AddProcessProc_Native (
    void(AAX_CALLBACK *inProcessProc)(aContextType *const inInstancesBegin[], const void *inInstancesEnd),
    , AAX_IPropertyMap * inProperties = NULL, int32_t(AAX_CALLBACK *inInstanceInitProc)(const
    aContextType *inInstanceContextPtr, AAX_EComponentInstanceInitAction inAction) = NULL,
    int32_t(AAX_CALLBACK *inBackgroundProc)(void) = NULL ) [inline]
```

Registers an algorithm processing entrypoint (process procedure) for the native architecture.

This template provides an `AAX_CALLBACK` based interface to the [AddProcessProc_Native](#) method.

See also

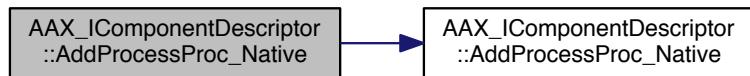
[AAX_IComponentDescriptor::AddProcessProc_Native\(AAX_CProcessProc,AAX_IPropertyMap*,AAX_CInstanceInitProc,AAX_CBackgroundProc,AAX_CSelector*\)](#)

Parameters

in	<i>inProperties</i>	A property map for this processing callback. The property map's values are copied by the host and associated with the new ProcessProc. The property map contents are unchanged and the map may be re-used when registering additional ProcessProcs.
----	---------------------	---

References AddProcessProc_Native().

Here is the call graph for this function:



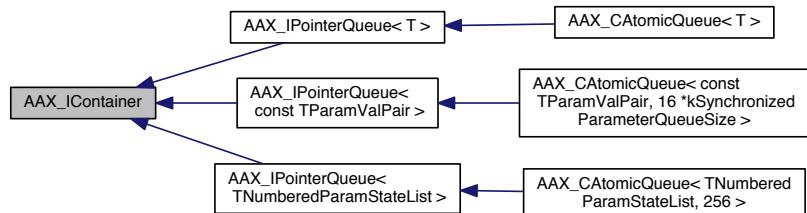
The documentation for this class was generated from the following file:

- [AAX_IComponentDescriptor.h](#)

14.82 AAX.IContainer Class Reference

```
#include <AAX.IContainer.h>
```

Inheritance diagram for AAX.IContainer:



14.82.1 Description

Abstract container interface

Public Types

- enum [EStatus](#) { [eStatus_Success](#) = 0, [eStatus_Overflow](#) = 1, [eStatus_NotInitialized](#) = 2, [eStatus_Unavailable](#) = 3, [eStatus_Unsupported](#) = 4 }

Public Member Functions

- virtual [~AAX.IContainer](#) ()

- virtual void [Clear \(\)=0](#)

14.82.2 Member Enumeration Documentation

14.82.2.1 enum AAX_IContainer::EStatus

Enumerator

- eStatus_Success** Operation succeeded.
- eStatus_Overflow** Internal buffer overflow.
- eStatus_NotInitialized** Uninitialized container.
- eStatus_Unavailable** An internal resource was not available.
- eStatus_Unsupported** Operation is unsupported.

14.82.3 Constructor & Destructor Documentation

14.82.3.1 virtual AAX_IContainer::~AAX_IContainer() [inline], [virtual]

14.82.4 Member Function Documentation

14.82.4.1 virtual void AAX_IContainer::Clear() [pure virtual]

Clear the container

Implemented in [AAX_CAtomicQueue< T, S >](#), [AAX_CAtomicQueue< TNumberedParamStateList, 256 >](#), [AAX_CAtomicQueue< const TParamValPair, 16 *kSynchronizedParameterQueueSize >](#), [AAX_IPointerQueue< T >](#), [AAX_IPointerQueue< const TParamValPair >](#), and [AAX_IPointerQueue< TNumberedParamStateList >](#).

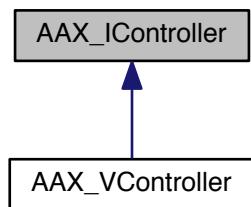
The documentation for this class was generated from the following file:

- [AAX_IContainer.h](#)

14.83 AAX_IController Class Reference

```
#include <AAX_IController.h>
```

Inheritance diagram for AAX_IController:



14.83.1 Description

Interface for the AAX host's view of a single instance of an effect. Used by both clients of the AAX host and by effect components.

:Implemented by the AAX Host

Public Member Functions

- virtual ~AAX_IController (void)
- virtual [AAX_Result GetHybridSignalLatency](#) (int32_t *outSamples) const =0

CALL: Returns the latency between the algorithm normal input samples and the inputs returning from the hybrid component.
- virtual [AAX_Result GetCurrentAutomationTimestamp](#) ([AAX_CTransportCounter](#) *outTimestamp) const =0

CALL: Returns the current automation timestamp if called during the [GenerateCoefficients\(\)](#) call AND the generation of coefficients is being triggered by an automation point instead of immediate changes.
- virtual [AAX_Result GetHostName](#) ([AAX_IString](#) *outHostNameString) const =0

CALL: Returns name of the host application this plug-in instance is being loaded by. This string also typically includes version information.
- virtual [AAX_Result GetPlugInTargetPlatform](#) ([AAX_CTargetPlatform](#) *outTargetPlatform) const =0

CALL: Returns execution platform type, native or TI.
- virtual [AAX_Result GetIsAudioSuite](#) ([AAX_CBoolean](#) *outIsAudioSuite) const =0

CALL: Returns true for AudioSuite instances.
- virtual [AAX_IPageTable * CreateTableCopyForEffect](#) ([AAX_CPropertyValue](#) inManufacturerID, [AAX_CProperty](#) inProductID, [AAX_CPropertyValue](#) inPlugInID, uint32_t inTableType, int32_t inTablePageSize) const =0

Copy the current page table data for a particular plug-in type.
- virtual [AAX_IPageTable * CreateTableCopyForLayout](#) (const char *inEffectID, const char *inLayoutName, uint32_t inTableType, int32_t inTablePageSize) const =0

Copy the current page table data for a particular plug-in effect and page table layout.
- virtual [AAX_IPageTable * CreateTableCopyForEffectFromFile](#) (const char *inPageTableFilePath, [AAX_ETextEncoding](#) inFilePathEncoding, [AAX_CPropertyValue](#) inManufacturerID, [AAX_CPropertyValue](#) inProductID, [AAX_CPropertyValue](#) inPlugInID, uint32_t inTableType, int32_t inTablePageSize) const =0

Copy the current page table data for a particular plug-in type.
- virtual [AAX_IPageTable * CreateTableCopyForLayoutFromFile](#) (const char *inPageTableFilePath, [AAX_ETextEncoding](#) inFilePathEncoding, const char *inLayoutName, uint32_t inTableType, int32_t inTablePageSize) const =0

Copy the current page table data for a particular plug-in effect and page table layout.

Host information getters

Call these methods to retrieve environment and run-time information from the AAX host.

- virtual [AAX_Result GetEffectID](#) ([AAX_IString](#) *outEffectID) const =0
- virtual [AAX_Result GetSampleRate](#) ([AAX_CSampleRate](#) *outSampleRate) const =0

CALL: Returns the current literal sample rate.
- virtual [AAX_Result GetInputStemFormat](#) ([AAX_EStemFormat](#) *outStemFormat) const =0

CALL: Returns the plug-in's input stem format.
- virtual [AAX_Result GetOutputStemFormat](#) ([AAX_EStemFormat](#) *outStemFormat) const =0

CALL: Returns the plug-in's output stem format.
- virtual [AAX_Result GetSignalLatency](#) (int32_t *outSamples) const =0

CALL: Returns the most recent signal (algorithmic) latency that has been published by the plug-in.
- virtual [AAX_Result GetCycleCount](#) ([AAX_EProperty](#) inWhichCycleCount, [AAX_CPropertyValue](#) *outNumCycles) const =0

CALL: returns the plug-in's current real-time DSP cycle count.
- virtual [AAX_Result GetTODLocation](#) ([AAX_CTimeOfDay](#) *outTODLocation) const =0

CALL: Returns the current Time Of Day (TOD) of the system.

Host information setters

Call these methods to set dynamic plug-in run-time information on the AAX host.

- virtual [AAX_Result SetSignalLatency](#) (int32_t inNumSamples)=0
 - CALL:* Submits a request to change the delay compensation value that the host uses to account for the plug-in's signal (algorithmic) latency.
- virtual [AAX_Result SetCycleCount](#) ([AAX_EProperty](#) *inWhichCycleCounts, [AAX_CPropertyValue](#) *iValues, int32_t numValues)=0
 - CALL:* Indicates a change in the plug-in's real-time DSP cycle count.

Posting methods

Call these methods to post new plug-in information to the host's data management system.

- virtual [AAX_Result PostPacket](#) ([AAX_CFieldIndex](#) inFieldIndex, const void *inPayloadP, uint32_t inPayloadSize)=0
 - CALL:* Posts a data packet to the host for routing between plug-in components.

Notification methods

Call these methods to send events among plug-in components

- virtual [AAX_Result SendNotification](#) ([AAX_CTypeID](#) inNotificationType, const void *inNotificationData, uint32_t inNotificationDataSize)=0
 - CALL:* Dispatch a notification.
- virtual [AAX_Result SendNotification](#) ([AAX_CTypeID](#) inNotificationType)=0
 - CALL:* Sends an event to the GUI (no payload)

Metering methods

Methods to access the plug-in's host-managed metering information.

See also

[Plug-in meters](#)

- virtual [AAX_Result GetCurrentMeterValue](#) ([AAX_CTypeID](#) inMeterID, float *outMeterValue) const =0
 - CALL:* Retrieves the current value of a host-managed plug-in meter.
- virtual [AAX_Result GetMeterPeakValue](#) ([AAX_CTypeID](#) inMeterID, float *outMeterPeakValue) const =0
 - CALL:* Retrieves the currently held peak value of a host-managed plug-in meter.
- virtual [AAX_Result ClearMeterPeakValue](#) ([AAX_CTypeID](#) inMeterID) const =0
 - CALL:* Clears the peak value from a host-managed plug-in meter.
- virtual [AAX_Result GetMeterCount](#) (uint32_t *outMeterCount) const =0
 - CALL:* Retrieves the number of host-managed meters registered by a plug-in.
- virtual [AAX_Result GetMeterClipped](#) ([AAX_CTypeID](#) inMeterID, [AAX_CBoolean](#) *outClipped) const =0
 - CALL:* Retrieves the clipped flag from a host-managed plug-in meter.
- virtual [AAX_Result ClearMeterClipped](#) ([AAX_CTypeID](#) inMeterID) const =0
 - CALL:* Clears the clipped flag from a host-managed plug-in meter.

MIDI methods

Methods to access the plug-in's host-managed MIDI information.

- virtual [AAX_Result GetNextMIDIpacket](#) ([AAX_CFieldIndex](#) *outPort, [AAX_CMidiPacket](#) *outPacket)=0
 - CALL:* Retrieves MIDI packets for described MIDI nodes.

14.83.2 Constructor & Destructor Documentation

14.83.2.1 virtual AAX_IController::~AAX_IController(void) [inline], [virtual]

14.83.3 Member Function Documentation

14.83.3.1 virtual AAX_Result AAX_IController::GetEffectID(AAX_IString * *outEffectID*) const [pure virtual]

Implemented in [AAX_VController](#).

14.83.3.2 virtual AAX_Result AAX_IController::GetSampleRate(AAX_CSampleRate * *outSampleRate*) const [pure virtual]

CALL: Returns the current literal sample rate.

Parameters

out	<i>outSampleRate</i>	The current sample rate
-----	----------------------	-------------------------

Implemented in [AAX_VController](#).

14.83.3.3 virtual AAX_Result AAX_IController::GetInputStemFormat(AAX_EStemFormat * *outStemFormat*) const [pure virtual]

CALL: Returns the plug-in's input stem format.

Parameters

out	<i>outStemFormat</i>	The current input stem format
-----	----------------------	-------------------------------

Implemented in [AAX_VController](#).

14.83.3.4 virtual AAX_Result AAX_IController::GetOutputStemFormat(AAX_EStemFormat * *outStemFormat*) const [pure virtual]

CALL: Returns the plug-in's output stem format.

Parameters

out	<i>outStemFormat</i>	The current output stem format
-----	----------------------	--------------------------------

Implemented in [AAX_VController](#).

14.83.3.5 virtual AAX_Result AAX_IController::GetSignalLatency(int32_t * *outSamples*) const [pure virtual]

CALL: Returns the most recent signal (algorithmic) latency that has been published by the plug-in.

This method provides the most recently published signal latency. The host may not have updated its delay compensation to match this signal latency yet, so plug-ins that dynamically change their latency using [SetSignalLatency\(\)](#) should always wait for an [AAX_eNotificationEvent_SignalLatencyChanged](#) notification before updating its algorithm to incur this latency.

See also

[SetSignalLatency\(\)](#)

Parameters

<code>out</code>	<code>outSamples</code>	The number of samples of signal delay published by the plug-in
------------------	-------------------------	--

Implemented in [AAX_VController](#).

14.83.3.6 virtual AAX_Result AAX_IController::GetCycleCount ([AAX_EProperty inWhichCycleCount](#), [AAX_CPropertyValue * outNumCycles](#)) const [pure virtual]

CALL: returns the plug-in's current real-time DSP cycle count.

This method provides the number of cycles that the AAX host expects the DSP plug-in to consume. The host uses this value when allocating DSP resources for the plug-in.

Note

A plug-in should never apply a DSP algorithm with more demanding resource requirements than what is currently accounted for by the host. To set a higher cycle count value, a plug-in must call [AAX_IController::SetCycleCount\(\)](#), then poll [AAX_IController::GetCycleCount\(\)](#) until the new value has been applied. Once the host has recognized the new cycle count value, the plug-in may apply the more demanding algorithm.

Parameters

<code>in</code>	<code>inWhichCycleCount</code>	Selector for the requested cycle count metric. One of: <ul style="list-style-type: none">• AAX_eProperty_TI_SharedCycleCount• AAX_eProperty_TI_InstanceCycleCount• AAX_eProperty_TI_MaxInstancesPerChip
<code>in</code>	<code>outNumCycles</code>	The current value of the selected cycle count metric

Todo PLACEHOLDER - NOT CURRENTLY IMPLEMENTED IN HOST

Implemented in [AAX_VController](#).

14.83.3.7 virtual AAX_Result AAX_IController::GetTODLocation ([AAX_CTimeOfDay * outTODLocation](#)) const [pure virtual]

CALL: Returns the current Time Of Day (TOD) of the system.

This method provides a plug-in the TOD (in samples) of the current system. TOD is the number of samples that the playhead has traversed since the beginning of playback.

Note

The TOD value is the immediate value of the audio engine playhead. This value is incremented within the audio engine's real-time rendering context; it is not synchronized with non-real-time calls to plug-in interface methods.

Parameters

<code>out</code>	<code>outTODLocation</code>	The current Time Of Day as set by the host
------------------	-----------------------------	--

Implemented in [AAX_VController](#).

14.83.3.8 virtual AAX_Result AAX_IController::SetSignalLatency (`int32_t inNumSamples`) [pure virtual]

CALL: Submits a request to change the delay compensation value that the host uses to account for the plug-in's signal (algorithmic) latency.

This method is used to request a change in the number of samples that the AAX host expects the plug-in to delay a signal.

The host is not guaranteed to immediately apply the new latency value. A plug-in should avoid incurring an actual algorithmic latency that is different than the latency accounted for by the host.

To set a new latency value, a plug-in must call [AAX_IController::SetSignalLatency\(\)](#), then wait for an [AAX_eNotificationEvent_SignalLatencyChanged](#) notification. Once this notification has been received, [AAX_IController::GetSignalLatency\(\)](#) will reflect the updated latency value and the plug-in should immediately apply any relevant algorithmic changes that alter its latency to this new value.

Warning

Parameters which affect the latency of a plug-in should not be made available for control through automation. This will result in audible glitches when delay compensation is adjusted while playing back automation for these parameters.

Parameters

in	<i>inNumSamples</i>	The number of samples of signal delay that the plug-in requests to incur
----	---------------------	--

Implemented in [AAX_VController](#).

14.83.3.9 virtual AAX_Result AAX_IController::SetCycleCount (AAX_EProperty * *inWhichCycleCounts*, AAX_CPropertyValue * *iValues*, int32_t *numValues*) [pure virtual]

CALL: Indicates a change in the plug-in's real-time DSP cycle count.

This method is used to request a change in the number of cycles that the AAX host expects the DSP plug-in to consume.

Note

A plug-in should never apply a DSP algorithm with more demanding resource requirements than what is currently accounted for by the host. To set a higher cycle count value, a plug-in must call [AAX_IController::SetCycleCount\(\)](#), then poll [AAX_IController::GetCycleCount\(\)](#) until the new value has been applied. Once the host has recognized the new cycle count value, the plug-in may apply the more demanding algorithm.

Parameters

in	<i>inWhichCycleCounts</i>	Array of selectors indicating the specific cycle count metrics that should be set. Each selector must be one of: <ul style="list-style-type: none"> • AAX_eProperty_TI_SharedCycleCount • AAX_eProperty_TI_InstanceCycleCount • AAX_eProperty_TI_MaxInstancesPerChip
----	---------------------------	---

in	<i>iValues</i>	An array of values requested, one for each of the selected cycle count metrics.
in	<i>numValues</i>	The size of <i>iValues</i>

Todo PLACEHOLDER - NOT CURRENTLY IMPLEMENTED IN HOST

Implemented in [AAX_VController](#).

14.83.3.10 virtual AAX_Result AAX_IController::PostPacket (AAX_CFieldIndex *inFieldIndex*, const void * *inPayloadP*, uint32_t *inPayloadSize*) [pure virtual]

CALL: Posts a data packet to the host for routing between plug-in components.

The posted packet is identified with a [AAX_CFieldIndex](#) packet index value, which is equivalent to the target data port's identifier. The packet's payload must have the expected size for the given packet index / data port, as defined when the port is created in [Describe](#). See [AAX_IComponentDescriptor::AddDataInPort\(\)](#).

Warning

Any data structures that will be passed between platforms (for example, sent to a TI DSP in an AAX DSP plug-in) must be properly data-aligned for compatibility across both platforms. See [AAX_ALIGN_FILE_ALG](#) for more information about guaranteeing cross-platform compatibility of data structures used for algorithm processing.

Note

All calls to this method should be made within the scope of [AAX_IEffectParameters::GenerateCoefficients\(\)](#). Calls from outside this method may result in packets not being delivered. See [PT-206161](#)

Parameters

in	<i>inFieldIndex</i>	The packet's destination port
in	<i>inPayloadP</i>	A pointer to the packet's payload data
in	<i>inPayloadSize</i>	The size, in bytes, of the payload data

Implemented in [AAX_VController](#).

Referenced by [AAX_CMonolithicParameters::GenerateCoefficients\(\)](#).

Here is the caller graph for this function:



14.83.3.11 virtual AAX_Result AAX_IController::SendNotification (AAX_CTypeID *inNotificationType*, const void * *inNotificationData*, uint32_t *inNotificationDataSize*) [pure virtual]

CALL: Dispatch a notification.

The notification is handled by the host and may be delivered back to other plug-in components such as the G↔UI or data model (via [AAX_IEffectGUI::NotificationReceived\(\)](#) or [AAX_IEffectParameters::NotificationReceived\(\)](#), respectively) depending on the notification type.

The host may choose to dispatch the posted notification either synchronously or asynchronously.

See the [AAX_ENotificationEvent](#) documentation for more information.

This method is supported by AAX V2 Hosts only. Check the return code on the return of this function. If the error is [AAX_ERROR_UNIMPLEMENTED](#), your plug-in is being loaded into a host that doesn't support this feature.

Parameters

in	<i>inNotificationType</i>	Type of notification to send
in	<i>inNotificationData</i>	Block of notification data
in	<i>inNotificationDataSize</i>	Size of <i>inNotificationData</i> , in bytes

Implemented in [AAX_VController](#).

14.83.3.12 virtual **AAX_Result** **AAX_IController::SendNotification** (**AAX_CTypeID** *inNotificationType*) [pure virtual]

CALL: Sends an event to the GUI (no payload)

This version of the notification method is a convenience for notifications which do not take any payload data. Internally, it simply calls [AAX_IController::SendNotification\(AAX_CTypeID, const void*, uint32_t\)](#) with a null payload.

Parameters

in	<i>inNotificationType</i>	Type of notification to send
----	---------------------------	------------------------------

Implemented in [AAX_VController](#).

14.83.3.13 virtual **AAX_Result** **AAX_IController::GetCurrentMeterValue** (**AAX_CTypeID** *inMeterID*, float * *outMeterValue*) const [pure virtual]

CALL: Retrieves the current value of a host-managed plug-in meter.

Parameters

in	<i>inMeterID</i>	ID of the meter that is being queried
out	<i>outMeterValue</i>	The queried meter's current value

Implemented in [AAX_VController](#).

14.83.3.14 virtual **AAX_Result** **AAX_IController::GetMeterPeakValue** (**AAX_CTypeID** *inMeterID*, float * *outMeterPeakValue*) const [pure virtual]

CALL: Retrieves the currently held peak value of a host-managed plug-in meter.

Parameters

in	<i>inMeterID</i>	ID of the meter that is being queried
out	<i>outMeterPeakValue</i>	The queried meter's currently held peak value

Implemented in [AAX_VController](#).

```
14.83.3.15 virtual AAX_Result AAX_IController::ClearMeterPeakValue ( AAX_CTypeID inMeterID ) const [pure  
virtual]
```

CALL: Clears the peak value from a host-managed plug-in meter.

Parameters

in	<i>inMeterID</i>	ID of the meter that is being cleared
----	------------------	---------------------------------------

Implemented in [AAX_VController](#).

14.83.3.16 virtual AAX_Result AAX_IController::GetMeterCount (*uint32_t * outMeterCount*) const [pure virtual]

CALL: Retrieves the number of host-managed meters registered by a plug-in.

See [AAX_IComponentDescriptor::AddMeters\(\)](#).

Parameters

out	<i>outMeterCount</i>	The number of registered plug-in meters.
-----	----------------------	--

Implemented in [AAX_VController](#).

14.83.3.17 virtual AAX_Result AAX_IController::GetMeterClipped (*AAX_CTypeID inMeterID*, *AAX_CBoolean * outClipped*) const [pure virtual]

CALL: Retrieves the clipped flag from a host-managed plug-in meter.

See [AAX_IComponentDescriptor::AddMeters\(\)](#).

Parameters

in	<i>inMeterID</i>	ID of the meter that is being queried.
out	<i>outClipped</i>	The queried meter's clipped flag.

Implemented in [AAX_VController](#).

14.83.3.18 virtual AAX_Result AAX_IController::ClearMeterClipped (*AAX_CTypeID inMeterID*) const [pure virtual]

CALL: Clears the clipped flag from a host-managed plug-in meter.

See [AAX_IComponentDescriptor::AddMeters\(\)](#).

Parameters

in	<i>inMeterID</i>	ID of the meter that is being cleared.
----	------------------	--

Implemented in [AAX_VController](#).

14.83.3.19 virtual AAX_Result AAX_IController::GetNextMIDIPacket (*AAX_CFieldIndex * outPort*, *AAX_CMidiPacket * outPacket*) [pure virtual]

CALL: Retrieves MIDI packets for described MIDI nodes.

Parameters

out	<i>outPort</i>	port ID of the MIDI node that has unhandled packet
out	<i>outPacket</i>	The MIDI packet

Implemented in [AAX_VController](#).

14.83.3.20 virtual AAX_Result AAX_IController::GetCurrentAutomationTimestamp (*AAX_CTransportCounter * outTimestamp*) const [pure virtual]

CALL: Returns the current automation timestamp if called during the [GenerateCoefficients\(\)](#) call AND the generation of coefficients is being triggered by an automation point instead of immediate changes.

Note

This function will return 0 if called from outside of [GenerateCoefficients\(\)](#) or if the [GenerateCoefficients\(\)](#) call was initiated due to a non-automated change. In those cases, you can get your sample offset from the transport start using [GetTODLocation\(\)](#).

Parameters

<code>out</code>	<code>outTimestamp</code>	The current coefficient timestamp. Sample count from transport start.
------------------	---------------------------	---

Implemented in [AAX_VController](#).

14.83.3.21 virtual AAX_Result AAX_IController::GetHostName (AAX_IString * outHostNameString) const [pure virtual]

CALL: Returns name of the host application this plug-in instance is being loaded by. This string also typically includes version information.

Host Compatibility Notes Pro Tools versions from Pro Tools 11.0 to Pro Tools 12.3.1 will return a generic version string to this call. This issue is resolved beginning in Pro Tools 12.4.

Parameters

<code>out</code>	<code>outHostNameString</code>	The name of the current host application.
------------------	--------------------------------	---

Implemented in [AAX_VController](#).

14.83.3.22 virtual AAX_Result AAX_IController::GetPlugInTargetPlatform (AAX_CTargetPlatform * outTargetPlatform) const [pure virtual]

CALL: Returns execution platform type, native or TI.

Parameters

<code>out</code>	<code>outTargetPlatform</code>	The type of the current execution platform as one of AAX_ETargetPlatform .
------------------	--------------------------------	--

Implemented in [AAX_VController](#).

14.83.3.23 virtual AAX_Result AAX_IController::GetIsAudioSuite (AAX_CBoolean * outIsAudioSuite) const [pure virtual]

CALL: Returns true for AudioSuite instances.

Parameters

<code>out</code>	<code>outIsAudioSuite</code>	The boolean flag which indicate true for AudioSuite instances.
------------------	------------------------------	--

Implemented in [AAX_VController](#).

14.83.3.24 virtual AAX_IPageTable* AAX_IController::CreateTableCopyForEffect (AAX_CPropertyValue inManufacturerID, AAX_CPropertyValue inProductID, AAX_CPropertyValue inPlugInID, uint32_t inTableType, int32_t inTablePageSize) const [pure virtual]

Copy the current page table data for a particular plug-in type.

The host may restrict plug-ins to only copying page table data from certain plug-in types, such as plug-ins from the same manufacturer or plug-in types within the same effect.

See [Page Table Guide](#) for more information about page tables.

Returns

A new page table object to which the requested page table data has been copied. Ownership of this object passes to the caller.

a null pointer if the requested plug-in type is unknown, if `inTableType` is unknown or if `inTablePageSize` is not a supported size for the given table type.

Parameters

in	<i>inManufacturerID</i>	Manufacturer ID of the desired plug-in type
in	<i>inProductID</i>	Product ID of the desired plug-in type
in	<i>inPlugInID</i>	Type ID of the desired plug-in type (AAX_eProperty_PlugInID_Native , AAX_eProperty_PlugInID_TI)
in	<i>inTableType</i>	Four-char type identifier for the requested table type (e.g. 'PgTL', 'Av81', etc.)
in	<i>inTablePageSize</i>	Page size for the requested table. Some tables support multiple page sizes.

Implemented in [AAX_VController](#).

14.83.3.25 virtual [AAX_IPageTable](#)* [AAX_IController::CreateTableCopyForLayout](#) (const char * *inEffectID*, const char * *inLayoutName*, uint32_t *inTableType*, int32_t *inTablePageSize*) const [pure virtual]

Copy the current page table data for a particular plug-in effect and page table layout.

The host may restrict plug-ins to only copying page table data from certain effects, such as effects registered within the current [AAX](#) plug-in bundle.

See [Page Table Guide](#) for more information about page tables.

Returns

A new page table object to which the requested page table data has been copied. Ownership of this object passes to the caller.

a null pointer if the requested effect ID is unknown or if `inLayoutName` is not a valid layout name for the page tables registered for the effect.

Parameters

in	<i>inEffectID</i>	Effect ID for the desired effect. See AAX_ICollection::AddEffect()
in	<i>inLayoutName</i>	Page table layout name ("name" attribute of the PTLayout XML tag)
in	<i>inTableType</i>	Four-char type identifier for the requested table type (e.g. 'PgTL', 'Av81', etc.)
in	<i>inTablePageSize</i>	Page size for the requested table. Some tables support multiple page sizes.

Implemented in [AAX_VController](#).

14.83.3.26 virtual [AAX_IPageTable](#)* [AAX_IController::CreateTableCopyForEffectFromFile](#) (const char * *inPageTableFilePath*, [AAX_ETextEncoding](#) *inFilePathEncoding*, [AAX_CPropertyValue](#) *inManufacturerID*, [AAX_CPropertyValue](#) *inProductID*, [AAX_CPropertyValue](#) *inPlugInID*, uint32_t *inTableType*, int32_t *inTablePageSize*) const [pure virtual]

Copy the current page table data for a particular plug-in type.

Returns

A new page table object to which the requested page table data has been copied. Ownership of this object passes to the caller.

a null pointer if the requested plug-in type is unkown, if `inTableType` is unknown or if `inTablePageSize` is not a supported size for the given table type.

Parameters

in	<i>inPageTable</i> \leftarrow <i>FilePath</i>	Path to XML page table file.
in	<i>inFilePath</i> \leftarrow <i>Encoding</i>	File path text encoding.
in	<i>in</i> \leftarrow <i>ManufacturerID</i>	Manufacturer ID of the desired plug-in type
in	<i>inProductID</i>	Product ID of the desired plug-in type
in	<i>inPlugInID</i>	Type ID of the desired plug-in type (AAX_eProperty_PlugInID_Native , AAX_eProperty_PlugInID_TI)
in	<i>inTableType</i>	Four-char type identifier for the requested table type (e.g. 'PgTL', 'Av81', etc.)
in	<i>inTablePageSize</i>	Page size for the requested table. Some tables support multiple page sizes.

Implemented in [AAX_VController](#).

14.83.3.27 virtual [AAX_IPageTable](#)* [AAX_IController::CreateTableCopyForLayoutFromFile](#) (const char * *inPageTableFilePath*, [AAX_ETextEncoding](#) *inFilePathEncoding*, const char * *inLayoutName*, uint32_t *inTableType*, int32_t *inTablePageSize*) const [pure virtual]

Copy the current page table data for a particular plug-in effect and page table layout.

Returns

A new page table object to which the requested page table data has been copied. Ownership of this object passes to the caller.
a null pointer if *inLayoutName* is not a valid layout name for the page tables file.

Parameters

in	<i>inPageTable</i> \leftarrow <i>FilePath</i>	Path to XML page table file.
in	<i>inFilePath</i> \leftarrow <i>Encoding</i>	File path text encoding.
in	<i>inLayoutName</i>	Page table layout name ("name" attribute of the PTLayout XML tag)
in	<i>inTableType</i>	Four-char type identifier for the requested table type (e.g. 'PgTL', 'Av81', etc.)
in	<i>inTablePageSize</i>	Page size for the requested table. Some tables support multiple page sizes.

Implemented in [AAX_VController](#).

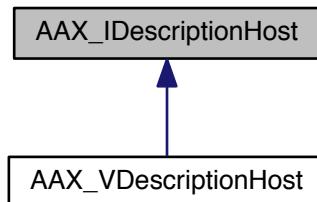
The documentation for this class was generated from the following file:

- [AAX_IController.h](#)

14.84 AAX_IDescriptionHost Class Reference

```
#include <AAX_IDescriptionHost.h>
```

Inheritance diagram for AAX_IDescriptionHost:



14.84.1 Description

Interface to host services provided during plug-in description

Public Member Functions

- virtual ~AAX_IDescriptionHost ()
- virtual const [AAX_IFeatureInfo](#) * AcquireFeatureProperties (const [AAX_Feature_UID](#) &inFeatureID) const =0

14.84.2 Constructor & Destructor Documentation

14.84.2.1 virtual AAX_IDescriptionHost::~AAX_IDescriptionHost() [inline], [virtual]

14.84.3 Member Function Documentation

14.84.3.1 virtual const [AAX_IFeatureInfo](#)* AAX_IDescriptionHost::AcquireFeatureProperties (const [AAX_Feature_UID](#) & inFeatureID) const [pure virtual]

Get the client's feature object for a given feature ID

Similar to `QueryInterface()` but uses a feature identifier rather than a true IID

Ownership of the returned object is passed to the caller; the caller is responsible for destroying the object, e.g. by capturing the returned object in a smart pointer.

```
// AAX_IDescriptionHost* descHost
std::unique_ptr<const AAX_IFeatureInfo> featureInfoPtr(descHost->AcquireFeatureProperties(someFeatureUID));
```

Returns

An [AAX_IFeatureInfo](#) interface with access to the host's feature properties for this feature.
NULL if the desired feature was not found or if an error occurred

Note

May return an [AAX_IFeatureInfo](#) object with limited method support, which would return an error such as [AAX_ERROR_NULL_OBJECT](#) or [AAX_ERROR_UNIMPLEMENTED](#) to interface calls.

If no [AAX_IFeatureInfo](#) is provided then that may mean that the host is unaware of the feature, or it may mean that the host is aware of the feature but has not implemented the AAX feature support interface for this feature yet.

Parameters

in	<i>inFeatureID</i>	Identifier of the requested feature
----	--------------------	-------------------------------------

Implemented in [AAX_VDescriptionHost](#).

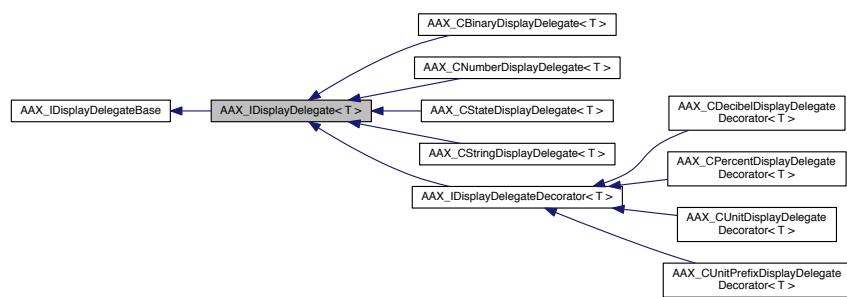
The documentation for this class was generated from the following file:

- [AAX_IDescriptionHost.h](#)

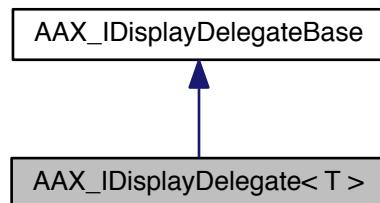
14.85 AAX_IDisplayDelegate< T > Class Template Reference

```
#include <AAX_IDisplayDelegate.h>
```

Inheritance diagram for AAX_IDisplayDelegate< T >:



Collaboration diagram for AAX_IDisplayDelegate< T >:



14.85.1 Description

```
template<typename T> class AAX_IDisplayDelegate< T >
```

Classes for parameter value string conversion.

Display delegate interface template

Display delegates are used to convert real parameter values to and from their formatted string representations. All display delegates implement the [AAX_IDisplayDelegate](#) interface, which contains two conversion functions:

```
virtual bool ValueToString(T value, std::string& valueString) const = 0;
virtual bool StringToValue(const std::string& valueString, T& value) const = 0;
```

14.85.2 Display delegate decorators

The AAX SDK utilizes a decorator pattern in order to provide code re-use while accounting for a wide variety of possible parameter display formats. The SDK includes a number of sample display delegate decorator classes.

Each concrete display delegate decorator implements [AAX_IDisplayDelegateDecorator](#) and adheres to the decorator pattern. The decorator pattern allows multiple display behaviors to be composited or wrapped together at run time. For instance it is possible to implement a dBV (dB Volts) decorator, by wrapping an [AAX_CDecibelDisplayDelegateDecorator](#) with an [AAX_CUnitDisplayDelegateDecorator](#).

14.85.2.1 Display delegate decorator implementation

By implementing [AAX_IDisplayDelegateDecorator](#), each concrete display delegate decorator class implements the full [AAX_IDisplayDelegate](#) interface. In addition, it retains a pointer to the [AAX_IDisplayDelegateDecorator](#) that it wraps. When the decorator performs a conversion, it calls into its wrapped class so that the wrapped decorator may apply its own conversion formatting. By repeating this pattern in each decorator, all of the decorator subclasses call into their "wrapper" in turn, resulting in a final string to which all of the decorators' conversions have been applied in sequence.

Here is the relevant implementation from [AAX_IDisplayDelegateDecorator](#) :

```
template <typename T>
AAX_IDisplayDelegateDecorator<T>::AAX_IDisplayDelegateDecorator
    (const AAX_IDisplayDelegate<T>& displayDelegate) :
    AAX_IDisplayDelegate<T>(),
    mWrappedDisplayDelegate(displayDelegate.Clone())
{
}

template <typename T>
bool      AAX_IDisplayDelegateDecorator<T>::ValueToString(
    T value, AAX_CString* valueString) const
{
    return mWrappedDisplayDelegate->ValueToString(value, valueString);
}

template <typename T>
bool      AAX_IDisplayDelegateDecorator<T>::StringToValue(
    const AAX_CString& valueString, T* value) const
{
    return mWrappedDisplayDelegate->StringToValue(valueString, value);
}
```

14.85.2.2 Decibel decorator example

Here is a concrete example of how a decibel decorator might be implemented

```
template <typename T>
bool      AAX_CDecibelDisplayDelegateDecorator<T>::ValueToString
    (T value, AAX_CString* valueString) const
{
    if (value <= 0)
    {
        *valueString = AAX_CString("--- dB");
        return true;
    }

    value = 20*log10(value);
    bool succeeded = AAX_IDisplayDelegateDecorator<T>::ValueToString
        (value, valueString);
    *valueString += AAX_CString("dB");
    return succeeded;
}
```

Notice in this example that the [ValueToString\(\)](#) method is called in the parent class, [AAX_IDisplayDelegateDecorator](#). This results in a call into the wrapped class' implementation of [ValueToString\(\)](#), which converts the decorated value to a redisplayed string, and so forth for additional decorators.

Public Member Functions

- virtual [AAX_IDisplayDelegate * Clone \(\) const =0](#)
Constructs and returns a copy of the display delegate.
- virtual bool [ValueToString \(T value, AAX_CString *valueString\) const =0](#)
Converts a real parameter value to a string representation.
- virtual bool [ValueToString \(T value, int32_t maxNumChars, AAX_CString *valueString\) const =0](#)
Converts a real parameter value to a string representation using a size hint, useful for control surfaces and other character limited displays.
- virtual bool [StringToValue \(const AAX_CString &valueString, T *value\) const =0](#)
Converts a string to a real parameter value.

14.85.3 Member Function Documentation

14.85.3.1 template<typename T> virtual AAX_IDisplayDelegate* AAX_IDisplayDelegate< T >::Clone () const [pure virtual]

Constructs and returns a copy of the display delegate.

In general, this method's implementation can use a simple copy constructor:

```
template <typename T>
AAX_CSubclassDisplayDelegate<T>* AAX_CSubclassDisplayDelegate<T>::Clone () const
{
    return new AAX_CSubclassDisplayDelegate(*this);
}
```

Implemented in [AAX_IDisplayDelegateDecorator< T >](#), [AAX_CStateDisplayDelegate< T >](#), [AAX_CUnitPrefixDisplayDelegateDecorator< T >](#), [AAX_CBinaryDisplayDelegate< T >](#), [AAX_CPercentDisplayDelegateDecorator< T >](#), [AAX_CStringDisplayDelegate< T >](#), [AAX_CDecibelDisplayDelegateDecorator< T >](#), [AAX_CUnitDisplayDelegateDecorator< T >](#), and [AAX_CNumberDisplayDelegate< T, Precision, SpaceAfter >](#).

Referenced by [AAX_CParameter< T >::SetDisplayDelegate\(\)](#).

Here is the caller graph for this function:



14.85.3.2 template<typename T> virtual bool AAX_IDisplayDelegate< T >::ValueToString (T value, AAX_CString *valueString) const [pure virtual]

Converts a real parameter value to a string representation.

Parameters

in	<i>value</i>	The real parameter value that will be converted
out	<i>valueString</i>	A string corresponding to value

Return values

<i>true</i>	The string conversion was successful
<i>false</i>	The string conversion was unsuccessful

Implemented in [AAX_IDisplayDelegateDecorator< T >](#), [AAX_CStateDisplayDelegate< T >](#), [AAX_CUnitPrefixDisplayDelegateDecorator< T >](#), [AAX_CBinaryDisplayDelegate< T >](#), [AAX_CPercentDisplayDelegateDecorator< T >](#), [AAX_CStringDisplayDelegate< T >](#), [AAX_CDecibelDisplayDelegateDecorator< T >](#), [AAX_CUnitDisplayDelegateDecorator< T >](#), and [AAX_CNumberDisplayDelegate< T, Precision, SpaceAfter >](#).

14.85.3.3 template<typename T> virtual bool AAX_IDisplayDelegate< T >::ValueToString (T *value*, int32_t *maxNumChars*, AAX_CString * *valueString*) const [pure virtual]

Converts a real parameter value to a string representation using a size hint, useful for control surfaces and other character limited displays.

Parameters

in	<i>value</i>	The real parameter value that will be converted
in	<i>maxNumChars</i>	Size hint for the desired maximum number of characters in the string (not including null termination)
out	<i>valueString</i>	A string corresponding to value

Return values

<i>true</i>	The string conversion was successful
<i>false</i>	The string conversion was unsuccessful

Implemented in [AAX_IDisplayDelegateDecorator< T >](#), [AAX_CStateDisplayDelegate< T >](#), [AAX_CUnitPrefixDisplayDelegateDecorator< T >](#), [AAX_CBinaryDisplayDelegate< T >](#), [AAX_CPercentDisplayDelegateDecorator< T >](#), [AAX_CStringDisplayDelegate< T >](#), [AAX_CDecibelDisplayDelegateDecorator< T >](#), [AAX_CUnitDisplayDelegateDecorator< T >](#), and [AAX_CNumberDisplayDelegate< T, Precision, SpaceAfter >](#).

14.85.3.4 template<typename T> virtual bool AAX_IDisplayDelegate< T >::StringToValue (const AAX_CString & *valueString*, T * *value*) const [pure virtual]

Converts a string to a real parameter value.

Parameters

in	<i>valueString</i>	The string that will be converted
out	<i>value</i>	The real parameter value corresponding to valueString

Return values

<i>true</i>	The string conversion was successful
<i>false</i>	The string conversion was unsuccessful

Implemented in [AAX_IDisplayDelegateDecorator< T >](#), [AAX_CStateDisplayDelegate< T >](#), [AAX_CUnitPrefixDisplayDelegateDecorator< T >](#), [AAX_CBinaryDisplayDelegate< T >](#), [AAX_CPercentDisplayDelegateDecorator< T >](#), [AAX_CStringDisplayDelegate< T >](#), [AAX_CDecibelDisplayDelegateDecorator< T >](#), [AAX_CUnitDisplayDelegateDecorator< T >](#), and [AAX_CNumberDisplayDelegate< T, Precision, SpaceAfter >](#).

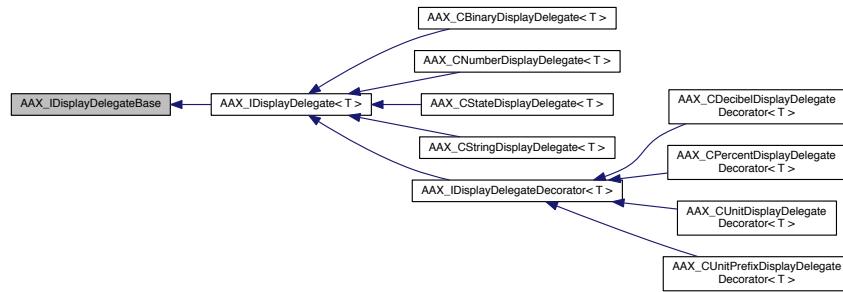
The documentation for this class was generated from the following file:

- [AAX_IDisplayDelegate.h](#)

14.86 AAX_IDisplayDelegateBase Class Reference

```
#include <AAx_IDisplayDelegate.h>
```

Inheritance diagram for AAX_IDisplayDelegateBase:



14.86.1 Description

Defines the display behavior for a parameter.

This interface represents a delegate class to be used in conjunction with [AAX_IParameter](#). [AAX_IParameter](#) delegates all conversion operations between strings and real parameter values to classes that meet this interface. You can think of [AAX_ITaperDelegate](#) subclasses as simple string serialization routines that enable a specific string conversions for an arbitrary parameter.

For more information about how parameter delegates operate, see the [AAX_ITaperDelegate](#) and [Parameter Manager](#) documentation.

Note

This class is *not* part of the AAX ABI and must not be passed between the plug-in and the host.

Public Member Functions

- virtual [~AAX_IDisplayDelegateBase \(\)](#)

Virtual destructor.

14.86.2 Constructor & Destructor Documentation

14.86.2.1 virtual AAX_IDisplayDelegateBase::~AAX_IDisplayDelegateBase() [inline], [virtual]

Virtual destructor.

Note

This destructor MUST be virtual to prevent memory leaks.

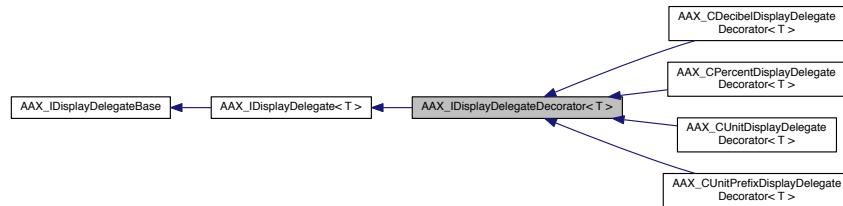
The documentation for this class was generated from the following file:

- [AAX_IDisplayDelegate.h](#)

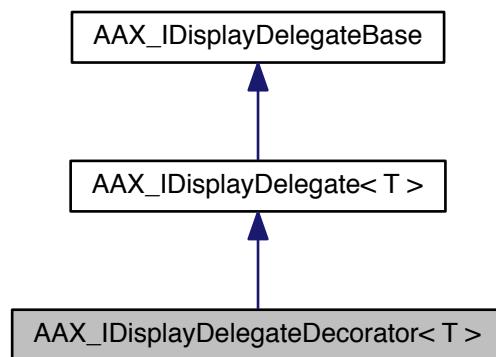
14.87 AAX_IDisplayDelegateDecorator< T > Class Template Reference

```
#include <AAX_IDisplayDelegateDecorator.h>
```

Inheritance diagram for AAX_IDisplayDelegateDecorator< T >:



Collaboration diagram for AAX_IDisplayDelegateDecorator< T >:



14.87.1 Description

```
template<typename T> class AAX_IDisplayDelegateDecorator< T >
```

The base class for all concrete display delegate decorators.

The AAX parameter display strategy uses a decorator pattern for parameter value formatting. This approach allows developers to maximize code re-use across display delegates with many different kinds of varying formatting, all without creating interdependencies between the different display delegates themselves.

For more information, see [Display delegate decorators](#). For even more information about the Decorator design pattern, please consult the GOF design patterns book.

Note

This class is *not* part of the AAX ABI and must not be passed between the plug-in and the host.

Public Member Functions

- [AAX_IDisplayDelegateDecorator](#) (const [AAX_IDisplayDelegate< T >](#) &displayDelegate)

Constructor.

- [AAX_IDisplayDelegateDecorator](#) (const [AAX_IDisplayDelegateDecorator](#) &other)

Copy constructor.

- [virtual ~AAX_IDisplayDelegateDecorator](#) ()

Virtual destructor.

- [virtual AAX_IDisplayDelegateDecorator< T > * Clone](#) () const [AAX_OVERRIDE](#)

Constructs and returns a copy of the display delegate decorator.

- [virtual bool ValueToString](#) (T value, [AAACString](#) *valueString) const [AAX_OVERRIDE](#)

Converts a string to a real parameter value.

- [virtual bool ValueToString](#) (T value, int32_t maxNumChars, [AAACString](#) *valueString) const [AAX_OVERRIDE](#)

Converts a string to a real parameter value with a size constraint.

- [virtual bool StringToValue](#) (const [AAACString](#) &valueString, T *value) const [AAX_OVERRIDE](#)

Converts a string to a real parameter value.

14.87.2 Constructor & Destructor Documentation

14.87.2.1 template<typename T > AAX_IDisplayDelegateDecorator< T >::AAX_IDisplayDelegateDecorator (const [AAX_IDisplayDelegate](#)< T > & displayDelegate)

Constructor.

This class implements the decorator pattern, which is a sort of wrapper. The object that is being wrapped is passed into this constructor. This object is passed by reference because it must be copied to prevent any potential memory ambiguities.

This constructor sets the local mWrappedDisplayDelegate member to a clone of the provided [AAX_IDisplayDelegate](#).

Parameters

in	displayDelegate	The decorated display delegate.
----	-----------------	---------------------------------

14.87.2.2 template<typename T > AAX_IDisplayDelegateDecorator< T >::AAX_IDisplayDelegateDecorator (const [AAX_IDisplayDelegateDecorator](#)< T > & other)

Copy constructor.

This class implements the decorator pattern, which is a sort of wrapper. The object that is being wrapped is passed into this constructor. This object is passed by reference because it must be copied to prevent any potential memory ambiguities.

This constructor sets the local mWrappedDisplayDelegate member to a clone of the provided [AAX_IDisplayDelegateDecorator](#), allowing multiply-decorated display delegates.

Parameters

in	other	The display delegate decorator that will be set as the wrapped delegate of this object
----	-------	--

14.87.2.3 template<typename T > AAX_IDisplayDelegateDecorator< T >::~AAX_IDisplayDelegateDecorator () [virtual]

Virtual destructor.

Note

This destructor must be overridden here in order to delete the wrapped display delegate object upon decorator destruction.

14.87.3 Member Function Documentation

14.87.3.1 template<typename T> AAX_IDisplayDelegateDecorator<T> * AAX_IDisplayDelegateDecorator<T>::Clone() const [virtual]

Constructs and returns a copy of the display delegate decorator.

In general, this method's implementation can use a simple copy constructor:

```
template <typename T>
AAX_CSubclassDisplayDelegate<T>* AAX_CSubclassDisplayDelegate<T>::Clone() const
{
    return new AAX_CSubclassDisplayDelegate(*this);
}
```

Note

This is an idiomatic method in the decorator pattern, so watch for potential problems if this method is ever changed or removed.

Implements [AAX_IDisplayDelegate< T >](#).

Reimplemented in [AAX_CUnitPrefixDisplayDelegateDecorator< T >](#), [AAX_CPercentDisplayDelegateDecorator< T >](#), [AAX_CDecibelDisplayDelegateDecorator< T >](#), and [AAX_CUnitDisplayDelegateDecorator< T >](#).

14.87.3.2 template<typename T> bool AAX_IDisplayDelegateDecorator<T>::ValueToString(T value, AAX_CString * valueString) const [virtual]

Converts a string to a real parameter value.

Override of the [AAX_IDisplayDelegate](#) implementation to call into the wrapped object. Display delegate decorators should call into this implementation to pass [ValueToString\(\)](#) calls on to the wrapped object after applying their own value-to-string decoration.

Parameters

in	<i>valueString</i>	The string that will be converted
out	<i>value</i>	The real parameter value corresponding to <i>valueString</i>

Return values

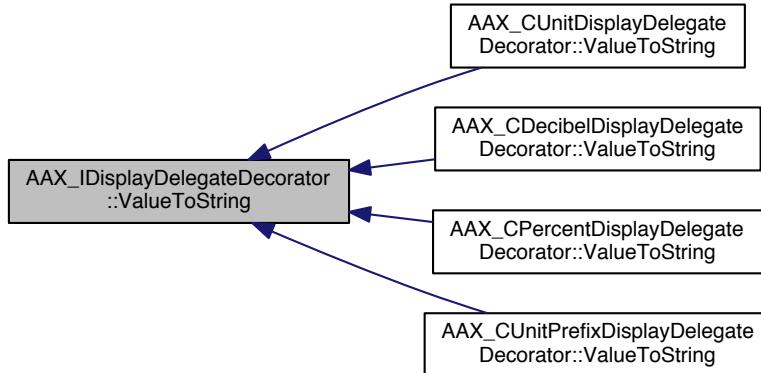
<i>true</i>	The string conversion was successful
<i>false</i>	The string conversion was unsuccessful

Implements [AAX_IDisplayDelegate< T >](#).

Reimplemented in [AAX_CUnitPrefixDisplayDelegateDecorator< T >](#), [AAX_CPercentDisplayDelegateDecorator< T >](#), [AAX_CDecibelDisplayDelegateDecorator< T >](#), and [AAX_CUnitDisplayDelegateDecorator< T >](#).

Referenced by [AAX_CUnitDisplayDelegateDecorator< T >::ValueToString\(\)](#), [AAX_CDecibelDisplayDelegateDecorator< T >::ValueToString\(\)](#), [AAX_CPercentDisplayDelegateDecorator< T >::ValueToString\(\)](#), and [AAX_CUnitPrefixDisplayDelegateDecorator< T >::ValueToString\(\)](#).

Here is the caller graph for this function:



14.87.3.3 template<typename T > bool AAX_IDisplayDelegateDecorator< T >::ValueToString (T value, int32_t maxNumChars, AAX_CString * valueString) const [virtual]

Converts a string to a real parameter value with a size constraint.

Override of the [AAX_IDisplayDelegate](#) implementation to call into the wrapped object. Display delegate decorators should call into this implementation to pass [ValueToString\(\)](#) calls on to the wrapped object after applying their own value-to-string decoration.

Parameters

in	valueString	The string that will be converted
in	maxNumChars	Size hint for the desired maximum number of characters in the string (not including null termination)
out	value	The real parameter value corresponding to valueString

Return values

true	The string conversion was successful
false	The string conversion was unsuccessful

Implements [AAX_IDisplayDelegate< T >](#).

Reimplemented in [AAX_CUnitPrefixDisplayDelegateDecorator< T >](#), [AAX_CPercentDisplayDelegateDecorator< T >](#), [AAX_CDecibelDisplayDelegateDecorator< T >](#), and [AAX_CUnitDisplayDelegateDecorator< T >](#).

14.87.3.4 template<typename T > bool AAX_IDisplayDelegateDecorator< T >::StringToValue (const AAX_CString & valueString, T * value) const [virtual]

Converts a string to a real parameter value.

Override of the DisplayDecorator implementation to call into the wrapped object. Display delegate decorators should call into this implementation to pass [StringToValue\(\)](#) calls on to the wrapped object after applying their own string-to-value decoding.

Parameters

in	<i>valueString</i>	The string that will be converted
out	<i>value</i>	The real parameter value corresponding to <i>valueString</i>

Return values

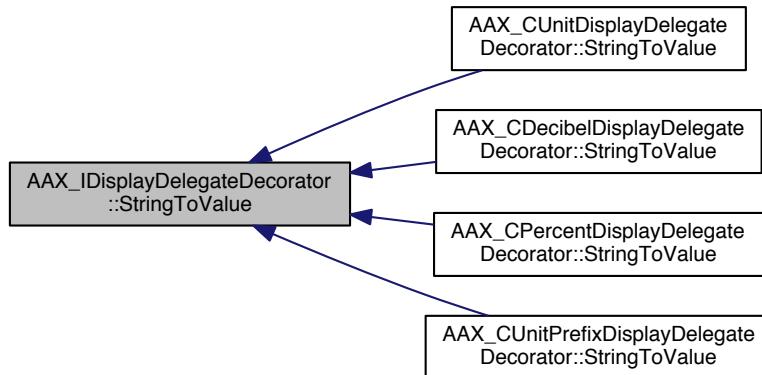
<i>true</i>	The string conversion was successful
<i>false</i>	The string conversion was unsuccessful

Implements [AAX_IDisplayDelegate< T >](#).

Reimplemented in [AAX_CUnitPrefixDisplayDelegateDecorator< T >](#), [AAX_CPercentDisplayDelegateDecorator< T >](#), [AAX_CDecibelDisplayDelegateDecorator< T >](#), and [AAX_CUnitDisplayDelegateDecorator< T >](#).

Referenced by [AAX_CUnitDisplayDelegateDecorator< T >](#)::StringToValue(), [AAX_CDecibelDisplayDelegateDecorator< T >](#)::StringToValue(), [AAX_CPercentDisplayDelegateDecorator< T >](#)::StringToValue(), and [AAX_CUnitPrefixDisplayDelegateDecorator< T >](#)::StringToValue().

Here is the caller graph for this function:



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

- [AAX_IDisplayDelegateDecorator.h](#)

14.88 AAX_IDma Class Reference

```
#include <AAX_IDma.h>
```

14.88.1 Description

Cross-platform interface for access to the host's direct memory access (DMA) facilities.

:Implemented by the AAX Host

This interface is provided via a DMA port in the plug-in's algorithm context.

See also

[AAX_IComponentDescriptor::AddDmaInstance\(\)](#)
Direct Memory Access

Public Types

- enum **EState** { **eState_Error** = -1, **eState_Init** = 0, **eState_Running** = 1, **eState_Complete** = 2, **eState_Pending** = 3 }
 - enum **EMode** { **eMode_Error** = -1, **eMode_Burst** = 6, **eMode_Gather** = 10, **eMode_Scatter** = 11 }
- DMA mode IDs.*

Public Member Functions

- virtual ~[AAX_IDma](#) ()

Basic DMA operation

- virtual [AAX_Result AAX_DMA_API PostRequest](#) ()=0
Posts the transfer request to the DMA server.
- virtual int32_t [AAX_DMA_API IsTransferComplete](#) ()=0
Query whether a transfer has completed.
- virtual [AAX_Result AAX_DMA_API SetDmaState](#) (**EState** iState)=0
Sets the DMA State.
- virtual [EState AAX_DMA_API GetDmaState](#) () const =0
Inquire to find the state of the DMA instance.
- virtual [EMode AAX_DMA_API GetDmaMode](#) () const =0
Inquire to find the mode of the DMA instance.

Methods for Burst operation

Use these methods in conjunction with [AAX_IDma::eMode_Burst](#)

- virtual [AAX_Result AAX_DMA_API SetSrc](#) (int8_t *iSrc)=0
Sets the address of the source buffer.
- virtual int8_t *[AAX_DMA_API GetSrc](#) ()=0
Gets the address of the source buffer.
- virtual [AAX_Result AAX_DMA_API SetDst](#) (int8_t *iDst)=0
Sets the address of the destination buffer.
- virtual int8_t *[AAX_DMA_API GetDst](#) ()=0
Gets the address of the destination buffer.
- virtual [AAX_Result AAX_DMA_API SetBurstLength](#) (int32_t iBurstLengthBytes)=0
Sets the length of each burst.
- virtual int32_t [AAX_DMA_API GetBurstLength](#) ()=0
Gets the length of each burst.
- virtual [AAX_Result AAX_DMA_API SetNumBursts](#) (int32_t iNumBursts)=0
Sets the number of bursts to perform before giving up priority to other DMA transfers.
- virtual int32_t [AAX_DMA_API GetNumBursts](#) ()=0
Gets the number of bursts to perform before giving up priority to other DMA transfers.
- virtual [AAX_Result AAX_DMA_API SetTransferSize](#) (int32_t iTransferSizeBytes)=0
Sets the size of the whole transfer.
- virtual int32_t [AAX_DMA_API GetTransferSize](#) ()=0
Gets the size of the whole transfer, in Bytes.

Methods for Scatter and Gather operation

Use these methods in conjunction with [AAX_IDma::eMode_Scatter](#) and [AAX_IDma::eMode_Gather](#)

- virtual [AAX_Result AAX_DMA_API SetFifoBuffer](#) (int8_t *iFifoBase)=0

- `virtual int8_t *AAX_DMA_API GetFifoBuffer ()=0`
Gets the address of the FIFO buffer for the DMA transfer.
- `virtual AAX_Result AAX_DMA_API SetLinearBuffer (int8_t *iLinearBase)=0`
Sets the address of the linear buffer for the DMA transfer (usually the internal memory block)
- `virtual int8_t *AAX_DMA_API GetLinearBuffer ()=0`
Gets the address of the linear buffer for the DMA transfer.
- `virtual AAX_Result AAX_DMA_API SetOffsetTable (const int32_t *iOffsetTable)=0`
Sets the offset table for the DMA transfer.
- `virtual const int32_t *AAX_DMA_API GetOffsetTable ()=0`
Gets the offset table for the DMA transfer.
- `virtual AAX_Result AAX_DMA_API SetNumOffsets (int32_t iNumOffsets)=0`
Sets the number of offsets in the offset table.
- `virtual int32_t AAX_DMA_API GetNumOffsets ()=0`
Gets the number of offsets in the offset table.
- `virtual AAX_Result AAX_DMA_API SetBaseOffset (int32_t iBaseOffsetBytes)=0`
Sets the relative base offset into the FIFO where transfers will begin.
- `virtual int32_t AAX_DMA_API GetBaseOffset ()=0`
Gets the relative base offset into the FIFO where transfers will begin.
- `virtual AAX_Result AAX_DMA_API SetFifoSize (int32_t iSizeBytes)=0`
Sets the size of the FIFO buffer, in bytes.
- `virtual int32_t AAX_DMA_API GetFifoSize ()=0`
Gets the size of the FIFO buffer, in bytes.

14.88.2 Member Enumeration Documentation

14.88.2.1 enum AAX_IDma::EState

Enumerator

- eState_Error**
- eState_Init**
- eState_Running**
- eState_Complete**
- eState_Pending**

14.88.2.2 enum AAX_IDma::EMode

DMA mode IDs.

These IDs are used to bind DMA context fields to a particular DMA mode when describing the fields with [AAX_IComponentDescriptor::AddDmaInstance\(\)](#)

Enumerator

- eMode_Error**
- eMode_Burst** Burst mode (uncommon)
- eMode_Gather** Gather mode.
- eMode_Scatter** Scatter mode.

14.88.3 Constructor & Destructor Documentation

14.88.3.1 virtual AAX_IDma::~AAX_IDma() [inline], [virtual]

14.88.4 Member Function Documentation

14.88.4.1 virtual **AAX_Result AAX_DMA_API** AAX_IDma::PostRequest() [pure virtual]

Posts the transfer request to the DMA server.

Note

Whichever mode this method is called on first will be the first mode to start transferring. Most plug-ins should therefore call this method for their Scatter DMA fields before their Gather DMA fields so that the scattered data is available as quickly as possible for future gathers.

Returns

AAX_SUCCESS on success

14.88.4.2 virtual int32_t AAX_DMA_API AAX_IDma::IsTransferComplete() [pure virtual]

Query whether a transfer has completed.

A return value of false indicates an error, and that the DMA missed its cycle count deadline

Note

This function should not be used for polling within a Process loop! Instead, it can be used as a test for DMA failure. This test is usually performed via a Debug-only assert.

Todo Clarify return value meaning – ambiguity in documentation

Returns

true if all pending transfers are complete
false if pending transfers are not complete

14.88.4.3 virtual **AAX_Result AAX_DMA_API** AAX_IDma::SetDmaState(EState iState) [pure virtual]

Sets the DMA State.

Note

This method is part of the host interface and should not be used by plug-ins

Returns

AAX_SUCCESS on success

14.88.4.4 virtual **EState AAX_DMA_API** AAX_IDma::GetDmaState() const [pure virtual]

Inquire to find the state of the DMA instance.

14.88.4.5 virtual **EMode AAX_DMA_API** AAX_IDma::GetDmaMode() const [pure virtual]

Inquire to find the mode of the DMA instance.

This value does not change, so there is no setter.

14.88.4.6 virtual AAX_Result AAX_DMA_API AAX_IDma::SetSrc(int8_t * *iSrc*) [pure virtual]

Sets the address of the source buffer.

Parameters

in	<i>iSrc</i>	Address of the location in the source buffer where the read transfer should begin
----	-------------	---

Returns

AAX_SUCCESS on success

14.88.4.7 virtual int8_t* AAX_DMA_API AAX_IDma::GetSrc() [pure virtual]

Gets the address of the source buffer.

14.88.4.8 virtual AAX_Result AAX_DMA_API AAX_IDma::SetDst(int8_t * *iDst*) [pure virtual]

Sets the address of the destination buffer.

Parameters

in	<i>iDst</i>	Address of the location in the destination buffer where the write transfer should begin
----	-------------	---

Returns

AAX_SUCCESS on success

14.88.4.9 virtual int8_t* AAX_DMA_API AAX_IDma::GetDst() [pure virtual]

Gets the address of the destination buffer.

14.88.4.10 virtual AAX_Result AAX_DMA_API AAX_IDma::SetBurstLength(int32_t *iBurstLengthBytes*) [pure virtual]

Sets the length of each burst.

Note

Burst length must be between 1 and 64 Bytes, inclusive
64-Byte transfers are recommended for the fastest overall transfer speed

Returns

AAX_SUCCESS on success

14.88.4.11 virtual int32_t AAX_DMA_API AAX_IDma::GetBurstLength() [pure virtual]

Gets the length of each burst.

14.88.4.12 virtual AAX_Result AAX_DMA_API AAX_IDma::SetNumBursts(int32_t *iNumBursts*) [pure virtual]

Sets the number of bursts to perform before giving up priority to other DMA transfers.

Valid values are 1, 2, 4, or 16.

The full transmission may be broken up into several series of bursts, and thus the total size of the data being transferred is not bounded by the number of bursts times the burst length.

Parameters

in	<i>iNumBursts</i>	The number of bursts
----	-------------------	----------------------

Returns

AAX_SUCCESS on success

14.88.4.13 virtual int32_t AAX_DMA_API AAX_IDma::GetNumBursts() [pure virtual]

Gets the number of bursts to perform before giving up priority to other DMA transfers.

14.88.4.14 virtual AAX_Result AAX_DMA_API AAX_IDma::SetTransferSize(int32_t *iTransferSizeBytes*) [pure virtual]

Sets the size of the whole transfer.

Parameters

in	<i>iTransferSizeBytes</i>	The transfer size, in Bytes
----	---------------------------	-----------------------------

Returns

AAX_SUCCESS on success

14.88.4.15 virtual int32_t AAX_DMA_API AAX_IDma::GetTransferSize() [pure virtual]

Gets the size of the whole transfer, in Bytes.

14.88.4.16 virtual AAX_Result AAX_DMA_API AAX_IDma::SetFifoBuffer(int8_t * *iFifoBase*) [pure virtual]

Sets the address of the FIFO buffer for the DMA transfer (usually the external memory block)

Returns

AAX_SUCCESS on success

14.88.4.17 virtual int8_t* AAX_DMA_API AAX_IDma::GetFifoBuffer() [pure virtual]

Gets the address of the FIFO buffer for the DMA transfer.

14.88.4.18 virtual AAX_Result AAX_DMA_API AAX_IDma::SetLinearBuffer(int8_t * *iLinearBase*) [pure virtual]

Sets the address of the linear buffer for the DMA transfer (usually the internal memory block)

Returns

AAX_SUCCESS on success

14.88.4.19 virtual int8_t* AAX_DMA_API AAX_IDma::GetLinearBuffer() [pure virtual]

Gets the address of the linear buffer for the DMA transfer.

```
14.88.4.20 virtual AAX_Result AAX_DMA_API AAX_IDma::SetOffsetTable( const int32_t * iOffsetTable ) [pure  
virtual]
```

Sets the offset table for the DMA transfer.

The offset table provides a list of Byte-aligned memory offsets into the FIFO buffer. The transfer will be broken into a series of individual bursts, each beginning at the specified offset locations within the FIFO buffer. The size of each burst is set by [SetBurstLength\(\)](#).

See also

[AAX_IDma::SetNumOffsets\(\)](#)
[AAX_IDma::SetBaseOffset\(\)](#)

Returns

AAX_SUCCESS on success

```
14.88.4.21 virtual const int32_t* AAX_DMA_API AAX_IDma::GetOffsetTable( ) [pure virtual]
```

Gets the offset table for the DMA transfer.

```
14.88.4.22 virtual AAX_Result AAX_DMA_API AAX_IDma::SetNumOffsets( int32_t iNumOffsets ) [pure  
virtual]
```

Sets the number of offsets in the offset table.

See also

[AAX_IDma::SetOffsetTable\(\)](#)

Returns

AAX_SUCCESS on success

```
14.88.4.23 virtual int32_t AAX_DMA_API AAX_IDma::GetNumOffsets( ) [pure virtual]
```

Gets the number of offsets in the offset table.

```
14.88.4.24 virtual AAX_Result AAX_DMA_API AAX_IDma::SetBaseOffset( int32_t iBaseOffsetBytes ) [pure  
virtual]
```

Sets the relative base offset into the FIFO where transfers will begin.

The base offset will be added to each value in the offset table in order to determine the starting offset within the FIFO buffer for each burst.

See also

[AAX_IDma::SetOffsetTable\(\)](#)

Returns

AAX_SUCCESS on success

14.88.4.25 virtual int32_t AAX_DMA_API AAX_IDma::GetBaseOffset() [pure virtual]

Gets the relative base offset into the FIFO where transfers will begin.

14.88.4.26 virtual AAX_Result AAX_DMA_API AAX_IDma::SetFifoSize(int32_t iSizeBytes) [pure virtual]

Sets the size of the FIFO buffer, in bytes.

Note

The FIFO buffer must be padded with at least enough memory to accommodate one burst, as defined by [SetBurstLength\(\)](#).

Returns

AAX_SUCCESS on success

14.88.4.27 virtual int32_t AAX_DMA_API AAX_IDma::GetFifoSize() [pure virtual]

Gets the size of the FIFO buffer, in bytes.

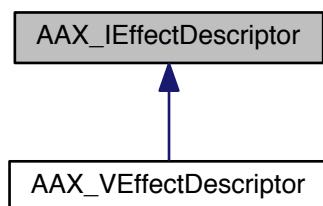
The documentation for this class was generated from the following file:

- [AAX_IDma.h](#)

14.89 AAX_IEffectDescriptor Class Reference

```
#include <AAX_IEffectDescriptor.h>
```

Inheritance diagram for AAX_IEffectDescriptor:



14.89.1 Description

Description interface for an effect's (plug-in type's) components.

[Implemented by the AAX Host](#)

Each Effect represents a different "type" of plug-in. The host will present different Effects to the user as separate products, even if they are derived from the same [AAX_ICollection](#) description.

See also

[AAX_ICollection::AddEffect\(\)](#)

Public Member Functions

- virtual [~AAX_IEffectDescriptor \(\)](#)
- virtual [AAX_IComponentDescriptor * NewComponentDescriptor \(\)=0](#)

Create an instance of a component descriptor.
- virtual [AAX_Result AddComponent \(AAX_IComponentDescriptor *inComponentDescriptor\)=0](#)

Add a component to an instance of a component descriptor.
- virtual [AAX_Result AddName \(const char *inPlugInName\)=0](#)

Add a name to the Effect.
- virtual [AAX_Result AddCategory \(uint32_t inCategory\)=0](#)

Add a category to your plug-in. See [AAX_EPlugInCategory](#).
- virtual [AAX_Result AddCategoryBypassParameter \(uint32_t inCategory, AAX_CParamID inParamID\)=0](#)

Add a category to your plug-in. See [AAX_EPlugInCategory](#).
- virtual [AAX_Result AddProcPtr \(void *inProcPtr, AAX_CProcPtrID inProcID\)=0](#)

Add a process pointer.
- virtual [AAX_IPropertyMap * NewPropertyMap \(\)=0](#)

Create a new property map.
- virtual [AAX_Result SetProperties \(AAX_IPropertyMap *inProperties\)=0](#)

Set the properties of a new property map.
- virtual [AAX_Result AddResourceInfo \(AAX_EResourceType in ResourceType, const char *inInfo\)=0](#)

Set resource file info.
- virtual [AAX_Result AddMeterDescription \(AAX_CTypeID inMeterID, const char *inMeterName, AAX_IPropertyMap *inProperties\)=0](#)

Add name and property map to meter with given ID.
- virtual [AAX_Result AddControlMIDIINode \(AAX_CTypeID inNodeID, AAX_EMIDIINodeType inNodeType, const char innodeName\[\], uint32_t inChannelMask\)=0](#)

Add a control MIDI node to the plug-in data model.

14.89.2 Constructor & Destructor Documentation

14.89.2.1 virtual [AAX_IEffectDescriptor::~AAX_IEffectDescriptor \(\)](#) [inline], [virtual]

14.89.3 Member Function Documentation

14.89.3.1 virtual [AAX_IComponentDescriptor* AAX_IEffectDescriptor::NewComponentDescriptor \(\)](#) [pure virtual]

Create an instance of a component descriptor.

Implemented in [AAX_VEffectDescriptor](#).

Referenced by [AAX_CMonolithicParameters::StaticDescribe\(\)](#).

Here is the caller graph for this function:



14.89.3.2 virtual AAX_Result AAX_IEffectDescriptor::AddComponent (AAX_IComponentDescriptor * *inComponentDescriptor*) [pure virtual]

Add a component to an instance of a component descriptor.

Unlike with [AAX_ICollection::AddEffect\(\)](#), the [AAX_IEffectDescriptor](#) does not take ownership of the [AAX_IComponentDescriptor](#) that is passed to it in this method. The host copies out the contents of this descriptor, and thus the plug-in may re-use the same descriptor object when creating additional similar components.

Parameters

in	<i>inComponentDescriptor</i>	
----	------------------------------	--

Implemented in [AAX_VEffectDescriptor](#).

Referenced by [AAX_CMonolithicParameters::StaticDescribe\(\)](#).

Here is the caller graph for this function:



14.89.3.3 virtual AAX_Result AAX_IEffectDescriptor::AddName (const char * *inPlugInName*) [pure virtual]

Add a name to the Effect.

May be called multiple times to add abbreviated Effect names.

Note

Every Effect must include at least one name variant with 31 or fewer characters, plus a null terminating character

Parameters

in	<i>inPlugInName</i>	The name assigned to the plug-in.
----	---------------------	-----------------------------------

Implemented in [AAX_VEffectDescriptor](#).

14.89.3.4 virtual AAX_Result AAX_IEffectDescriptor::AddCategory(uint32_t *inCategory*) [pure virtual]

Add a category to your plug-in. See [AAX_EPlugInCategory](#).

Parameters

in	<i>inCategory</i>	One of the categories for the plug-in.
----	-------------------	--

Implemented in [AAX_VEffectDescriptor](#).

14.89.3.5 virtual AAX_Result AAX_IEffectDescriptor::AddCategoryBypassParameter(uint32_t *inCategory*, AAX_CParamID *inParamID*) [pure virtual]

Add a category to your plug-in. See [AAX_EPlugInCategory](#).

Parameters

in	<i>inCategory</i>	One of the categories for the plug-in.
in	<i>inParamID</i>	The parameter ID of the parameter used to bypass the category section of the plug-in.

Implemented in [AAX_VEffectDescriptor](#).

14.89.3.6 virtual AAX_Result AAX_IEffectDescriptor::AddProcPtr(void * *inProcPtr*, AAX_CProcPtrID *inProcID*) [pure virtual]

Add a process pointer.

Parameters

in	<i>inProcPtr</i>	A process pointer.
in	<i>inProcID</i>	A process ID.

Implemented in [AAX_VEffectDescriptor](#).

14.89.3.7 virtual AAX_IPropertyMap* AAX_IEffectDescriptor::NewPropertyMap() [pure virtual]

Create a new property map.

Implemented in [AAX_VEffectDescriptor](#).

14.89.3.8 virtual AAX_Result AAX_IEffectDescriptor::SetProperties(AAX_IPropertyMap * *inProperties*) [pure virtual]

Set the properties of a new property map.

Parameters

in	<i>inProperties</i>	Description
----	---------------------	-------------

Implemented in [AAX_VEffectDescriptor](#).

14.89.3.9 virtual AAX_Result AAX_IEffectDescriptor::AddResourceInfo (AAX_EResourceType *in ResourceType*, const char * *inInfo*) [pure virtual]

Set resource file info.

Parameters

in	<i>inResourceType</i>	See AAX_EResourceType.
in	<i>inInfo</i>	Definition varies on the resource type.

Implemented in [AAX_VEffectDescriptor](#).

14.89.3.10 virtual AAX_Result AAX_IEffectDescriptor::AddMeterDescription (AAX_CTypeID *inMeterID*, const char * *inMeterName*, AAX_IPropertyMap * *inProperties*) [pure virtual]

Add name and property map to meter with given ID.

Parameters

in	<i>inMeterID</i>	The ID of the meter being described.
in	<i>inMeterName</i>	The name of the meter.
in	<i>inProperties</i>	The property map containing meter related data such as meter type, orientation, etc.

Implemented in [AAX_VEffectDescriptor](#).

14.89.3.11 virtual AAX_Result AAX_IEffectDescriptor::AddControlMIDINode (AAX_CTypeID *inNodeID*, AAX_EMIDINodeType *inNodeType*, const char *innodeName*[], uint32_t *inChannelMask*) [pure virtual]

Add a control MIDI node to the plug-in data model.

- This MIDI node may receive note data as well as control data.
- To send MIDI data to the plug-in's algorithm, use [AAX_IComponentDescriptor::AddMIDINode\(\)](#).

See also

[AAX_IACFEffectorParameters_V2::UpdateControlMIDINodes\(\)](#)

Parameters

in	<i>inNodeID</i>	The ID for the new control MIDI node.
in	<i>inNodeType</i>	The type of the node.
in	<i>innodeName</i>	The name of the node.
in	<i>inChannelMask</i>	The bit mask for required nodes channels (up to 16) or required global events for global node.

Implemented in [AAX_VEffectDescriptor](#).

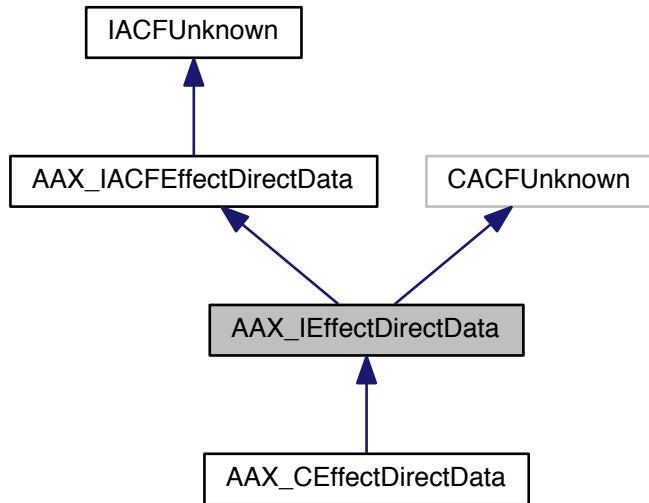
The documentation for this class was generated from the following file:

- [AAX_IEffectDescriptor.h](#)

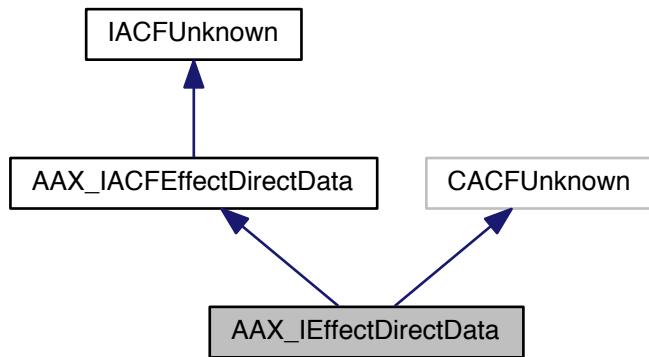
14.90 AAX_IEffectDirectData Class Reference

```
#include <AAX_IEffectDirectData.h>
```

Inheritance diagram for AAX_IEffectDirectData:



Collaboration diagram for AAX_IEffectDirectData:



14.90.1 Description

The interface for a AAX Plug-in's direct data interface.

:Implemented by the Plug-In

This is the interface for an instance of a plug-in's direct data interface that gets exposed to the host application. A plug-in needs to inherit from this interface and override all of the virtual functions to support direct data access functionality.

Direct data access allows a plug-in to directly manipulate the data in its algorithm's private data blocks. The callback methods in this interface provide a safe context from which this kind of access may be attempted.

Note

This class always inherits from the latest version of the interface and thus requires any subclass to implement all the methods in the latest version of the interface.

See [AAX_IACFEffctDirectData](#) for further information.

Public Member Functions

- [ACF_DECLARE_STANDARD_UNKNOWN \(\) ACFMETHOD\(InternalQueryInterface\)\(const acfIID &riid\)](#)
- [AAX_DELETE \(AAX_IEffectDirectData &operator=\(const AAX_IEffectDirectData &\)\)](#)

Public Attributes

- [void ** ppvObjOut](#)

14.90.2 Member Function Documentation

14.90.2.1 AAX_IEffectDirectData::ACF_DECLARE_STANDARD_UNKNOWN () const

14.90.2.2 AAX_IEffectDirectData::AAX_DELETE (AAX_IEffectDirectData & operator = (const AAX_IEffectDirectData &))

14.90.3 Member Data Documentation

14.90.3.1 void** AAX_IEffectDirectData::ppvObjOut

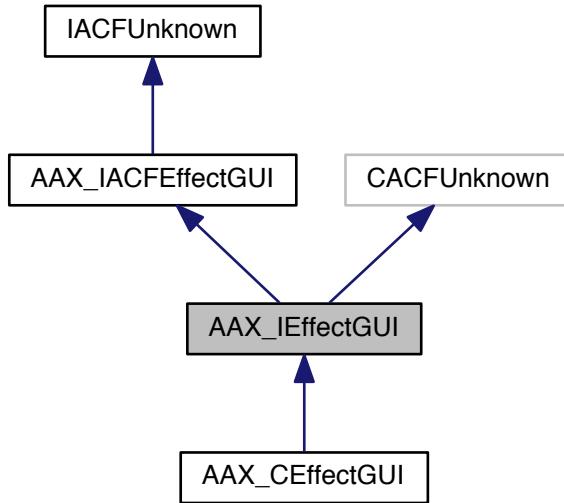
The documentation for this class was generated from the following file:

- [AAX_IEffectDirectData.h](#)

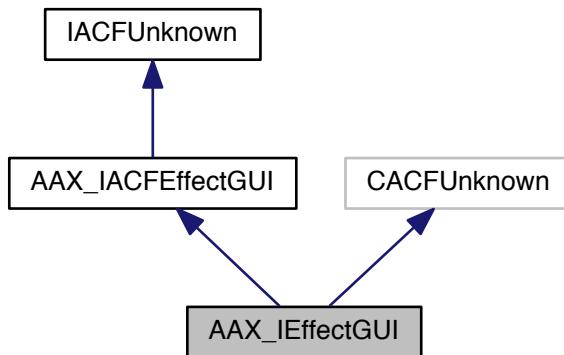
14.91 AAX_IEffectGUI Class Reference

```
#include <AAX_IEffectGUI.h>
```

Inheritance diagram for AAX_IEffectGUI:



Collaboration diagram for AAX_IEffectGUI:



14.91.1 Description

The interface for a AAX Plug-in's user interface.

:Implemented by the Plug-In

This is the interface for an instance of a plug-in's GUI that gets exposed to the host application. You need to inherit from this interface and override all of the virtual functions to create a plug-in GUI.

To create the GUI for an AAX plug-in it is required that you inherit from this interface and override all of the virtual functions from [AAX_IACFEffectGUI](#). In nearly all cases you will be able to take advantage of the implementations in the AAX library's [AAX_CEffectGUI](#) class and only override the few specific methods that you want to explicitly customize.

Note

This class always inherits from the latest version of the interface and thus requires any subclass to implement all the methods in the latest version of the interface.

Public Member Functions

- [ACF_DECLARE_STANDARD_UNKNOWN \(\) ACFMETHOD\(InternalQueryInterface\)\(const acfIID &riid\)](#)
- [AAX_DELETE \(AAX_IEffectGUI &operator=\(const AAX_IEffectGUI &\)\)](#)

Public Attributes

- [void ** ppvObjOut](#)

14.91.2 Member Function Documentation

14.91.2.1 AAX_IEffectGUI::ACF_DECLARE_STANDARD_UNKNOWN () const

14.91.2.2 AAX_IEffectGUI::AAX_DELETE (AAX_IEffectGUI & operator= (const AAX_IEffectGUI &))

14.91.3 Member Data Documentation

14.91.3.1 void** AAX_IEffectGUI::ppvObjOut

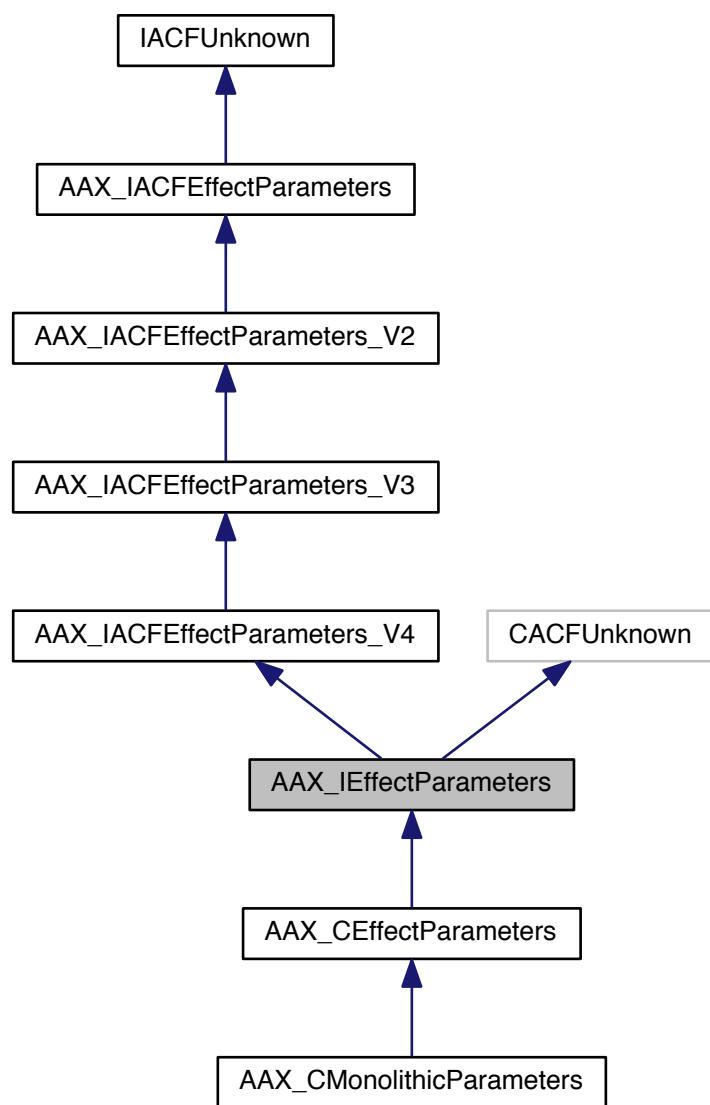
The documentation for this class was generated from the following file:

- [AAX_IEffectGUI.h](#)

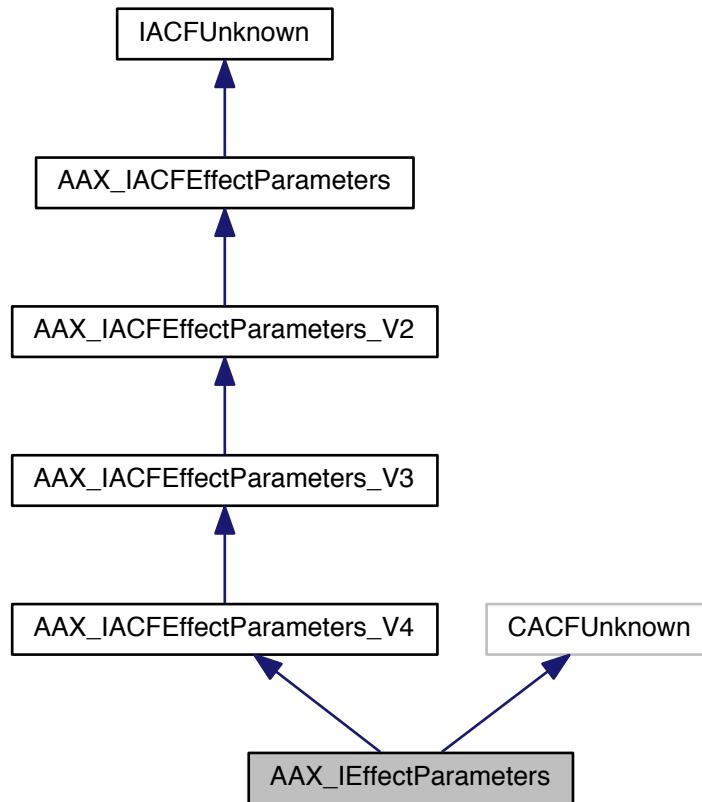
14.92 AAX_IEffectParameters Class Reference

```
#include <AAX_IEffectParameters.h>
```

Inheritance diagram for AAX_IEffectParameters:



Collaboration diagram for AAX_IEffectParameters:



14.92.1 Description

The interface for an AAX Plug-in's data model.

:Implemented by the Plug-In

The interface for an instance of a plug-in's data model. A plug-in's implementation of this interface is responsible for creating the plug-in's set of parameters and for defining how the plug-in will respond when these parameters are changed via control updates or preset loads. In order for information to be routed from the plug-in's data model to its algorithm, the parameters that are created here must be registered with the host in the plug-in's [Description callback](#).

At [initialization](#), the host provides this interface with a reference to [AAX_IController](#), which provides access from the data model back to the host. This reference provides a means of querying information from the host such as stem format or sample rate, and is also responsible for communication between the data model and the plug-in's (decoupled) algorithm. See [Real-time algorithm callback](#).

You will most likely inherit your implementation of this interface from [AAX_CEffectParameters](#), a default implementation that provides basic data model functionality such as adding custom parameters, setting control values, restoring state, generating coefficients, etc., which you can override and customize as needed.

The following tags appear in the descriptions for methods of this class and its derived classes:

- **CALL**: Components in the plug-in should call this method to get / set data in the data model.

Note

- This class always inherits from the latest version of the interface and thus requires any subclass to implement all the methods in the latest version of the interface. The current version of [AAX_CEffectParameters](#) provides a convenient default implementation for all methods in the latest interface.
- Except where noted otherwise, the parameter values referenced by the methods in this interface are normalized values. See [Parameter Manager](#) for more information.

Legacy Porting Notes In the legacy plug-in SDK, these methods were found in `CProcess` and `CEffectProcess`. For additional `CProcess` methods, see [AAX_IEffectGUI](#).

14.92.2 Related classes

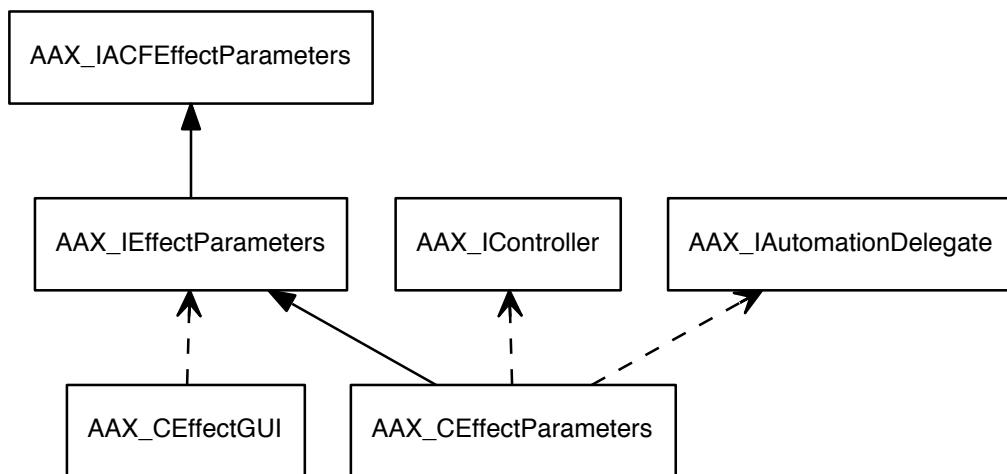


Figure 14.3: Classes related to `AAX_IEffectParameters` by inheritance or composition

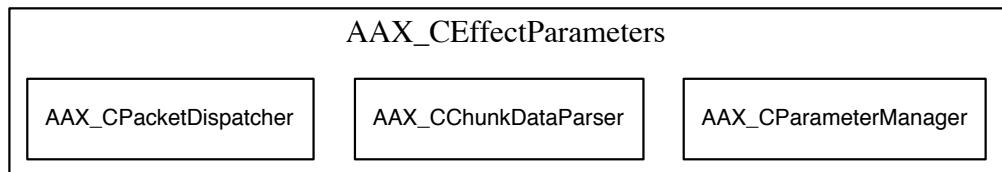


Figure 14.4: Classes owned as member objects of `AAX_CEffectParameters`

Public Member Functions

- `ACF_DECLARE_STANDARD_UNKNOWN () ACFMETHOD(InternalQueryInterface)(const acfIID &iid`

- [AAX_DELETE \(AAX_IEffectParameters &operator=\(const AAX_IEffectParameters &\)\)](#)

Public Attributes

- void ** [ppvObjOut](#)

14.92.3 Member Function Documentation

14.92.3.1 [AAX_IEffectParameters::ACF_DECLARE_STANDARD_UNKNOWN \(\) const](#)

14.92.3.2 [AAX_IEffectParameters::AAX_DELETE \(AAX_IEffectParameters & operator = \(const AAX_IEffectParameters &\) \)](#)

14.92.4 Member Data Documentation

14.92.4.1 [void** AAX_IEffectParameters::ppvObjOut](#)

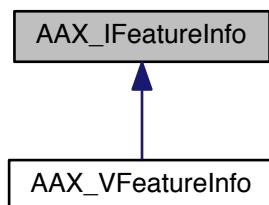
The documentation for this class was generated from the following file:

- [AAX_IEffectParameters.h](#)

14.93 AAX_IFeatureInfo Class Reference

#include <AAX_IFeatureInfo.h>

Inheritance diagram for AAX_IFeatureInfo:



Public Member Functions

- virtual [~AAX_IFeatureInfo \(\)](#)
- virtual [AAX_Result SupportLevel \(AAX_ESupportLevel &oSupportLevel\) const =0](#)
- virtual const [AAX_IPropertyMap * AcquireProperties \(\) const =0](#)
- virtual const [AAX_Feature_UID & ID \(\) const =0](#)

14.93.1 Constructor & Destructor Documentation

14.93.1.1 [virtual AAX_IFeatureInfo::~AAX_IFeatureInfo \(\) \[inline\], \[virtual\]](#)

14.93.2 Member Function Documentation

14.93.2.1 `virtual AAX_Result AAX_IFeatureInfo::SupportLevel(AAX_ESupportLevel & oSupportLevel) const [pure virtual]`

Determine the level of support for this feature by the host

Note

The host will not provide an underlying `AAX_IACFFeatureInfo` interface for features which it does not recognize at all, resulting in a `AAX_ERROR_NULL_OBJECT` error code

Implemented in `AAX_VFeatureInfo`.

14.93.2.2 `virtual const AAX_IPropertyMap* AAX_IFeatureInfo::AcquireProperties() const [pure virtual]`

Additional properties providing details of the feature support

See the feature's UID for documentation of which features provide additional properties

Ownership of the returned object is passed to the caller; the caller is responsible for destroying the object, e.g. by capturing the returned object in a smart pointer.

```
// AAX_IFeatureInfo* featureInfo
std::unique_ptr<const AAX_IPropertyMap> featurePropertiesPtr(featureInfo->AcquireProperties());
```

Returns

An `AAX_IPropertyMap` interface with access to the host's properties for this feature.
NULL if the desired feature was not found or if an error occurred

Note

May return an `AAX_IPropertyMap` object with limited method support, which would return an error such as `AAX_ERROR_NULL_OBJECT` or `AAX_ERROR_UNIMPLEMENTED` to interface calls.

Implemented in `AAX_VFeatureInfo`.

14.93.2.3 `virtual const AAX_Feature_UID& AAX_IFeatureInfo::ID() const [pure virtual]`

Returns the ID of the feature which this object represents

Implemented in `AAX_VFeatureInfo`.

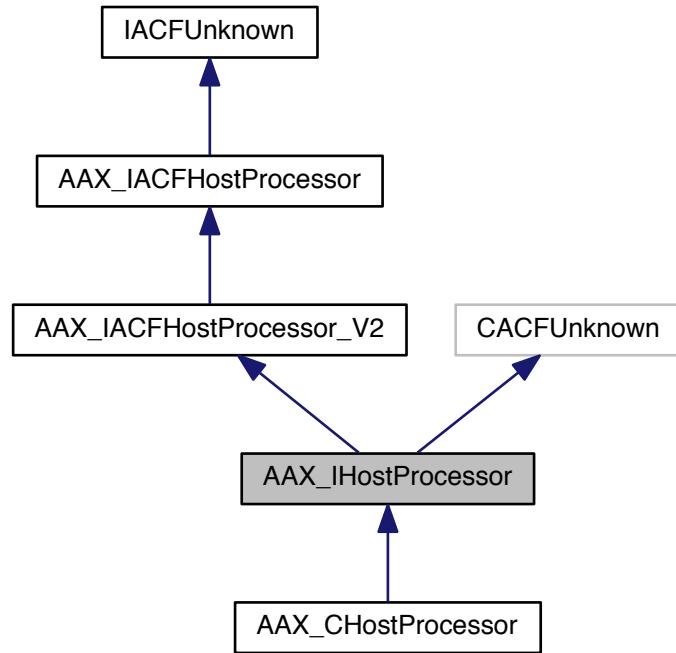
The documentation for this class was generated from the following file:

- `AAX_IFeatureInfo.h`

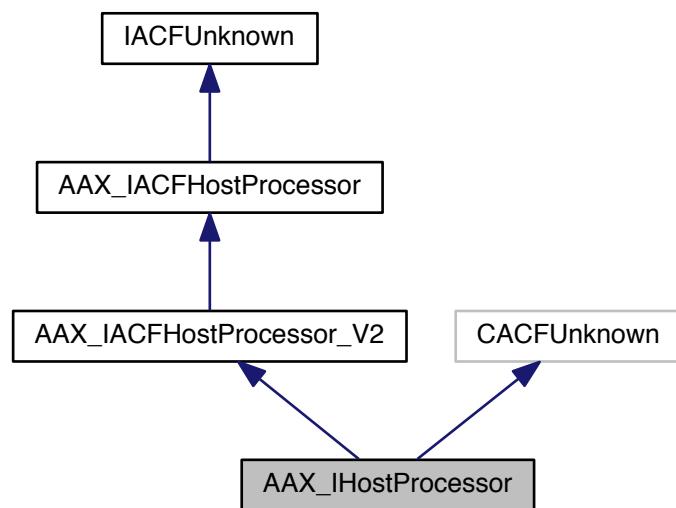
14.94 AAX_IHostProcessor Class Reference

```
#include <AAX_IHostProcessor.h>
```

Inheritance diagram for AAX_IHostProcessor:



Collaboration diagram for AAX_IHostProcessor:



14.94.1 Description

Base class for the host processor interface.

:Implemented by the Plug-In

Note

This class always inherits from the latest version of the interface and thus requires any subclass to implement all the methods in the latest version of the interface. Most plug-ins will inherit from the [AAX_CHostProcessor](#) convenience class.

Public Member Functions

- [ACF_DECLARE_STANDARD_UNKNOWN \(\) ACFMETHOD\(InternalQueryInterface\)\(const acfIID &riid\)](#)
- [AAX_DELETE \(AAX_IHostProcessor &operator=\(const AAX_IHostProcessor &\)\)](#)

Public Attributes

- `void ** ppvObjOut`

14.94.2 Member Function Documentation

14.94.2.1 AAX_IHostProcessor::ACF_DECLARE_STANDARD_UNKNOWN () const

14.94.2.2 AAX_IHostProcessor::AAX_DELETE (AAX_IHostProcessor & operator= (const AAX_IHostProcessor &))

14.94.3 Member Data Documentation

14.94.3.1 void** AAX_IHostProcessor::ppvObjOut

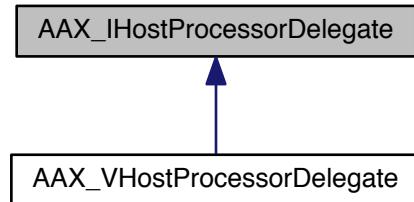
The documentation for this class was generated from the following file:

- [AAX_IHostProcessor.h](#)

14.95 AAX_IHostProcessorDelegate Class Reference

```
#include <AAX_IHostProcessorDelegate.h>
```

Inheritance diagram for AAX_IHostProcessorDelegate:



14.95.1 Description

Versioned interface for host methods specific to offline processing.

:Implemented by the AAX Host

The host provides a host processor delegate to a plug-in's [host processor](#) object at initialization. The host processor object may make calls to this object to get information about the current render pass or to affect the plug-in's offline processing behavior.

Public Member Functions

- virtual [~AAX_IHostProcessorDelegate \(\)](#)
- virtual [AAX_Result GetAudio \(const float *const inAudioIns\[\], int32_t inAudioInCount, int64_t inLocation, int32_t *ioNumSamples\)=0](#)

CALL: Randomly access audio from the timeline.
- virtual int32_t [GetSideChainInputNum \(\)=0](#)

CALL: Returns the index of the side chain input buffer.
- virtual [AAX_Result ForceAnalyze \(\)=0](#)

CALL: Request an analysis pass.
- virtual [AAX_Result ForceProcess \(\)=0](#)

CALL: Request a process pass.

14.95.2 Constructor & Destructor Documentation

14.95.2.1 virtual AAX_IHostProcessorDelegate::[~AAX_IHostProcessorDelegate \(\) \[inline\], \[virtual\]](#)

14.95.3 Member Function Documentation

14.95.3.1 virtual [AAX_Result AAX_IHostProcessorDelegate::GetAudio \(const float *const inAudioIns\[\], int32_t inAudioInCount, int64_t inLocation, int32_t * ioNumSamples \) \[pure virtual\]](#)

CALL: Randomly access audio from the timeline.

Called from within [AAX_IHostProcessor::RenderAudio\(\)](#), this method fills a buffer of samples with randomly-accessed data from the current input processing region on the timeline, including any extra samples such as processing "handles".

Note

Plug-ins that use this feature must set [AAX_eProperty_UsesRandomAccess](#) to true
 It is not possible to retrieve samples from outside of the current input processing region
 Always check the return value of this method before using the randomly-accessed samples

Parameters

<i>in</i>	<i>inAudioIns</i>	Timeline audio buffer(s). This must be set to <i>inAudioIns</i> from AAX_IHostProcessor::RenderAudio()
-----------	-------------------	--

Parameters

<i>in</i>	<i>inAudioInCount</i>	Number of buffers in <i>inAudioIns</i> . This must be set to <i>inAudioInCount</i> from AAX_IHostProcessor::RenderAudio()
-----------	-----------------------	---

Parameters

<i>in</i>	<i>inLocation</i>	A sample location relative to the beginning of the currently processed region, e.g. a value of 0 corresponds to the timeline location returned by AAX_CHostProcessor::GetSrcStart()
<i>in, out</i>	<i>ioNumSamples</i>	<ul style="list-style-type: none"> • Input: The maximum number of samples to read. • Output: The actual number of samples that were read from the timeline

Implemented in [AAX_VHostProcessorDelegate](#).

14.95.3.2 virtual int32_t AAX_IHostProcessorDelegate::GetSideChainInputNum() [pure virtual]

CALL: Returns the index of the side chain input buffer.

Called from within [AAX_IHostProcessor::RenderAudio\(\)](#), this method returns the index of the side chain input sample buffer within *inAudioIns*.

Implemented in [AAX_VHostProcessorDelegate](#).

14.95.3.3 virtual AAX_Result AAX_IHostProcessorDelegate::ForceAnalyze() [pure virtual]

CALL: Request an analysis pass.

Call this method to request an analysis pass from within the plug-in. Most plug-ins should rely on the host to trigger analysis passes when appropriate. However, plug-ins that require an analysis pass a) outside of the context of host-driven render or analysis, or b) when internal plug-in data changes may need to call [ForceAnalyze\(\)](#).

Implemented in [AAX_VHostProcessorDelegate](#).

14.95.3.4 virtual AAX_Result AAX_IHostProcessorDelegate::ForceProcess() [pure virtual]

CALL: Request a process pass.

Call this method to request a process pass from within the plug-in. If [AAX_eProperty_RequiresAnalysis](#) is defined, the resulting process pass will be preceded by an analysis pass. This method should only be used in rare circumstances by plug-ins that must launch processing outside of the normal host AudioSuite workflow.

Implemented in [AAX_VHostProcessorDelegate](#).

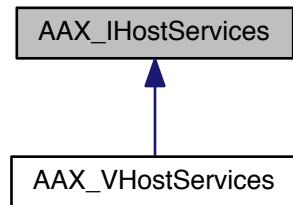
The documentation for this class was generated from the following file:

- [AAX_IHostProcessorDelegate.h](#)

14.96 AAX_IHostServices Class Reference

```
#include <AAX_IHostServices.h>
```

Inheritance diagram for AAX_IHostServices:



14.96.1 Description

Interface to diagnostic and debugging services provided by the AAX host.

:Implemented by the AAX Host

See also

[AAX_IACFHostServices](#)

Public Member Functions

- virtual [~AAX_IHostServices\(\)](#)
- virtual [AAX_Result HandleAssertFailure](#) (const char **iFile*, int32_t *iLine*, const char **iNote*, int32_t *iFlags*) const =0
Handle an assertion failure.
- virtual [AAX_Result Trace](#) (int32_t *iPriority*, const char **iMessage*) const =0
Log a trace message.
- virtual [AAX_Result StackTrace](#) (int32_t *iTracePriority*, int32_t *iStackTracePriority*, const char **iMessage*) const =0
Log a trace message or a stack trace.

14.96.2 Constructor & Destructor Documentation

14.96.2.1 virtual AAX_IHostServices::[~AAX_IHostServices\(\)](#) [inline], [virtual]

14.96.3 Member Function Documentation

14.96.3.1 virtual [AAX_Result AAX_IHostServices::HandleAssertFailure](#) (const char * *iFile*, int32_t *iLine*, const char * *iNote*, int32_t *iFlags*) const [pure virtual]

Handle an assertion failure.

Use this method to delegate assertion failure handling to the host

Use *inFlags* to request that specific behavior be included when handling the failure. This request may not be fulfilled by the host, and absence of a flag does not preclude the host from using that behavior when handling the failure.

Parameters

in	<i>iFile</i>	The name of the file containing the assert check. Usually <code>__FILE__</code>
in	<i>iLine</i>	The line number of the assert check. Usually <code>__LINE__</code>
in	<i>iNote</i>	Text to display related to the assert. Usually the condition which failed
in	<i>iFlags</i>	Bitfield of AAX_EAssertFlags to request specific handling behavior

Implemented in [AAX_VHostServices](#).

14.96.3.2 virtual AAX_Result AAX_IHostServices::Trace (int32_t *iPriority*, const char * *iMessage*) const [pure virtual]

Log a trace message.

Parameters

in	<i>iPriority</i>	Priority of the trace, used for log filtering. One of kAAX_Trace_Priority_Low , kAAX_Trace_Priority_Normal , kAAX_Trace_Priority_High
in	<i>iMessage</i>	Message string to log

Implemented in [AAX_VHostServices](#).

14.96.3.3 virtual AAX_Result AAX_IHostServices::StackTrace (int32_t *iTracePriority*, int32_t *iStackTracePriority*, const char * *iMessage*) const [pure virtual]

Log a trace message or a stack trace.

If the logging output filtering is set to include logs with *iStackTracePriority* then both the logging message and a stack trace will be emitted, regardless of *iTracePriority*.

If the logging output filtering is set to include logs with *iTracePriority* but to exclude logs with *iStackTracePriority* then this will emit a normal log with no stack trace.

Parameters

in	<i>iTracePriority</i>	Priority of the trace, used for log filtering. One of kAAX_Trace_Priority_Low , kAAX_Trace_Priority_Normal , kAAX_Trace_Priority_High
in	<i>iStackTracePriority</i>	Priority of the stack trace, used for log filtering. One of kAAX_Trace_Priority_Low , kAAX_Trace_Priority_Normal , kAAX_Trace_Priority_High
in	<i>iMessage</i>	Message string to log

Implemented in [AAX_VHostServices](#).

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

- [AAX_IHostServices.h](#)

14.97 AAX_IMIDIMessageInfoDelegate Class Reference

Public Member Functions

- virtual ~AAX_IMIDIMessageInfoDelegate ()
- virtual uint32_t [Mask](#) () const =0
- virtual uint32_t [Length](#) () const =0

- virtual [AAX_CString ToString](#) (uint32_t inLength, const uint8_t *inData) const =0
- virtual bool [Accepts](#) (uint32_t inLength, const uint8_t *inData) const

Protected Member Functions

- bool [Accepts_ExactStatus](#) (uint32_t inLength, const uint8_t *inData) const

Static Protected Member Functions

- static void [ToString_AppendNumber](#) (const char *inLabel, int32_t inData, [AAX_CString](#) &outString)
- static void [ToString_AppendCStr](#) (const char *inLabel, const char *inCStr, [AAX_CString](#) &outString)
- static void [ToString_AppendByteRange](#) (const char *inLabel, const uint8_t *inData, int32_t inNumBytes, [AAX_CString](#) &outString)
- static void [ToString_AppendValid](#) (bool inCheck, [AAX_CString](#) &outString)

14.97.1 Constructor & Destructor Documentation

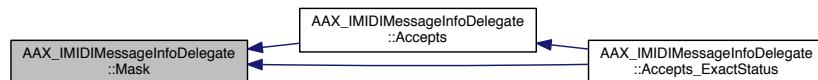
14.97.1.1 virtual AAX_IMIDIMessageInfoDelegate::~AAX_IMIDIMessageInfoDelegate() [inline], [virtual]

14.97.2 Member Function Documentation

14.97.2.1 virtual uint32_t AAX_IMIDIMessageInfoDelegate::Mask() const [pure virtual]

Referenced by [Accepts\(\)](#), and [Accepts_ExactStatus\(\)](#).

Here is the caller graph for this function:



14.97.2.2 virtual uint32_t AAX_IMIDIMessageInfoDelegate::Length() const [pure virtual]

Referenced by [Accepts\(\)](#).

Here is the caller graph for this function:



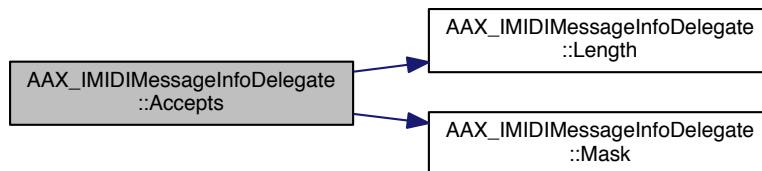
14.97.2.3 virtual [AAX_CString](#) AAX_IMIDIMessageInfoDelegate::ToString(uint32_t inLength, const uint8_t * inData) const [pure virtual]

14.97.2.4 virtual bool AAX_IMIDIMessageInfoDelegate::Accepts (uint32_t *inLength*, const uint8_t * *inData*) const
 [inline], [virtual]

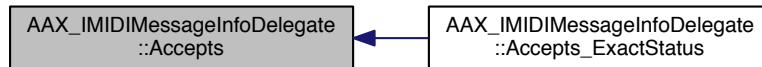
References Length(), and Mask().

Referenced by Accepts_ExactStatus().

Here is the call graph for this function:



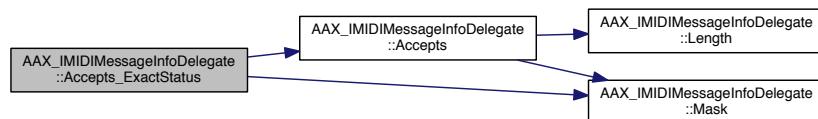
Here is the caller graph for this function:



14.97.2.5 bool AAX_IMIDIMessageInfoDelegate::Accepts_ExactStatus (uint32_t *inLength*, const uint8_t * *inData*) const
 [inline], [protected]

References Accepts(), and Mask().

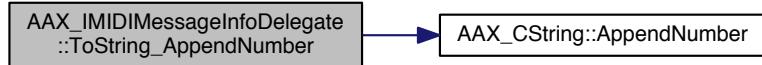
Here is the call graph for this function:



14.97.2.6 static void AAX_IMIDIMessageInfoDelegate::ToString_AppendNumber (const char * *inLabel*, int32_t *inData*,
 AAX_CString & *outString*) [inline], [static], [protected]

References AAX_CString::AppendNumber().

Here is the call graph for this function:



14.97.2.7 static void AAX_IMIDIMessageInfoDelegate::ToString_AppendCStr (const char * *inLabel*, const char * *inCStr*, AAX_CString & *outString*) [inline], [static], [protected]

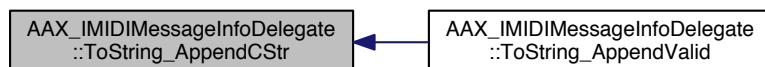
References AAX_CString::Append().

Referenced by ToString_AppendValid().

Here is the call graph for this function:



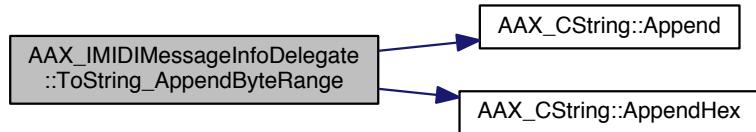
Here is the caller graph for this function:



14.97.2.8 static void AAX_IMIDIMessageInfoDelegate::ToString_AppendByteRange (const char * *inLabel*, const uint8_t * *inData*, int32_t *inNumBytes*, AAX_CString & *outString*) [inline], [static], [protected]

References AAX_CString::Append(), and AAX_CString::AppendHex().

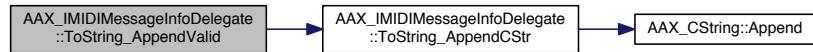
Here is the call graph for this function:



14.97.2.9 static void AAX_IMIDIMessageInfoDelegate::ToString_AppendValid (bool *inCheck*, AAX_CString & *outString*) [inline], [static], [protected]

References ToString_AppendCStr().

Here is the call graph for this function:



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

- [AAX_MIDILogging.cpp](#)

14.98 AAX_IMIDINode Class Reference

```
#include <AAX_IMIDINode.h>
```

14.98.1 Description

Interface for accessing information in a MIDI node.

:Implemented by the AAX Host

See also

[AAX_IComponentDescriptor::AddMIDINode](#)

Public Member Functions

- virtual [~AAX_IMIDINode \(\)](#)
- virtual [AAX_CMidiStream * GetNodeBuffer \(\)=0](#)
Returns a MIDI stream data structure.
- virtual [AAX_Result PostMIDIpacket \(AAX_CMidiPacket *packet\)=0](#)

Posts an [AAX_CMidiPacket](#) to an output MIDI node.

- virtual [AAX_ITransport](#) * GetTransport ()=0

Returns a transport object.

14.98.2 Constructor & Destructor Documentation

14.98.2.1 virtual [AAX_IMIDINode](#)::~[AAX_IMIDINode](#) () [inline], [virtual]

14.98.3 Member Function Documentation

14.98.3.1 virtual [AAX_CMidiStream](#)* [AAX_IMIDINode](#)::GetNodeBuffer () [pure virtual]

Returns a MIDI stream data structure.

14.98.3.2 virtual [AAX_Result](#) [AAX_IMIDINode](#)::PostMIDIpacket ([AAX_CMidiPacket](#) * *packet*) [pure virtual]

Posts an [AAX_CMidiPacket](#) to an output MIDI node.

Host Compatibility Notes Pro Tools supports the following MIDI events from plug-ins:

- NoteOn
- NoteOff
- Pitch bend
- Polyphonic key pressure
- Bank select (controller #0)
- Program change (no bank)
- Channel pressure

Parameters

in	<i>packet</i>	The MIDI packet to be pushed to a MIDI output node
----	---------------	--

14.98.3.3 virtual [AAX_ITransport](#)* [AAX_IMIDINode](#)::GetTransport () [pure virtual]

Returns a transport object.

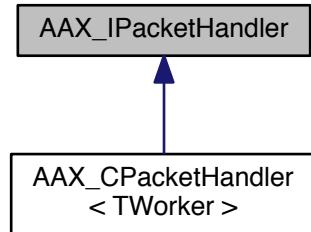
The documentation for this class was generated from the following file:

- [AAX_IMIDINode.h](#)

14.99 AAX_IPacketHandler Struct Reference

```
#include <AAX_CPacketDispatcher.h>
```

Inheritance diagram for AAX_IPacketHandler:



14.99.1 Description

Callback container used by [AAX_CPacketDispatcher](#).

Public Member Functions

- virtual [~AAX_IPacketHandler\(\)](#)
- virtual [AAX_IPacketHandler * Clone\(\)](#) const =0
- virtual [AAAX_Result Call\(AAX_CParamID inParamID, AAX_CPacket &ioPacket\)](#) const =0

14.99.2 Constructor & Destructor Documentation

14.99.2.1 virtual [AAX_IPacketHandler::~AAX_IPacketHandler\(\)](#) [inline], [virtual]

14.99.3 Member Function Documentation

14.99.3.1 virtual [AAX_IPacketHandler* AAX_IPacketHandler::Clone\(\)](#) const [pure virtual]

Implemented in [AAX_CPacketHandler< TWorker >](#).

14.99.3.2 virtual [AAAX_Result AAX_IPacketHandler::Call\(AAX_CParamID inParamID, AAX_CPacket & ioPacket\)](#) const [pure virtual]

Implemented in [AAX_CPacketHandler< TWorker >](#).

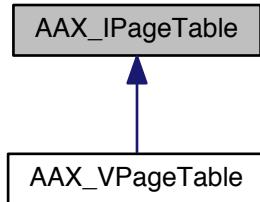
The documentation for this struct was generated from the following file:

- [AAX_CPacketDispatcher.h](#)

14.100 AAX_IPageTable Class Reference

```
#include <AAX_IPageTable.h>
```

Inheritance diagram for AAX_IPageTable:



14.100.1 Description

Interface to the host's representation of a plug-in instance's page table.

See also

[AAX_IEffectParameters::UpdatePageTable\(\)](#)

Public Member Functions

- virtual [`~AAX_IPageTable \(\)`](#)
Virtual destructor.
- virtual [`AAX_Result Clear \(\)=0`](#)
Clears all parameter mappings from the table.
- virtual [`AAX_Result Empty \(AAX_CBoolean &oEmpty\) const =0`](#)
Indicates whether the table is empty.
- virtual [`AAX_Result GetNumPages \(int32_t &oNumPages\) const =0`](#)
Get the number of pages currently in this table.
- virtual [`AAX_Result InsertPage \(int32_t iPage\)=0`](#)
Insert a new empty page before the page at index `iPage`.
- virtual [`AAX_Result RemovePage \(int32_t iPage\)=0`](#)
Remove the page at index `iPage`.
- virtual [`AAX_Result GetNumMappedParameterIDs \(int32_t iPage, int32_t &oNumParameterIdentifiers\) const =0`](#)
Returns the total number of parameter IDs which are mapped to a page.
- virtual [`AAX_Result ClearMappedParameter \(int32_t iPage, int32_t iIndex\)=0`](#)
Clear the parameter at a particular index in this table.
- virtual [`AAX_Result GetMappedParameterID \(int32_t iPage, int32_t iIndex, AAX_IString &oParameterIdentifier\) const =0`](#)
Get the parameter identifier which is currently mapped to an index in this table.
- virtual [`AAX_Result MapParameterID \(AAX_CPageTableParamID iParameterIdentifier, int32_t iPage, int32_t iIndex\)=0`](#)
Map a parameter to this table.
- virtual [`AAX_Result GetNumParametersWithNameVariations \(int32_t &oNumParameterIdentifiers\) const =0`](#)
- virtual [`AAX_Result GetNameVariationParameterIDAtIndex \(int32_t iIndex, AAX_IString &oParameterIdentifier\) const =0`](#)

- virtual `AAX_Result GetNumNameVariationsForParameter (AAX_CPageTableParamID iParameterIdentifier, int32_t &oNumVariations) const =0`
- virtual `AAX_Result GetParameterNameVariationAtIndex (AAX_CPageTableParamID iParameterIdentifier, int32_t iIndex, AAX_IString &oNameVariation, int32_t &oLength) const =0`
- virtual `AAX_Result GetParameterNameVariationOfLength (AAX_CPageTableParamID iParameterIdentifier, int32_t iLength, AAX_IString &oNameVariation) const =0`
- virtual `AAX_Result ClearParameterNameVariations ()=0`
- virtual `AAX_Result ClearNameVariationsForParameter (AAX_CPageTableParamID iParameterIdentifier)=0`
- virtual `AAX_Result SetParameterNameVariation (AAX_CPageTableParamID iParameterIdentifier, const AAX_IString &iNameVariation, int32_t iLength)=0`

14.100.2 Constructor & Destructor Documentation

14.100.2.1 virtual `AAX_IPageTable::~AAX_IPageTable () [inline], [virtual]`

Virtual destructor.

Note

This destructor MUST be virtual to prevent memory leaks.

14.100.3 Member Function Documentation

14.100.3.1 virtual `AAX_Result AAX_IPageTable::Clear () [pure virtual]`

Clears all parameter mappings from the table.

This method does not clear any parameter name variations from the table. For that, use `AAX_IPageTable::ClearParameterNameVariations()` or `AAX_IPageTable::ClearNameVariationsForParameter()`

Implemented in `AAX_VPageTable`.

14.100.3.2 virtual `AAX_Result AAX_IPageTable::Empty (AAX_CBoolean & oEmpty) const [pure virtual]`

Indicates whether the table is empty.

A table is empty if it contains no pages. A table which contains pages but no parameter assignments is not empty. A table which has associated parameter name variations but no pages is empty.

Parameters

<code>out</code>	<code>oEmpty</code>	true if this table is empty
------------------	---------------------	-----------------------------

Implemented in `AAX_VPageTable`.

14.100.3.3 virtual `AAX_Result AAX_IPageTable::GetNumPages (int32_t & oNumPages) const [pure virtual]`

Get the number of pages currently in this table.

Parameters

<code>out</code>	<code>oNumPages</code>	The number of pages which are present in the page table. Some pages might not contain any parameter assignments.
------------------	------------------------	--

Implemented in [AAX_VPageTable](#).

14.100.3.4 virtual AAX_Result AAX_IPageTable::InsertPage (int32_t iPage) [pure virtual]

Insert a new empty page before the page at index *iPage*.

Returns

[AAX_ERROR_INVALID_ARGUMENT](#) if *iPage* is greater than the total number of pages

Parameters

in	<i>iPage</i>	The insertion point page index
----	--------------	--------------------------------

Implemented in [AAX_VPageTable](#).

14.100.3.5 virtual AAX_Result AAX_IPageTable::RemovePage (int32_t iPage) [pure virtual]

Remove the page at index *iPage*.

Returns

[AAX_ERROR_INVALID_ARGUMENT](#) if *iPage* is greater than the index of the last existing page

Parameters

in	<i>iPage</i>	The target page index
----	--------------	-----------------------

Implemented in [AAX_VPageTable](#).

14.100.3.6 virtual AAX_Result AAX_IPageTable::GetNumMappedParameterIDs (int32_t iPage, int32_t & oNumParameterIdentifiers) const [pure virtual]

Returns the total number of parameter IDs which are mapped to a page.

Note

The number of mapped parameter IDs does not correspond to the actual slot indices of the parameter assignments. For example, a page could have three total parameter assignments with parameters mapped to slots 2, 4, and 6.

Returns

[AAX_ERROR_INVALID_ARGUMENT](#) if *iPage* is greater than the index of the last existing page

Parameters

in	<i>iPage</i>	The target page index
out	<i>oNumParameterIdentifiers</i>	The number of parameter identifiers which are mapped to the target page

Implemented in [AAX_VPageTable](#).

14.100.3.7 virtual AAX_Result AAX_IPageTable::ClearMappedParameter (int32_t *iPage*, int32_t *iIndex*) [pure virtual]

Clear the parameter at a particular index in this table.

Returns

[AAX_SUCCESS](#) even if no parameter was mapped at the given index (the index is still clear)

Parameters

in	<i>iPage</i>	The target page index
in	<i>iIndex</i>	The target parameter slot index within the target page

Implemented in [AAX_VPageTable](#).

14.100.3.8 virtual AAX_Result AAX_IPageTable::GetMappedParameterID (int32_t *iPage*, int32_t *iIndex*, AAX_IString & *oParameterIdentifier*) const [pure virtual]

Get the parameter identifier which is currently mapped to an index in this table.

Returns

[AAX_ERROR_INVALID_ARGUMENT](#) if no parameter is mapped at the specified page/index location

Parameters

in	<i>iPage</i>	The target page index
in	<i>iIndex</i>	The target parameter slot index within the target page
out	<i>oParameterIdentifier</i>	The identifier used for the mapped parameter in the page table (may be parameter name or ID)

Implemented in [AAX_VPageTable](#).

14.100.3.9 virtual AAX_Result AAX_IPageTable::MapParameterID (AAX_CPageTableParamID *iParameterIdentifier*, int32_t *iPage*, int32_t *iIndex*) [pure virtual]

Map a parameter to this table.

If *iParameterIdentifier* is an empty string then the parameter assignment will be cleared

Returns

[AAX_ERROR_NULL_ARGUMENT](#) if *iParameterIdentifier* is null
[AAX_ERROR_INVALID_ARGUMENT](#) if *iPage* is greater than the index of the last existing page
[AAX_ERROR_INVALID_ARGUMENT](#) if *iIndex* is negative

Parameters

in	<i>iParameterIdentifier</i>	The identifier for the parameter which will be mapped
----	-----------------------------	---

in	<i>iPage</i>	The target page index
in	<i>iIndex</i>	The target parameter slot index within the target page

Implemented in [AAX_VPageTable](#).

14.100.3.10 virtual AAX_Result AAX_IPageTable::GetNumParametersWithNameVariations (int32_t & *oNumParameterIdentifiers*) const [pure virtual]

Get the number of parameters with name variations defined for the current table type

Provides the number of parameters with *lt;ControlNameVariations* which are explicitly defined for the current page table type.

Note

Normally parameter name variations are only used with the 'PgTL' table type

See also

- [AAX_IPageTable::GetNameVariationParameterIDAtIndex\(\)](#)

Parameters

out	<i>oNumParameterIdentifiers</i>	The number of parameters with name variations explicitly associated with the current table type.
-----	---------------------------------	--

Implemented in [AAX_VPageTable](#).

14.100.3.11 virtual AAX_Result AAX_IPageTable::GetNameVariationParameterIDAtIndex (int32_t *iIndex*, AAX_IString & *oParameterIdentifier*) const [pure virtual]

Get the identifier for a parameter with name variations defined for the current table type

Note

Normally parameter name variations are only used with the 'PgTL' table type

See also

- [AAX_IPageTable::GetNumParametersWithNameVariations\(\)](#)

Parameters

in	<i>iIndex</i>	The target parameter index within the list of parameters with explicit name variations defined for this table type.
out	<i>oParameterIdentifier</i>	The identifier used for the parameter in the page table name variations list (may be parameter name or ID)

Implemented in [AAX_VPageTable](#).

14.100.3.12 virtual AAX_Result AAX_IPageTable::GetNumNameVariationsForParameter (AAX_CPageTableParamID *iParameterIdentifier*, int32_t & *oNumVariations*) const [pure virtual]

Get the number of name variations defined for a parameter

Provides the number of *lt;ControlNameVariations* which are explicitly defined for *iParameterIdentifier* for the current page table type. No fallback logic is used to resolve this to the list of variations which would actually be used for an attached control surface if no explicit variations are defined for the current table type.

Note

Normally parameter name variations are only used with the 'PgTL' table type

See also

- [AAX_IPageTable::GetParameterNameVariationAtIndex\(\)](#)

Returns

AAX_SUCCESS and provides zero to *oNumVariations* if *iParameterIdentifier* is not found

Parameters

in	<i>iParameterIdentifier</i>	The identifier for the parameter
out	<i>oNumVariations</i>	The number of name variations which are defined for this parameter and explicitly associated with the current table type.

Implemented in [AAX_VPageTable](#).

```
14.100.3.13 virtual AAX_Result AAX_IPageTable::GetParameterNameVariationAtIndex ( AAX_CPageTableParamID
    iParameterIdentifier, int32_t iIndex, AAX_IString & oNameVariation, int32_t & oLength ) const [pure
    virtual]
```

Get a parameter name variation from the page table

Only returns *lt;ControlNameVariations*; which are explicitly defined for the current page table type. No fallback logic is used to resolve this to the abbreviation which would actually be shown on an attached control surface if no explicit variation is defined for the current table type.

Note

Normally parameter name variations are only used with the 'PgTL' table type

See also

- [AAX_IPageTable::GetNumNameVariationsForParameter\(\)](#)

See also

- [AAX_IPageTable::GetParameterNameVariationOfLength\(\)](#)

Returns

AAX_ERROR_NO_ABBREVIATED_PARAMETER_NAME if no suitable variation is defined for this table
AAX_ERROR_ARGUMENT_OUT_OF_RANGE if *iIndex* is out of range

Parameters

in	<i>iParameterIdentifier</i>	The identifier for the parameter
in	<i>iIndex</i>	Index of the name variation
out	<i>oNameVariation</i>	The name variation, if one is explicitly defined for this table type
out	<i>oLength</i>	The length value for this name variation. This corresponds to the variation's sz attribute in the page table XML and may be different from the string length of <i>iNameVariation</i> .

Implemented in [AAX_VPageTable](#).

14.100.3.14 virtual AAX_Result AAX_IPageTable::GetParameterNameVariationOfLength (AAX_CPageTableParamID iParameterIdentifier, int32_t iLength, AAX_IString & oNameVariation) const [pure virtual]

Get a parameter name variation of a particular length from the page table

Only returns `lt;ControlNameVariations[lt;` which are explicitly defined of `iLength` for the current page table type. No fallback logic is used to resolve this to the abbreviation which would actually be shown on an attached control surface if no explicit variation is defined for the specified length or current table type.

Note

Normally parameter name variations are only used with the 'PgTL' table type

See also

- [AAX_IPageTable::GetParameterNameVariationAtIndex\(\)](#)

Returns

[AAX_ERROR_NO_ABBREVIATED_PARAMETER_NAME](#) if no suitable variation is defined for this table

Parameters

in	<i>iParameterIdentifier</i>	The identifier for the parameter
in	<i>iLength</i>	The variation length to check, i.e. the <code>sz</code> attribute for the name variation in the page table XML
out	<i>oNameVariation</i>	The name variation, if one is explicitly defined for this table type and <code>iLength</code>

Implemented in [AAX_VPageTable](#).

14.100.3.15 virtual AAX_Result AAX_IPageTable::ClearParameterNameVariations () [pure virtual]

Clears all name variations for the current page table type

Note

Normally parameter name variations are only used with the 'PgTL' table type

See also

- [AAX_IPageTable::Clear\(\)](#)
- [AAX_IPageTable::ClearNameVariationsForParameter\(\)](#)

Implemented in [AAX_VPageTable](#).

14.100.3.16 virtual AAX_Result AAX_IPageTable::ClearNameVariationsForParameter (AAX_CPageTableParamID iParameterIdentifier) [pure virtual]

Clears all name variations for a single parameter for the current page table type

Warning

This will invalidate the list of parameter name variations indices, i.e. the parameter identifier associated with each index by [AAX_IPageTable::GetNameVariationParameterIDAtIndex\(\)](#)

Note

Normally parameter name variations are only used with the 'PgTL' table type

See also

[AAX_IPageTable::Clear\(\)](#)
[AAX_IPageTable::ClearParameterNameVariations\(\)](#)

Returns

`AAX_SUCCESS` and provides zero to `oNumVariations` if `iParameterIdentifier` is not found

Parameters

in	<i>iParameterIdentifier</i>	The identifier for the parameter
----	-----------------------------	----------------------------------

Implemented in [AAX_VPageTable](#).

14.100.3.17 virtual AAX_Result AAX_IPageTable::SetParameterNameVariation (AAX_CPageTableParamID *iParameterIdentifier*, const AAX_IString & *iNameVariation*, int32_t *iLength*) [pure virtual]

Sets a name variation explicitly for the current page table type

This will add a new name variation or overwrite the existing name variation with the same length which is defined for the current table type.

Warning

If no name variation previously existed for this parameter then this will invalidate the list of parameter name variations indices, i.e. the parameter identifier associated with each index by [AAX_IPageTable::GetNameVariationParameterIDAtIndex\(\)](#)

Note

Normally parameter name variations are only used with the 'PgTL' table type

Returns

`AAX_ERROR_INVALID_ARGUMENT` if `iNameVariation` is empty or if `iLength` is less than zero

Parameters

in	<i>iParameterIdentifier</i>	The identifier for the parameter
in	<i>iNameVariation</i>	The new parameter name variation
in	<i>iLength</i>	The length value for this name variation. This corresponds to the variation's <code>sz</code> attribute in the page table XML and is not required to match the length of <code>iNameVariation</code> .

Implemented in [AAX_VPageTable](#).

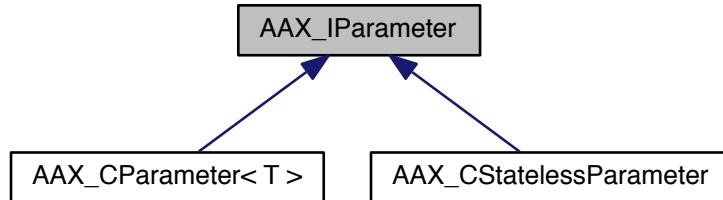
The documentation for this class was generated from the following file:

- [AAX_IPageTable.h](#)

14.101 AAX_IParameter Class Reference

```
#include <AAX_IParameter.h>
```

Inheritance diagram for AAX_IParameter:



14.101.1 Description

The base interface for all normalizable plug-in parameters.

:Internal to the AAX SDK

This class is an outside interface for an arbitrarily typed parameter. The subclasses of this generic interface hold the parameter's state and conversion functionality.

Note

This class is *not* part of the AAX ABI and must not be passed between the plug-in and the host. Version checking is recommended when passing references to this interface between plug-in modules (e.g. between the data model and the GUI)

Public Member Functions

- virtual [~AAX_IParameter \(\)](#)
Virtual destructor.
- virtual [AAX_IParameterValue * CloneValue \(\) const =0](#)
Clone the parameter's value to a new AAX_IParameterValue object.
- virtual [void SetType \(AAX_EParameterType iControlType\)=0](#)
Sets the type of this parameter.
- virtual [AAX_EParameterType GetType \(\) const =0](#)
Returns the type of this parameter as an AAX_EParameterType.
- virtual [void SetOrientation \(AAX_EParameterOrientation iOrientation\)=0](#)
Sets the orientation of this parameter.
- virtual [AAX_EParameterOrientation GetOrientation \(\) const =0](#)
Returns the orientation of this parameter.
- virtual [void SetTaperDelegate \(AAX_ITaperDelegateBase &inTaperDelegate, bool inPreserveValue\)=0](#)
Sets the parameter's taper delegate.
- virtual [void SetDisplayDelegate \(AAX_IDisplayDelegateBase &inDisplayDelegate\)=0](#)
Sets the parameter's display delegate.

Identification methods

- virtual `AAX_CParamID Identifier () const =0`
Returns the parameter's unique identifier.
- virtual void `SetName (const AAX_CString &name)=0`
Sets the parameter's display name.
- virtual const `AAX_CString & Name () const =0`
Returns the parameter's display name.
- virtual void `AddShortenedName (const AAX_CString &name)=0`
Sets the parameter's shortened display name.
- virtual const `AAX_CString & ShortenedName (int32_t iNumCharacters) const =0`
Returns the parameter's shortened display name.
- virtual void `ClearShortenedNames ()=0`
Clears the internal list of shortened display names.

Automation methods

- virtual bool `Automatable () const =0`
Returns true if the parameter is automatable, false if it is not.
- virtual void `SetAutomationDelegate (AAX_IAutomationDelegate *iAutomationDelegate)=0`
Sets the automation delegate (if one is required)
- virtual void `Touch ()=0`
Signals the automation system that a control has been touched.
- virtual void `Release ()=0`
Signals the automation system that a control has been released.

Taper methods

- virtual void `SetNormalizedValue (double newNormalizedValue)=0`
Sets a parameter value using its normalized representation.
- virtual double `GetNormalizedValue () const =0`
Returns the normalized representation of the parameter's current real value.
- virtual void `SetNormalizedDefaultValue (double normalizedDefault)=0`
Sets the parameter's default value using its normalized representation.
- virtual double `GetNormalizedDefaultValue () const =0`
Returns the normalized representation of the parameter's real default value.
- virtual void `SetToDefaultValue ()=0`
Restores the state of this parameter to its default value.
- virtual void `SetNumberOfSteps (uint32_t numSteps)=0`
Sets the number of discrete steps for this parameter.
- virtual uint32_t `GetNumberOfSteps () const =0`
Returns the number of discrete steps used by the parameter.
- virtual uint32_t `GetStepValue () const =0`
Returns the current step for the current value of the parameter.
- virtual double `GetNormalizedValueFromStep (uint32_t iStep) const =0`
Returns the normalized value for a given step.
- virtual uint32_t `GetStepValueFromNormalizedValue (double normalizedValue) const =0`
Returns the step value for a normalized value of the parameter.
- virtual void `SetStepValue (uint32_t iStep)=0`
Returns the current step for the current value of the parameter.

Display methods

This functionality is most often used by GUIs, but can also be useful for state serialization.

- virtual bool `GetValueString (AAX_CString *valueString) const =0`
Serializes the parameter value into a string.
- virtual bool `GetValueString (int32_t iMaxNumChars, AAX_CString *valueString) const =0`
Serializes the parameter value into a string, size hint included.
- virtual bool `GetNormalizedValueFromBool (bool value, double *normalizedValue) const =0`

- Converts a bool to a normalized parameter value.
- virtual bool `GetNormalizedValueFromInt32` (int32_t value, double *normalizedValue) const =0
Converts an integer to a normalized parameter value.
- virtual bool `GetNormalizedValueFromFloat` (float value, double *normalizedValue) const =0
Converts a float to a normalized parameter value.
- virtual bool `GetNormalizedValueFromDouble` (double value, double *normalizedValue) const =0
Converts a double to a normalized parameter value.
- virtual bool `GetNormalizedValueFromString` (const AAX_CString &valueString, double *normalizedValue) const =0
Converts a given string to a normalized parameter value.
- virtual bool `GetBoolFromNormalizedValue` (double normalizedValue, bool *value) const =0
Converts a normalized parameter value to a bool representing the corresponding real value.
- virtual bool `GetInt32FromNormalizedValue` (double normalizedValue, int32_t *value) const =0
Converts a normalized parameter value to an integer representing the corresponding real value.
- virtual bool `GetFloatFromNormalizedValue` (double normalizedValue, float *value) const =0
Converts a normalized parameter value to a float representing the corresponding real value.
- virtual bool `GetDoubleFromNormalizedValue` (double normalizedValue, double *value) const =0
Converts a normalized parameter value to a double representing the corresponding real value.
- virtual bool `GetStringFromNormalizedValue` (double normalizedValue, AAX_CString &valueString) const =0
Converts a normalized parameter value to a string representing the corresponding real value.
- virtual bool `GetStringFromNormalizedValue` (double normalizedValue, int32_t iMaxNumChars, AAX_CString &valueString) const =0
Converts a normalized parameter value to a string representing the corresponding real, size hint included. value.
- virtual bool `SetValueFromString` (const AAX_CString &newValueString)=0
Converts a string to a real parameter value and sets the parameter to this value.

Typed accessors

- virtual bool `GetValueAsBool` (bool *value) const =0
Retrieves the parameter's value as a bool.
- virtual bool `GetValueAsInt32` (int32_t *value) const =0
Retrieves the parameter's value as an int32_t.
- virtual bool `GetValueAsFloat` (float *value) const =0
Retrieves the parameter's value as a float.
- virtual bool `GetValueAsDouble` (double *value) const =0
Retrieves the parameter's value as a double.
- virtual bool `GetValueAsString` (AAX_IString *value) const =0
Retrieves the parameter's value as a string.
- virtual bool `SetValueWithBool` (bool value)=0
Sets the parameter's value as a bool.
- virtual bool `SetValueWithInt32` (int32_t value)=0
Sets the parameter's value as an int32_t.
- virtual bool `SetValueWithFloat` (float value)=0
Sets the parameter's value as a float.
- virtual bool `SetValueWithDouble` (double value)=0
Sets the parameter's value as a double.
- virtual bool `SetValueWithString` (const AAX_IString &value)=0
Sets the parameter's value as a string.

Host interface methods

- virtual void `UpdateNormalizedValue` (double newNormalizedValue)=0
Sets the parameter's state given a normalized value.

14.101.2 Constructor & Destructor Documentation

14.101.2.1 virtual [AAX_IParameter](#)::~[AAX_IParameter](#)() [inline], [virtual]

Virtual destructor.

Note

This destructor MUST be virtual to prevent memory leaks.

14.101.3 Member Function Documentation

14.101.3.1 virtual [AAX_IPAttributeValue](#)* [AAX_IParameter](#)::[CloneValue](#)() const [pure virtual]

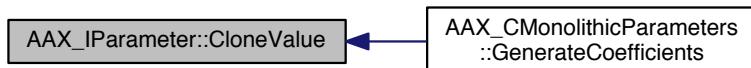
Clone the parameter's value to a new [AAX_IPAttributeValue](#) object.

The returned object is independent from the [AAX_IParameter](#). For example, changing the state of the returned object will not result in a change to the original [AAX_IParameter](#).

Implemented in [AAX_CStatelessParameter](#), and [AAX_CParameter< T >](#).

Referenced by [AAX_CMonolithicParameters::GenerateCoefficients\(\)](#).

Here is the caller graph for this function:



14.101.3.2 virtual [AAX_CParamID](#) [AAX_IParameter](#)::[Identifier](#)() const [pure virtual]

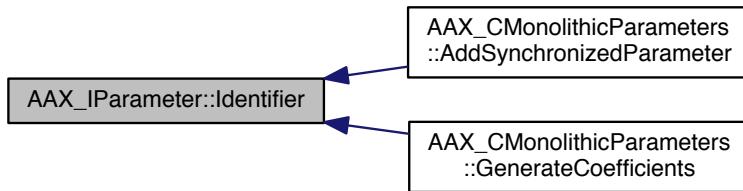
Returns the parameter's unique identifier.

This unique ID is used by the [Parameter Manager](#) and by outside applications to uniquely identify and target control messages. This value may not be changed after the parameter has been constructed.

Implemented in [AAX_CStatelessParameter](#), and [AAX_CParameter< T >](#).

Referenced by [AAX_CMonolithicParameters::AddSynchronizedParameter\(\)](#), and [AAX_CMonolithicParameters::GenerateCoefficients\(\)](#).

Here is the caller graph for this function:



14.101.3.3 virtual void AAX_IParameter::SetName (const AAX_CString & name) [pure virtual]

Sets the parameter's display name.

This name is used for display only, it is not used for indexing or identifying the parameter. This name may be changed after the parameter has been created, but display name changes may not be recognized by all AAX hosts.

Parameters

in	name	Display name that will be assigned to the parameter
----	------	---

Implemented in [AAX_CStatelessParameter](#), and [AAX_CParameter< T >](#).

14.101.3.4 virtual const AAX_CString& AAX_IParameter::Name () const [pure virtual]

Returns the parameter's display name.

Note

This method returns a const reference in order to prevent a string copy. Do not cast away the const to change this value.

Implemented in [AAX_CStatelessParameter](#), and [AAX_CParameter< T >](#).

14.101.3.5 virtual void AAX_IParameter::AddShortenedName (const AAX_CString & name) [pure virtual]

Sets the parameter's shortened display name.

This name is used for display only, it is not used for indexing or identifying the parameter. These names show up when the host asks for shorter length parameter names for display on Control Surfaces or other string length constrained situations.

Parameters

in	name	Shortened display names that will be assigned to the parameter
----	------	--

Implemented in [AAX_CStatelessParameter](#), and [AAX_CParameter< T >](#).

14.101.3.6 virtual const AAX_CString& AAX_IParameter::ShortenedName (int32_t iNumCharacters) const [pure virtual]

Returns the parameter's shortened display name.

Note

This method returns a const reference in order to prevent a string copy. Do not cast away the const to change this value.

Implemented in [AAX_CStatelessParameter](#), and [AAX_CParameter< T >](#).

14.101.3.7 virtual void AAX_IParameter::ClearShortenedNames() [pure virtual]

Clears the internal list of shortened display names.

Implemented in [AAX_CStatelessParameter](#), and [AAX_CParameter< T >](#).

14.101.3.8 virtual bool AAX_IParameter::Automatable() const [pure virtual]

Returns true if the parameter is automatable, false if it is not.

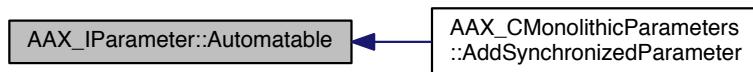
Note

Subclasses that return true in this method must support host-based automation.

Implemented in [AAX_CStatelessParameter](#), and [AAX_CParameter< T >](#).

Referenced by [AAX_CMonolithicParameters::AddSynchronizedParameter\(\)](#).

Here is the caller graph for this function:

**14.101.3.9 virtual void AAX_IParameter::SetAutomationDelegate(AAX_IAutomationDelegate * iAutomationDelegate) [pure virtual]**

Sets the automation delegate (if one is required)

Parameters

in	<i>iAutomationDelegate</i>	A reference to the parameter manager's automation delegate interface
----	----------------------------	--

Implemented in [AAX_CStatelessParameter](#), and [AAX_CParameter< T >](#).

14.101.3.10 virtual void AAX_IParameter::Touch() [pure virtual]

Signals the automation system that a control has been touched.

Call this method in response to GUI events that begin editing, such as a mouse down. After this method has been called you are free to call [SetNormalizedValue\(\)](#) as much as you need, e.g. in order to respond to subsequent mouse moved events. Call [Release\(\)](#) to free the parameter for updates from other controls.

Implemented in [AAX_CStatelessParameter](#), and [AAX_CParameter< T >](#).

14.101.3.11 virtual void AAX_IParameter::Release() [pure virtual]

Signals the automation system that a control has been released.

Call this method in response to GUI events that complete editing, such as a mouse up. Once this method has been called you should not call [SetNormalizedValue\(\)](#) again until after the next call to [Touch\(\)](#).

Implemented in [AAX_CStatelessParameter](#), and [AAX_CParameter< T >](#).

14.101.3.12 virtual void AAX_IParameter::SetNormalizedValue (double *newNormalizedValue*) [pure virtual]

Sets a parameter value using it's normalized representation.

For more information regarding normalized values, see [Parameter Manager](#)

Parameters

in	<i>newNormalizedValue</i>	New value (normalized) to which the parameter will be set
----	---------------------------	---

Implemented in [AAX_CStatelessParameter](#), and [AAX_CParameter< T >](#).

14.101.3.13 virtual double AAX_IParameter::GetNormalizedValue() const [pure virtual]

Returns the normalized representation of the parameter's current real value.

Implemented in [AAX_CStatelessParameter](#), and [AAX_CParameter< T >](#).

14.101.3.14 virtual void AAX_IParameter::SetNormalizedDefaultValue (double *normalizedDefault*) [pure virtual]

Sets the parameter's default value using its normalized representation.

Implemented in [AAX_CStatelessParameter](#), and [AAX_CParameter< T >](#).

14.101.3.15 virtual double AAX_IParameter::GetNormalizedDefaultValue() const [pure virtual]

Returns the normalized representation of the parameter's real default value.

Implemented in [AAX_CStatelessParameter](#), and [AAX_CParameter< T >](#).

14.101.3.16 virtual void AAX_IParameter::SetToDefaultValue() [pure virtual]

Restores the state of this parameter to its default value.

Implemented in [AAX_CStatelessParameter](#), and [AAX_CParameter< T >](#).

14.101.3.17 virtual void AAX_IParameter::SetNumberOfSteps (uint32_t *numSteps*) [pure virtual]

Sets the number of discrete steps for this parameter.

Stepped parameter values are useful for discrete parameters and for "jumping" events such as mouse wheels, page up/down, etc. The parameter's step size is used to specify the coarseness of those changes.

Note

numSteps MUST be greater than zero. All other values may be considered an error by the host.

Parameters

in	<i>numSteps</i>	The number of steps that the parameter will use
----	-----------------	---

Implemented in [AAX_CStatelessParameter](#), and [AAX_CParameter< T >](#).

14.101.3.18 virtual uint32_t AAX_IParameter::GetNumberOfSteps() const [pure virtual]

Returns the number of discrete steps used by the parameter.

See [SetNumberOfSteps\(\)](#) for more information about parameter steps.

Implemented in [AAX_CStatelessParameter](#), and [AAX_CParameter< T >](#).

14.101.3.19 virtual uint32_t AAX_IParameter::GetStepValue() const [pure virtual]

Returns the current step for the current value of the parameter.

See [SetNumberOfSteps\(\)](#) for more information about parameter steps.

Implemented in [AAX_CStatelessParameter](#), and [AAX_CParameter< T >](#).

14.101.3.20 virtual double AAX_IParameter::GetNormalizedValueFromStep(uint32_t iStep) const [pure virtual]

Returns the normalized value for a given step.

See [SetNumberOfSteps\(\)](#) for more information about parameter steps.

Implemented in [AAX_CStatelessParameter](#), and [AAX_CParameter< T >](#).

14.101.3.21 virtual uint32_t AAX_IParameter::GetStepValueFromNormalizedValue(double normalizedValue) const [pure virtual]

Returns the step value for a normalized value of the parameter.

See [SetNumberOfSteps\(\)](#) for more information about parameter steps.

Implemented in [AAX_CStatelessParameter](#), and [AAX_CParameter< T >](#).

14.101.3.22 virtual void AAX_IParameter::SetStepValue(uint32_t iStep) [pure virtual]

Returns the current step for the current value of the parameter.

See [SetNumberOfSteps\(\)](#) for more information about parameter steps.

Implemented in [AAX_CStatelessParameter](#), and [AAX_CParameter< T >](#).

14.101.3.23 virtual bool AAX_IParameter::GetValueString(AAX_CString * valueString) const [pure virtual]

Serializes the parameter value into a string.

Parameters

out	<i>valueString</i>	A string representing the parameter's real value
-----	--------------------	--

Return values

<i>true</i>	The string conversion was successful
<i>false</i>	The string conversion was unsuccessful

Implemented in [AAX_CStatelessParameter](#), and [AAX_CParameter< T >](#).

14.101.3.24 virtual bool AAX_IParameter::GetValueString (int32_t *iMaxNumChars*, AAX_CString * *valueString*) const [pure virtual]

Serializes the parameter value into a string, size hint included.

Parameters

<i>in</i>	<i>iMaxNumChars</i>	A size hint for the size of the string being requested. Useful for control surfaces and other limited area text fields. (make sure that size of desired string also has room for null termination)
<i>out</i>	<i>valueString</i>	A string representing the parameter's real value

Return values

<i>true</i>	The string conversion was successful
<i>false</i>	The string conversion was unsuccessful

Implemented in [AAX_CStatelessParameter](#), and [AAX_CParameter< T >](#).

14.101.3.25 virtual bool AAX_IParameter::GetNormalizedValueFromBool (bool *value*, double * *normalizedValue*) const [pure virtual]

Converts a bool to a normalized parameter value.

Parameters

<i>in</i>	<i>value</i>	A value for the parameter
<i>out</i>	<i>normalizedValue</i>	The normalized parameter value associated with value

Return values

<i>true</i>	The value conversion was successful
<i>false</i>	The value conversion was unsuccessful

Implemented in [AAX_CStatelessParameter](#), [AAX_CParameter< T >](#), and [AAX_CParameter< T >](#).

14.101.3.26 virtual bool AAX_IParameter::GetNormalizedValueFromInt32 (int32_t *value*, double * *normalizedValue*) const [pure virtual]

Converts an integer to a normalized parameter value.

Parameters

<i>in</i>	<i>value</i>	A value for the parameter
<i>out</i>	<i>normalizedValue</i>	The normalized parameter value associated with value

Return values

<i>true</i>	The value conversion was successful
<i>false</i>	The value conversion was unsuccessful

Implemented in [AAX_CStatelessParameter](#), [AAX_CParameter< T >](#), and [AAX_CParameter< T >](#).

```
14.101.3.27 virtual bool AAX_IParameter::GetNormalizedValueFromFloat ( float value, double * normalizedValue ) const  
[pure virtual]
```

Converts a float to a normalized parameter value.

Parameters

in	<i>value</i>	A value for the parameter
out	<i>normalizedValue</i>	The normalized parameter value associated with value

Return values

<i>true</i>	The value conversion was successful
<i>false</i>	The value conversion was unsuccessful

Implemented in [AAX_CStatelessParameter](#), [AAX_CParameter< T >](#), and [AAX_CParameter< T >](#).

14.101.3.28 virtual bool AAX_IParameter::GetNormalizedValueFromDouble (double *value*, double * *normalizedValue*) const [pure virtual]

Converts a double to a normalized parameter value.

Parameters

in	<i>value</i>	A value for the parameter
out	<i>normalizedValue</i>	The normalized parameter value associated with value

Return values

<i>true</i>	The value conversion was successful
<i>false</i>	The value conversion was unsuccessful

Implemented in [AAX_CStatelessParameter](#), [AAX_CParameter< T >](#), and [AAX_CParameter< T >](#).

14.101.3.29 virtual bool AAX_IParameter::GetNormalizedValueFromString (const AAX_CString & *valueString*, double * *normalizedValue*) const [pure virtual]

Converts a given string to a normalized parameter value.

Parameters

in	<i>valueString</i>	A string representing a possible real value for the parameter
out	<i>normalizedValue</i>	The normalized parameter value associated with valueString

Return values

<i>true</i>	The string conversion was successful
<i>false</i>	The string conversion was unsuccessful

Implemented in [AAX_CStatelessParameter](#), and [AAX_CParameter< T >](#).

14.101.3.30 virtual bool AAX_IParameter::GetBoolFromNormalizedValue (double *normalizedValue*, bool * *value*) const [pure virtual]

Converts a normalized parameter value to a bool representing the corresponding real value.

Parameters

in	<i>normalizedValue</i>	The normalized value to convert
out	<i>value</i>	The converted value. Set only if conversion is successful.

Return values

<i>true</i>	The conversion to bool was successful
<i>false</i>	The conversion to bool was unsuccessful

Implemented in [AAX_CStatelessParameter](#), [AAX_CParameter< T >](#), and [AAX_CParameter< T >](#).

14.101.3.31 virtual bool AAX_IParameter::GetInt32FromNormalizedValue (double *normalizedValue*, int32_t * *value*) const [pure virtual]

Converts a normalized parameter value to an integer representing the corresponding real value.

Parameters

in	<i>normalizedValue</i>	The normalized value to convert
out	<i>value</i>	The converted value. Set only if conversion is successful.

Return values

<i>true</i>	The conversion to int32_t was successful
<i>false</i>	The conversion to int32_t was unsuccessful

Implemented in [AAX_CStatelessParameter](#), [AAX_CParameter< T >](#), and [AAX_CParameter< T >](#).

14.101.3.32 virtual bool AAX_IParameter::GetFloatFromNormalizedValue (double *normalizedValue*, float * *value*) const [pure virtual]

Converts a normalized parameter value to a float representing the corresponding real value.

Parameters

in	<i>normalizedValue</i>	The normalized value to convert
out	<i>value</i>	The converted value. Set only if conversion is successful.

Return values

<i>true</i>	The conversion to float was successful
<i>false</i>	The conversion to float was unsuccessful

Implemented in [AAX_CStatelessParameter](#), [AAX_CParameter< T >](#), and [AAX_CParameter< T >](#).

14.101.3.33 virtual bool AAX_IParameter::GetDoubleFromNormalizedValue (double *normalizedValue*, double * *value*) const [pure virtual]

Converts a normalized parameter value to a double representing the corresponding real value.

Parameters

in	<i>normalizedValue</i>	The normalized value to convert
out	<i>value</i>	The converted value. Set only if conversion is successful.

Return values

<i>true</i>	The conversion to double was successful
<i>false</i>	The conversion to double was unsuccessful

Implemented in [AAX_CStatelessParameter](#), [AAX_CParameter< T >](#), and [AAX_CParameter< T >](#).

14.101.3.34 virtual bool AAX_IParameter::GetStringFromNormalizedValue (double *normalizedValue*, AAX_CString & *valueString*) const [pure virtual]

Converts a normalized parameter value to a string representing the corresponding real value.

Parameters

in	<i>normalizedValue</i>	A normalized parameter value
out	<i>valueString</i>	A string representing the parameter value associated with <i>normalizedValue</i>

Return values

<i>true</i>	The string conversion was successful
<i>false</i>	The string conversion was unsuccessful

Implemented in [AAX_CStatelessParameter](#), and [AAX_CParameter< T >](#).

14.101.3.35 virtual bool AAX_IParameter::GetStringFromNormalizedValue (double *normalizedValue*, int32_t *iMaxNumChars*, AAX_CString & *valueString*) const [pure virtual]

Converts a normalized parameter value to a string representing the corresponding real, size hint included. value.

Parameters

in	<i>normalizedValue</i>	A normalized parameter value
in	<i>iMaxNumChars</i>	A size hint for the size of the string being requested. Useful for control surfaces and other limited area text fields. (make sure that size of desired string also has room for null termination)
out	<i>valueString</i>	A string representing the parameter value associated with <i>normalizedValue</i>

Return values

<i>true</i>	The string conversion was successful
<i>false</i>	The string conversion was unsuccessful

Implemented in [AAX_CStatelessParameter](#), and [AAX_CParameter< T >](#).

14.101.3.36 virtual bool AAX_IParameter::SetValueFromString (const AAX_CString & *newValueString*) [pure virtual]

Converts a string to a real parameter value and sets the parameter to this value.

Parameters

in	<i>newValueString</i>	A string representing the parameter's new real value
----	-----------------------	--

Return values

<i>true</i>	The string conversion was successful
<i>false</i>	The string conversion was unsuccessful

Implemented in [AAX_CStatelessParameter](#), and [AAX_CParameter< T >](#).

14.101.3.37 virtual bool AAX_IParameter::GetValueAsBool (bool * *value*) const [pure virtual]

Retrieves the parameter's value as a bool.

Parameters

out	<i>value</i>	The parameter's real value. Set only if conversion is successful.
-----	--------------	---

Return values

<i>true</i>	The conversion to bool was successful
<i>false</i>	The conversion to bool was unsuccessful

Implemented in [AAX_CStatelessParameter](#), and [AAX_CParameter< T >](#).

14.101.3.38 virtual bool AAX_IParameter::GetValueAsInt32(int32_t * value) const [pure virtual]

Retrieves the parameter's value as an int32_t.

Parameters

<i>out</i>	<i>value</i>	The parameter's real value. Set only if conversion is successful.
------------	--------------	---

Return values

<i>true</i>	The conversion to int32_t was successful
<i>false</i>	The conversion to int32_t was unsuccessful

Implemented in [AAX_CStatelessParameter](#), and [AAX_CParameter< T >](#).

14.101.3.39 virtual bool AAX_IParameter::GetValueAsFloat(float * value) const [pure virtual]

Retrieves the parameter's value as a float.

Parameters

<i>out</i>	<i>value</i>	The parameter's real value. Set only if conversion is successful.
------------	--------------	---

Return values

<i>true</i>	The conversion to float was successful
<i>false</i>	The conversion to float was unsuccessful

Implemented in [AAX_CStatelessParameter](#), and [AAX_CParameter< T >](#).

14.101.3.40 virtual bool AAX_IParameter::GetValueAsDouble(double * value) const [pure virtual]

Retrieves the parameter's value as a double.

Parameters

<i>out</i>	<i>value</i>	The parameter's real value. Set only if conversion is successful.
------------	--------------	---

Return values

<i>true</i>	The conversion to double was successful
<i>false</i>	The conversion to double was unsuccessful

Implemented in [AAX_CStatelessParameter](#), and [AAX_CParameter< T >](#).

14.101.3.41 virtual bool AAX_IParameter::GetValueAsString(AAX_IString * value) const [pure virtual]

Retrieves the parameter's value as a string.

Parameters

<i>out</i>	<i>value</i>	The parameter's real value. Set only if conversion is successful.
------------	--------------	---

Return values

<i>true</i>	The conversion to string was successful
<i>false</i>	The conversion to string was unsuccessful

Implemented in [AAX_CStatelessParameter](#), [AAX_CParameter< T >](#), and [AAX_CParameter< T >](#).

14.101.3.42 virtual bool AAX_IParameter::SetValueWithBool(bool value) [pure virtual]

Sets the parameter's value as a bool.

Parameters

<i>out</i>	<i>value</i>	The parameter's real value. Set only if conversion is successful.
------------	--------------	---

Return values

<i>true</i>	The conversion from bool was successful
<i>false</i>	The conversion from bool was unsuccessful

Implemented in [AAX_CStatelessParameter](#), [AAX_CParameter< T >](#), and [AAX_CParameter< T >](#).

14.101.3.43 virtual bool AAX_IParameter::SetValueWithInt32(int32_t value) [pure virtual]

Sets the parameter's value as an int32_t.

Parameters

<i>out</i>	<i>value</i>	The parameter's real value. Set only if conversion is successful.
------------	--------------	---

Return values

<i>true</i>	The conversion from int32_t was successful
<i>false</i>	The conversion from int32_t was unsuccessful

Implemented in [AAX_CStatelessParameter](#), [AAX_CParameter< T >](#), and [AAX_CParameter< T >](#).

14.101.3.44 virtual bool AAX_IParameter::SetValueWithFloat(float value) [pure virtual]

Sets the parameter's value as a float.

Parameters

<i>out</i>	<i>value</i>	The parameter's real value. Set only if conversion is successful.
------------	--------------	---

Return values

<i>true</i>	The conversion from float was successful
<i>false</i>	The conversion from float was unsuccessful

Implemented in [AAX_CStatelessParameter](#), [AAX_CParameter< T >](#), and [AAX_CParameter< T >](#).

14.101.3.45 virtual bool AAX_IParameter::SetValueWithDouble(double value) [pure virtual]

Sets the parameter's value as a double.

Parameters

<code>out</code>	<code>value</code>	The parameter's real value. Set only if conversion is successful.
------------------	--------------------	---

Return values

<code>true</code>	The conversion from double was successful
<code>false</code>	The conversion from double was unsuccessful

Implemented in [AAX_CStatelessParameter](#), [AAX_CParameter< T >](#), and [AAX_CParameter< T >](#).

14.101.3.46 virtual bool AAX_IParameter::SetValueWithString (const AAX_IString & value) [pure virtual]

Sets the parameter's value as a string.

Parameters

<code>out</code>	<code>value</code>	The parameter's real value. Set only if conversion is successful.
------------------	--------------------	---

Return values

<code>true</code>	The conversion from string was successful
<code>false</code>	The conversion from string was unsuccessful

Implemented in [AAX_CStatelessParameter](#), [AAX_CParameter< T >](#), and [AAX_CParameter< T >](#).

14.101.3.47 virtual void AAX_IParameter::SetType (AAX_EParameterType iControlType) [pure virtual]

Sets the type of this parameter.

See [GetType](#) for use cases

Parameters

<code>in</code>	<code>iControlType</code>	The parameter's new type as an AAX_EParameterType
-----------------	---------------------------	---

Implemented in [AAX_CStatelessParameter](#), and [AAX_CParameter< T >](#).

14.101.3.48 virtual AAX_EParameterType AAX_IParameter::GetType () const [pure virtual]

Returns the type of this parameter as an AAX_EParameterType.

Todo Document use cases for control type

Implemented in [AAX_CStatelessParameter](#), and [AAX_CParameter< T >](#).

14.101.3.49 virtual void AAX_IParameter::SetOrientation (AAX_EParameterOrientation iOrientation) [pure virtual]

Sets the orientation of this parameter.

Parameters

<code>in</code>	<code>iOrientation</code>	The parameter's new orientation
-----------------	---------------------------	---------------------------------

Implemented in [AAX_CStatelessParameter](#), and [AAX_CParameter< T >](#).

14.101.3.50 virtual AAX_EParameterOrientation AAX_IParameter::GetOrientation () const [pure virtual]

Returns the orientation of this parameter.

Implemented in [AAX_CStatelessParameter](#), and [AAX_CParameter< T >](#).

14.101.3.51 virtual void AAX_IParameter::SetTaperDelegate (AAX_ITaperDelegateBase & *inTaperDelegate*, bool *inPreserveValue*) [pure virtual]

Sets the parameter's taper delegate.

Parameters

in	<i>inTaperDelegate</i>	A reference to the parameter's new taper delegate
in	<i>inPreserveValue</i>	

Todo Document this parameter

Implemented in [AAX_CStatelessParameter](#), and [AAX_CParameter< T >](#).

14.101.3.52 virtual void AAX_IParameter::SetDisplayDelegate (AAX_IDisplayDelegateBase & *inDisplayDelegate*) [pure virtual]

Sets the parameter's display delegate.

Parameters

in	<i>inDisplayDelegate</i>	A reference to the parameter's new display delegate

Implemented in [AAX_CStatelessParameter](#), and [AAX_CParameter< T >](#).

14.101.3.53 virtual void AAX_IParameter::UpdateNormalizedValue (double *newNormalizedValue*) [pure virtual]

Sets the parameter's state given a normalized value.

This is the second half of the parameter setting operation that is initiated with a call to `SetValue()`. Parameters should not be set directly using this method; instead, use `SetValue()`.

Parameters

in	<i>newNormalizedValue</i>	Normalized value that will be used to set the parameter's new state

Implemented in [AAX_CStatelessParameter](#), and [AAX_CParameter< T >](#).

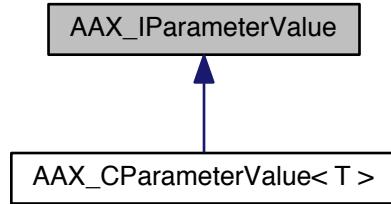
The documentation for this class was generated from the following file:

- [AAX_IParameter.h](#)

14.102 AAX_IParameterValue Class Reference

```
#include <AAX_IParameter.h>
```

Inheritance diagram for AAX_IParameterValue:



14.102.1 Description

An abstract interface representing a parameter value of arbitrary type.

:Internal to the AAX SDK

See also

[AAX_IParameter](#)

Public Member Functions

- virtual [~AAX_IParameterValue \(\)](#)
Virtual destructor.
- virtual [AAX_IParameterValue * Clone \(\) const =0](#)
Clones the parameter object.
- virtual [AAX_CParamID Identifier \(\) const =0](#)
Returns the parameter's unique identifier.

Typed accessors

- virtual bool [GetValueAsBool \(bool *value\) const =0](#)
Retrieves the parameter's value as a bool.
- virtual bool [GetValueAsInt32 \(int32_t *value\) const =0](#)
Retrieves the parameter's value as an int32_t.
- virtual bool [GetValueAsFloat \(float *value\) const =0](#)
Retrieves the parameter's value as a float.
- virtual bool [GetValueAsDouble \(double *value\) const =0](#)
Retrieves the parameter's value as a double.
- virtual bool [GetValueAsString \(AAX_IString *value\) const =0](#)
Retrieves the parameter's value as a string.

14.102.2 Constructor & Destructor Documentation

14.102.2.1 virtual AAX_IParameterValue::[~AAX_IParameterValue \(\) \[inline\], \[virtual\]](#)

Virtual destructor.

Note

This destructor MUST be virtual to prevent memory leaks.

14.102.3 Member Function Documentation**14.102.3.1 virtual AAX_IParameterValue* AAX_IParameterValue::Clone() const [pure virtual]**

Clones the parameter object.

Note

Does NOT set the automation delegate on the clone; ownership of the automation delegate and parameter registration/unregistration stays with the original parameter

Implemented in [AAX_CParameterValue< T >](#).

14.102.3.2 virtual AAX_CParamID AAX_IParameterValue::Identifier() const [pure virtual]

Returns the parameter's unique identifier.

This unique ID is used by the [Parameter Manager](#) and by outside applications to uniquely identify and target control messages. This value may not be changed after the parameter has been constructed.

Implemented in [AAX_CParameterValue< T >](#).

14.102.3.3 virtual bool AAX_IParameterValue::GetValueAsBool(bool * value) const [pure virtual]

Retrieves the parameter's value as a bool.

Parameters

<code>out</code>	<code>value</code>	The parameter's real value. Set only if conversion is successful.
------------------	--------------------	---

Return values

<code>true</code>	The conversion to bool was successful
<code>false</code>	The conversion to bool was unsuccessful

Implemented in [AAX_CParameterValue< T >](#), and [AAX_CParameterValue< T >](#).

14.102.3.4 virtual bool AAX_IParameterValue::GetValueAsInt32(int32_t * value) const [pure virtual]

Retrieves the parameter's value as an int32_t.

Parameters

<code>out</code>	<code>value</code>	The parameter's real value. Set only if conversion is successful.
------------------	--------------------	---

Return values

<code>true</code>	The conversion to int32_t was successful
<code>false</code>	The conversion to int32_t was unsuccessful

Implemented in [AAX_CParameterValue< T >](#), and [AAX_CParameterValue< T >](#).

14.102.3.5 virtual bool AAX_IParameterValue::GetValueAsFloat(float * value) const [pure virtual]

Retrieves the parameter's value as a float.

Parameters

<code>out</code>	<code>value</code>	The parameter's real value. Set only if conversion is successful.
------------------	--------------------	---

Return values

<code>true</code>	The conversion to float was successful
<code>false</code>	The conversion to float was unsuccessful

Implemented in [AAX_CParameterValue< T >](#), and [AAX_CParameterValue< T >](#).

14.102.3.6 virtual bool AAX_IParameterValue::GetValueAsDouble (double * value) const [pure virtual]

Retrieves the parameter's value as a double.

Parameters

<code>out</code>	<code>value</code>	The parameter's real value. Set only if conversion is successful.
------------------	--------------------	---

Return values

<code>true</code>	The conversion to double was successful
<code>false</code>	The conversion to double was unsuccessful

Implemented in [AAX_CParameterValue< T >](#), and [AAX_CParameterValue< T >](#).

14.102.3.7 virtual bool AAX_IParameterValue::GetValueAsString (AAX_IString * value) const [pure virtual]

Retrieves the parameter's value as a string.

Parameters

<code>out</code>	<code>value</code>	The parameter's real value. Set only if conversion is successful.
------------------	--------------------	---

Return values

<code>true</code>	The conversion to string was successful
<code>false</code>	The conversion to string was unsuccessful

Implemented in [AAX_CParameterValue< T >](#), and [AAX_CParameterValue< T >](#).

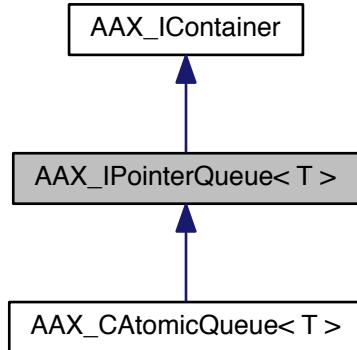
The documentation for this class was generated from the following file:

- [AAX_IParameter.h](#)

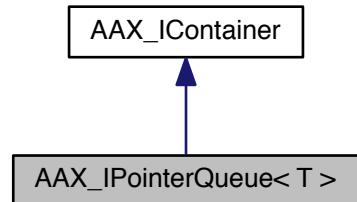
14.103 AAX_IPointerQueue< T > Class Template Reference

```
#include <AAX_IPointerQueue.h>
```

Inheritance diagram for AAX_IPointerQueue< T >:



Collaboration diagram for AAX_IPointerQueue< T >:



14.103.1 Description

```
template<typename T>class AAX_IPointerQueue< T >
```

Abstract interface for a basic FIFO queue of pointers-to-objects

Public Types

- **typedef T template_type**
The type used for this template instance.
- **typedef T * value_type**
The type of values stored in this queue.

Public Member Functions

- **virtual ~AAX_IPointerQueue ()**

- virtual void [Clear \(\)=0](#)
- virtual [AAX_IContainer::EStatus Push \(value_type inElem\)=0](#)
- virtual [value_type Pop \(\)=0](#)
- virtual [value_type Peek \(\) const =0](#)

14.103.2 Member Typedef Documentation

14.103.2.1 template<typename T> typedef T [AAX_IPointerQueue< T >::template_type](#)

The type used for this template instance.

14.103.2.2 template<typename T> typedef T* [AAX_IPointerQueue< T >::value_type](#)

The type of values stored in this queue.

14.103.3 Constructor & Destructor Documentation

14.103.3.1 template<typename T> virtual [AAX_IPointerQueue< T >::~AAX_IPointerQueue \(\) \[inline\], \[virtual\]](#)

14.103.4 Member Function Documentation

14.103.4.1 template<typename T> virtual void [AAX_IPointerQueue< T >::Clear \(\) \[pure virtual\]](#)

Note

This operation is NOT atomic

This does NOT call the destructor for any pointed-to elements; it only clears the pointer values in the queue

Implements [AAX_IContainer](#).

Implemented in [AAX_CAtomicQueue< T, S >](#), [AAX_CAtomicQueue< TNumeredParamStateList, 256 >](#), and [AAX_CAtomicQueue< const TParamValPair, 16 *kSynchronizedParameterQueueSize >](#).

14.103.4.2 template<typename T> virtual [AAX_IContainer::EStatus AAX_IPointerQueue< T >::Push \(value_type inElem \) \[pure virtual\]](#)

Push an element onto the queue

Call from: Write thread

Returns

[AAX_IContainer::eStatus_Success](#) if the push succeeded

Implemented in [AAX_CAtomicQueue< T, S >](#).

14.103.4.3 template<typename T> virtual [value_type AAX_IPointerQueue< T >::Pop \(\) \[pure virtual\]](#)

Pop the front element from the queue

Call from: Read thread

Returns

NULL if no element is available

Implemented in [AAX_CAtomicQueue< T, S >](#), [AAX_CAtomicQueue< TNumberedParamStateList, 256 >](#), and [AAX_CAtomicQueue< const TParamValPair, 16 *kSynchronizedParameterQueueSize >](#).

14.103.4.4 template<typename T> virtual value_type AAX_IPointerQueue< T >::Peek () const [pure virtual]

Get the current top element without popping it off of the queue

Call from: Read thread

Note

This value will change if another thread calls [Pop\(\)](#)

Implemented in [AAX_CAtomicQueue< T, S >](#), [AAX_CAtomicQueue< TNumberedParamStateList, 256 >](#), and [AAX_CAtomicQueue< const TParamValPair, 16 *kSynchronizedParameterQueueSize >](#).

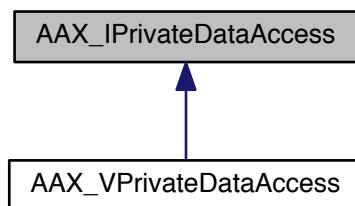
The documentation for this class was generated from the following file:

- [AAX_IPointerQueue.h](#)

14.104 AAX_IPrivateDataAccess Class Reference

```
#include <AAX_IPrivateDataAccess.h>
```

Inheritance diagram for AAX_IPrivateDataAccess:



14.104.1 Description

Interface to data access provided by host to plug-in.

[:Implemented by the AAX Host](#)

WARNING: [AAX_IPrivateDataAccess](#) objects are not reference counted and are not guaranteed to exist beyond the scope of the method(s) they are passed into.

See also

[AAX_IACFEffectDirectData::TimerWakeup](#)

Public Member Functions

- virtual [~AAX_IPrivateDataAccess \(\)](#)
- virtual [AAX_Result ReadPortDirect \(AAX_CFieldIndex inFieldIndex, const uint32_t inOffset, const uint32_t inSize, void *outBuffer\)=0](#)

Read data directly from DSP at the given port.

- virtual [AAX_Result WritePortDirect \(AAX_CFieldIndex inFieldIndex, const uint32_t inOffset, const uint32_t inSize, const void *inBuffer\)=0](#)

Write data directly to DSP at the given port.

14.104.2 Constructor & Destructor Documentation

14.104.2.1 virtual [AAX_IPrivateDataAccess::~AAX_IPrivateDataAccess \(\) \[inline\], \[virtual\]](#)

14.104.3 Member Function Documentation

14.104.3.1 virtual [AAX_Result AAX_IPrivateDataAccess::ReadPortDirect \(AAX_CFieldIndex inFieldIndex, const uint32_t inOffset, const uint32_t inSize, void * outBuffer \) \[pure virtual\]](#)

Read data directly from DSP at the given port.

Note

Blocking

Parameters

in	<i>inFieldIndex</i>	The port to read from.
in	<i>inOffset</i>	Offset into data to start reading.
in	<i>inSize</i>	Amount of data to read (in bytes).
out	<i>outBuffer</i>	Pointer to storage for data to be read into.

Implemented in [AAX_VPrivateDataAccess](#).

14.104.3.2 virtual [AAX_Result AAX_IPrivateDataAccess::WritePortDirect \(AAX_CFieldIndex inFieldIndex, const uint32_t inOffset, const uint32_t inSize, const void * inBuffer \) \[pure virtual\]](#)

Write data directly to DSP at the given port.

Note

Blocking

Parameters

in	<i>inFieldIndex</i>	The port to write to.
in	<i>inOffset</i>	Offset into data to begin writing.
in	<i>inSize</i>	Amount of data to write (in bytes).
in	<i>inBuffer</i>	Pointer to data being written.

Implemented in [AAX_VPrivateDataAccess](#).

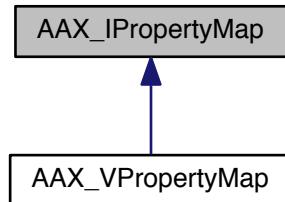
The documentation for this class was generated from the following file:

- [AAX_IPrivateDataAccess.h](#)

14.105 AAX_IPropertyMap Class Reference

```
#include <AAx_IPropertyMap.h>
```

Inheritance diagram for AAX_IPropertyMap:



14.105.1 Description

Generic plug-in description property map.

:Implemented by the AAX Host

Property Maps are used to associate specific sets of properties with plug-in description interfaces. For example, an audio processing component might register mono and stereo callbacks, or Native and TI callbacks, assigning each `ProcessProc` the applicable property mapping. This allows the host to determine the correct callback to use depending on the environment in which the plug-in is instantiated.

AAX does not require that every value in AAX IPropertyMap be assigned by the developer. Unassigned properties do not have defined default values; if a specific value is not assigned to one of an element's properties then the element must support any value for that property. For example, if an audio processing component does not define its callback's audio buffer length property, the host will assume that the callback will support any buffer length.

- To create a new property map: [AAX_IComponentDescriptor::NewPropertyMap\(\)](#)
- To copy an existing property map: [AAX_IComponentDescriptor::DuplicatePropertyMap\(\)](#)

Public Member Functions

- virtual ~[AAx_IPropertyMap](#) ()
- virtual [AAx_CBoolean GetProperty](#) ([AAx_EProperty](#) inProperty, [AAx_CPropertyValue](#) *outValue) const =0
Get a property value from a property map.
- virtual [AAx_CBoolean GetPointerProperty](#) ([AAx_EProperty](#) inProperty, const void **outValue) const =0
Get a property value from a property map with a pointer-sized value.
- virtual [AAx_Result AddProperty](#) ([AAx_EProperty](#) inProperty, [AAx_CPropertyValue](#) inValue)=0
Add a property to a property map.
- virtual [AAx_Result AddPointerProperty](#) ([AAx_EProperty](#) inProperty, const void *inValue)=0
Add a property to a property map with a pointer-sized value.
- virtual [AAx_Result AddPointerProperty](#) ([AAx_EProperty](#) inProperty, const char *inValue)=0
Add a property to a property map with a pointer-sized value.
- virtual [AAx_Result RemoveProperty](#) ([AAx_EProperty](#) inProperty)=0

- Remove a property from a property map.*
- virtual `AAX_Result AddPropertyWithIDArray (AAX_EProperty inProperty, const AAX_SPlugInIdentifierTriad *inPluginIDs, uint32_t inNumPluginIDs)=0`
Add an array of plug-in IDs to a property map.
 - virtual `AAX_CBoolean GetPropertyWithIDArray (AAX_EProperty inProperty, const AAX_SPlugInIdentifierTriad **outPluginIDs, uint32_t *outNumPluginIDs) const =0`
Get an array of plug-in IDs from a property map.
 - virtual `IACFUnknown * GetIUnknown ()=0`

14.105.2 Constructor & Destructor Documentation

14.105.2.1 virtual `AAX_IPropertyMap::~AAX_IPropertyMap () [inline], [virtual]`

14.105.3 Member Function Documentation

14.105.3.1 virtual `AAX_CBoolean AAX_IPropertyMap::GetProperty (AAX_EProperty inProperty, AAX_CPropertyValue * outValue) const [pure virtual]`

Get a property value from a property map.

Returns true if the selected property is supported, false if it is not

Parameters

in	<code>inProperty</code>	The property ID
out	<code>outValue</code>	The property value

Implemented in [AAX_VPropertyMap](#).

14.105.3.2 virtual `AAX_CBoolean AAX_IPropertyMap::GetPointerProperty (AAX_EProperty inProperty, const void ** outValue) const [pure virtual]`

Get a property value from a property map with a pointer-sized value.

Returns true if the selected property is supported, false if it is not

Parameters

in	<code>inProperty</code>	The property ID
out	<code>outValue</code>	The property value

Implemented in [AAX_VPropertyMap](#).

14.105.3.3 virtual `AAX_Result AAX_IPropertyMap::AddProperty (AAX_EProperty inProperty, AAX_CPropertyValue inValue) [pure virtual]`

Add a property to a property map.

Note

This method may return an error if adding the property was unsuccessful. If there is a failure when adding a required property then registration of the relevant description element must be abandoned and the plug-in's description logic should proceed to the next element.

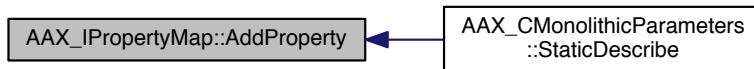
Parameters

in	<i>inProperty</i>	The property ID.
in	<i>inValue</i>	

Implemented in [AAX_VPropertyMap](#).

Referenced by `AAX_CMonolithicParameters::StaticDescribe()`.

Here is the caller graph for this function:



14.105.3.4 virtual AAX_Result AAX_IPropertyMap::AddPointerProperty (AAX_EProperty *inProperty*, const void * *inValue*) [pure virtual]

Add a property to a property map with a pointer-sized value.

Use this method to add properties which require a pointer-sized value. Do not use this method to add a property unless a pointer-sized value is explicitly specified in the property documentation.

Note

This method may return an error if adding the property was unsuccessful. If there is a failure when adding a required property then registration of the relevant description element must be abandoned and the plug-in's description logic should proceed to the next element.

Parameters

in	<i>inProperty</i>	The property ID.
in	<i>inValue</i>	

Implemented in [AAX_VPropertyMap](#).

14.105.3.5 virtual AAX_Result AAX_IPropertyMap::AddPointerProperty (AAX_EProperty *inProperty*, const char * *inValue*) [pure virtual]

Add a property to a property map with a pointer-sized value.

Use this method to add properties which require a pointer-sized value. Do not use this method to add a property unless a pointer-sized value is explicitly specified in the property documentation.

Note

This method may return an error if adding the property was unsuccessful. If there is a failure when adding a required property then registration of the relevant description element must be abandoned and the plug-in's description logic should proceed to the next element.

Parameters

in	<i>inProperty</i>	The property ID.
in	<i>inValue</i>	

Implemented in [AAX_VPropertyMap](#).

14.105.3.6 virtual AAX_Result AAX_IPropertyMap::RemoveProperty (AAX_EProperty *inProperty*) [pure virtual]

Remove a property from a property map.

Parameters

in	<i>inProperty</i>	The property ID.
----	-------------------	------------------

Implemented in [AAX_VPropertyMap](#).

14.105.3.7 virtual AAX_Result AAX_IPropertyMap::AddPropertyWithIDArray (AAX_EProperty *inProperty*, const AAX_SPlugInIdentifierTriad * *inPluginIDs*, uint32_t *inNumPluginIDs*) [pure virtual]

Add an array of plug-in IDs to a property map.

Parameters

in	<i>inProperty</i>	The property ID.
in	<i>inPluginIDs</i>	An array of AAX_SPlugInIdentifierTriad
in	<i>inNumPluginIDs</i>	The length of <i>iPluginIDs</i>

Implemented in [AAX_VPropertyMap](#).

14.105.3.8 virtual AAX_CBoolean AAX_IPropertyMap::GetPropertyWithIDArray (AAX_EProperty *inProperty*, const AAX_SPlugInIdentifierTriad ** *outPluginIDs*, uint32_t * *outNumPluginIDs*) const [pure virtual]

Get an array of plug-in IDs from a property map.

Parameters

in	<i>inProperty</i>	The property ID.
out	<i>outPluginIDs</i>	A pointer that will be set to reference an array of AAX_SPlugInIdentifierTriad
in	<i>outNumPluginIDs</i>	The length of <i>oPluginIDs</i>

Implemented in [AAX_VPropertyMap](#).

14.105.3.9 virtual IACFUnknown* AAX_IPropertyMap::GetIUnknown () [pure virtual]

Returns the most up-to-date underlying interface

Implemented in [AAX_VPropertyMap](#).

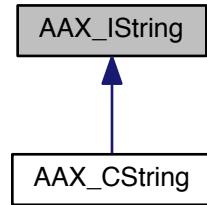
The documentation for this class was generated from the following file:

- [AAX_IPropertyMap.h](#)

14.106 AAX_IString Class Reference

```
#include <AAX_IString.h>
```

Inheritance diagram for AAX_IString:



14.106.1 Description

A simple string container that can be passed across a binary boundary. This class, for simplicity, is not versioned and thus can never change.

For a real string implementation, see [AAX_CString](#), which inherits from this interface, but provides a much richer string interface.

This object is not versioned with ACF for a variety of reasons, but the biggest implication of that is that THIS INTERFACE CAN NEVER CHANGE!

Public Member Functions

- virtual [~AAX_IString \(\)](#)
- virtual uint32_t [Length \(\) const](#) =0
- virtual uint32_t [MaxLength \(\) const](#) =0
- virtual const char * [Get \(\) const](#) =0
- virtual void [Set \(const char *iString\)=0](#)
- virtual [AAX_IString & operator= \(const AAX_IString &iOther\)=0](#)
- virtual [AAX_IString & operator= \(const char *iString\)=0](#)

14.106.2 Constructor & Destructor Documentation

14.106.2.1 virtual [AAX_IString::~AAX_IString \(\) \[inline\], \[virtual\]](#)

Virtual Destructor

14.106.3 Member Function Documentation

14.106.3.1 virtual uint32_t [AAX_IString::Length \(\) const \[pure virtual\]](#)

Length methods

Implemented in [AAX_CString](#).

Referenced by [AAX::String2Binary\(\)](#).

Here is the caller graph for this function:



14.106.3.2 `virtual uint32_t AAX_IString::MaxLength() const [pure virtual]`

Implemented in [AAX_CString](#).

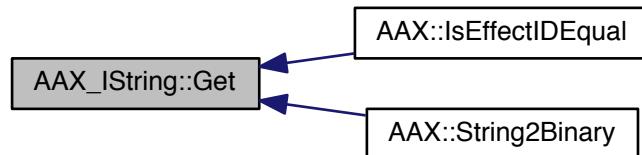
14.106.3.3 `virtual const char* AAX_IString::Get() const [pure virtual]`

C string methods

Implemented in [AAX_CString](#).

Referenced by `AAX::IsEffectIDEqual()`, and `AAX::String2Binary()`.

Here is the caller graph for this function:



14.106.3.4 `virtual void AAX_IString::Set(const char * iString) [pure virtual]`

Implemented in [AAX_CString](#).

14.106.3.5 `virtual AAX_IString& AAX_IString::operator=(const AAX_IString & iOther) [pure virtual]`

Assignment operators

Implemented in [AAX_CString](#).

14.106.3.6 `virtual AAX_IString& AAX_IString::operator=(const char * iString) [pure virtual]`

Implemented in [AAX_CString](#).

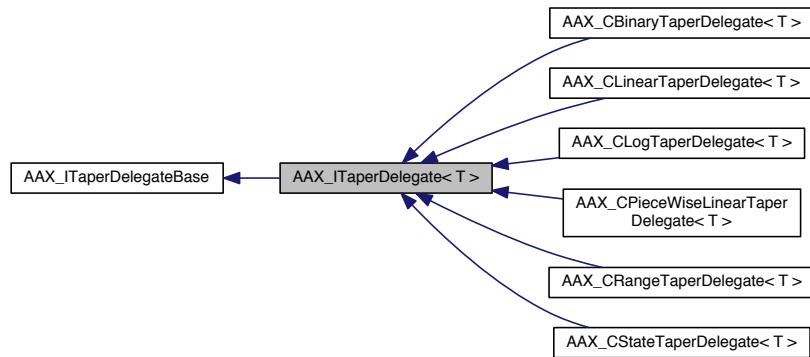
The documentation for this class was generated from the following file:

- [AAX_IString.h](#)

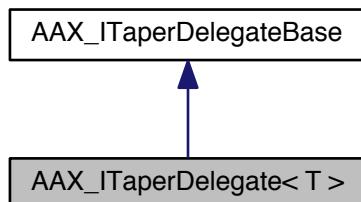
14.107 AAX_ITaperDelegate< T > Class Template Reference

```
#include <AAX_ITaperDelegate.h>
```

Inheritance diagram for AAX_ITaperDelegate< T >:



Collaboration diagram for AAX_ITaperDelegate< T >:



14.107.1 Description

```
template<typename T>class AAX_ITaperDelegate< T >
```

Classes for conversion to and from normalized parameter values.

Taper delegate interface template

Taper delegates are used to convert real parameter values to and from their normalized representations. All taper delegates implement the AAX_ITaperDelegate<T> interface template, which contains two conversion functions:

```
virtual T      NormalizedToReal(double normalizedValue) const = 0;
virtual double RealToNormalized(T realValue) const = 0;
```

In addition, tapers may incorporate logical value constraints via the following interface methods:

```

virtual T      GetMaximumValue() const = 0;
virtual T      GetMinimumValue() const = 0;
virtual T      ConstrainRealValue(T value) const = 0;

```

For more information, see the [AAx_ITaperDelegate](#) class documentation.

Public Member Functions

- virtual [AAx_ITaperDelegate](#) * **Clone** () const =0
Constructs and returns a copy of the taper delegate.
- virtual T **GetMaximumValue** () const =0
Returns the taper's maximum real value.
- virtual T **GetMinimumValue** () const =0
Returns the taper's minimum real value.
- virtual T **ConstrainRealValue** (T value) const =0
Applies a constraint to the value and returns the constrained value.
- virtual T **NormalizedToReal** (double normalizedValue) const =0
Converts a normalized value to a real value.
- virtual double **RealToNormalized** (T realValue) const =0
Normalizes a real parameter value.

14.107.2 Member Function Documentation

14.107.2.1 template<typename T> virtual AAX_ITaperDelegate* AAX_ITaperDelegate< T >::Clone () const [pure virtual]

Constructs and returns a copy of the taper delegate.

In general, this method's implementation can use a simple copy constructor:

```

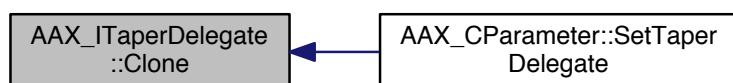
template <typename T>
AAX_CSubclassTaperDelegate<T>* AAX_CSubclassTaperDelegate<T>::Clone() const
{
    return new AAX_CSubclassTaperDelegate(*this);
}

```

Implemented in [AAx_CRangeTaperDelegate< T, RealPrecision >](#), [AAx_CPieceWiseLinearTaperDelegate< T, RealPrecision >](#), [AAx_CLogTaperDelegate< T, RealPrecision >](#), [AAx_CLinearTaperDelegate< T, RealPrecision >](#), [AAx_CStateTaperDelegate< T >](#), and [AAx_CBinaryTaperDelegate< T >](#).

Referenced by [AAx_CParameter< T >::SetTaperDelegate\(\)](#).

Here is the caller graph for this function:



14.107.2.2 template<typename T> virtual T AAX_ITaperDelegate< T >::GetMaximumValue() const [pure virtual]

Returns the taper's maximum real value.

Implemented in [AAACRangeTaperDelegate< T, RealPrecision >](#), [AAACPieceWiseLinearTaperDelegate< T, RealPrecision >](#), [AAACLogTaperDelegate< T, RealPrecision >](#), [AAACLinearTaperDelegate< T, RealPrecision >](#), [AAACStateTaperDelegate< T >](#), and [AAACBinaryTaperDelegate< T >](#).

14.107.2.3 template<typename T> virtual T AAX_ITaperDelegate< T >::GetMinimumValue() const [pure virtual]

Returns the taper's minimum real value.

Implemented in [AAACRangeTaperDelegate< T, RealPrecision >](#), [AAACPieceWiseLinearTaperDelegate< T, RealPrecision >](#), [AAACLogTaperDelegate< T, RealPrecision >](#), [AAACLinearTaperDelegate< T, RealPrecision >](#), [AAACStateTaperDelegate< T >](#), and [AAACBinaryTaperDelegate< T >](#).

14.107.2.4 template<typename T> virtual T AAX_ITaperDelegate< T >::ConstrainRealValue(T value) const [pure virtual]

Applies a constraint to the value and returns the constrained value.

This method is useful if the taper requires a constraint beyond simple minimum and maximum real value limits.

Note

This is the function that should actually enforces the constraints in [NormalizeToReal\(\)](#) and [RealToNormalized\(\)](#).

Parameters

in	value	The unconstrained value
----	-------	-------------------------

Implemented in [AAACRangeTaperDelegate< T, RealPrecision >](#), [AAACPieceWiseLinearTaperDelegate< T, RealPrecision >](#), [AAACLogTaperDelegate< T, RealPrecision >](#), [AAACLinearTaperDelegate< T, RealPrecision >](#), [AAACStateTaperDelegate< T >](#), and [AAACBinaryTaperDelegate< T >](#).

14.107.2.5 template<typename T> virtual T AAX_ITaperDelegate< T >::NormalizedToReal(double normalizedValue) const [pure virtual]

Converts a normalized value to a real value.

This is where the actual taper algorithm is implemented.

This function should perform the exact inverse of [RealToNormalized\(\)](#), to within the roundoff precision of the individual taper implementation.

Parameters

in	normalizedValue	The normalized value that will be converted
----	-----------------	---

Implemented in [AAACRangeTaperDelegate< T, RealPrecision >](#), [AAACPieceWiseLinearTaperDelegate< T, RealPrecision >](#), [AAACLogTaperDelegate< T, RealPrecision >](#), [AAACLinearTaperDelegate< T, RealPrecision >](#), [AAACStateTaperDelegate< T >](#), and [AAACBinaryTaperDelegate< T >](#).

14.107.2.6 template<typename T> virtual double AAX_ITaperDelegate< T >::RealToNormalized(T realValue) const [pure virtual]

Normalizes a real parameter value.

This is where the actual taper algorithm is implemented.

This function should perform the exact inverse of [NormalizedToReal\(\)](#), to within the roundoff precision of the individual taper implementation.

Parameters

in	<i>realValue</i>	The real parameter value that will be normalized
----	------------------	--

Implemented in [AAx_CRangeTaperDelegate< T, RealPrecision >](#), [AAx_CPieceWiseLinearTaperDelegate< T, RealPrecision >](#), [AAx_CLogTaperDelegate< T, RealPrecision >](#), [AAx_CLinearTaperDelegate< T, RealPrecision >](#), [AAx_CStateTaperDelegate< T >](#), and [AAx_CBinaryTaperDelegate< T >](#).

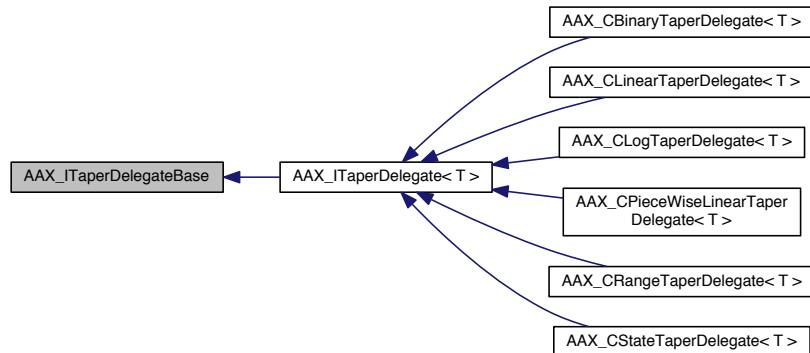
The documentation for this class was generated from the following file:

- [AAx_ITaperDelegate.h](#)

14.108 AAX_ITaperDelegateBase Class Reference

```
#include <AAx_ITaperDelegate.h>
```

Inheritance diagram for AAX_ITaperDelegateBase:



14.108.1 Description

Defines the taper conversion behavior for a parameter.

:Internal to the AAX SDK

This interface represents a delegate class to be used in conjunction with [AAx_IParameter](#). [AAx_IParameter](#) delegates all conversion operations between normalized and real parameter values to classes that meet this interface. You can think of [AAx_ITaperDelegate](#) subclasses as simple taper conversion routines that enable a specific taper or range conversion function on an arbitrary parameter.

To demonstrate the use of this interface, we will examine a simple call routine into a parameter:

1. The host application calls into the plug-in's [AAx_CParameterManager](#) with a Parameter ID and a new normalized parameter value. This new value could be coming from an automation lane, a control surface, or any other parameter control; from the plug-in's perspective, these are all identical.

2. The [AAX_CParameterManager](#) finds the specified [AAX_CParameter](#) and calls [AAX_IParameter::SetNormalizedValue\(\)](#) on that parameter

3. [AAX_IParameter::SetNormalizedValue\(\)](#) results in a call into the parameter's concrete taper delegate to convert the normalized value to a real value.

Using this pattern, the parameter manager is able to use real parameter values without actually knowing how to perform the conversion between normalized and real values.

The inverse of the above example can also happen, e.g. when a control is updated from within the data model. In this case, the parameter can call into its concrete taper delegate in order to normalize the updated value, which can then be passed on to any observers that require normalized values, such as the host app.

For more information about the parameter manager, see the [Parameter Manager](#) documentation page.

Public Member Functions

- virtual [~AAX_ITaperDelegateBase \(\)](#)

Virtual destructor.

14.108.2 Constructor & Destructor Documentation

14.108.2.1 virtual AAX_ITaperDelegateBase::~AAX_ITaperDelegateBase() [inline], [virtual]

Virtual destructor.

Note

This destructor MUST be virtual to prevent memory leaks.

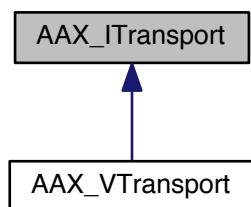
The documentation for this class was generated from the following file:

- [AAX_ITaperDelegate.h](#)

14.109 AAX_ITransport Class Reference

```
#include <AAX_ITransport.h>
```

Inheritance diagram for AAX_ITransport:



14.109.1 Description

Interface to information about the host's transport state.

:Implemented by the AAX Host

Plug-ins that use this interface should describe [AAX_eProperty_UsesTransport](#) as 1

Acquire this interface using [AAX_IMIDINode::GetTransport\(\)](#). Classes that inherit from [AAX_CEffectParameters](#) or [AAX_CEffectGUI](#) can also use [AAX_CEffectParameters::Transport\(\)](#) / [AAX_CEffectGUI::Transport\(\)](#).

Public Member Functions

- virtual [~AAX_ITransport \(\)](#)
Virtual destructor.
- virtual [AAX_Result GetCurrentTempo \(double *TempoBPM\) const =0](#)
CALL: Gets the current tempo.
- virtual [AAX_Result GetCurrentMeter \(int32_t *MeterNumerator, int32_t *MeterDenominator\) const =0](#)
CALL: Gets the current meter.
- virtual [AAX_Result IsTransportPlaying \(bool *isPlaying\) const =0](#)
CALL: Indicates whether or not the transport is playing back.
- virtual [AAX_Result GetCurrentTickPosition \(int64_t *TickPosition\) const =0](#)
CALL: Gets the current tick position.
- virtual [AAX_Result GetCurrentLoopPosition \(bool *bLooping, int64_t *LoopStartTick, int64_t *LoopEndTick\) const =0](#)
CALL: Gets current information on loop playback.
- virtual [AAX_Result GetCurrentNativeSampleLocation \(int64_t *SampleLocation\) const =0](#)
CALL: Gets the current playback location of the native audio engine.
- virtual [AAX_Result GetCustomTickPosition \(int64_t *oTickPosition, int64_t iSampleLocation\) const =0](#)
CALL: Given an absolute sample position, gets the corresponding tick position.
- virtual [AAX_Result GetBarBeatPosition \(int32_t *Bars, int32_t *Beats, int64_t *DisplayTicks, int64_t *SampleLocation\) const =0](#)
CALL: Given an absolute sample position, gets the corresponding bar and beat position.
- virtual [AAX_Result GetTicksPerQuarter \(uint32_t *ticks\) const =0](#)
CALL: Retrieves the number of ticks per quarter note.
- virtual [AAX_Result GetCurrentTicksPerBeat \(uint32_t *ticks\) const =0](#)
CALL: Retrieves the number of ticks per beat.
- virtual [AAX_Result GetTimelineSelectionStartPosition \(int64_t *oSampleLocation\) const =0](#)
CALL: Retrieves the current absolute sample position of the beginning of the current transport selection.
- virtual [AAX_Result GetTimeCodeInfo \(AAX_EFrameRate *oFrameRate, int32_t *oOffset\) const =0](#)
CALL: Retrieves the current time code frame rate and offset.
- virtual [AAX_Result GetFeetFramesInfo \(AAX_EFootFramesRate *oFootFramesRate, int64_t *oOffset\) const =0](#)
CALL: Retrieves the current timecode feet/frames rate and offset.
- virtual [AAX_Result IsMetronomeEnabled \(int32_t *isEnabled\) const =0](#)
Sets isEnabled to true if the metronome is enabled.

14.109.2 Constructor & Destructor Documentation

14.109.2.1 `virtual AAX_ITransport::~AAX_ITransport() [inline], [virtual]`

Virtual destructor.

Note

This destructor MUST be virtual to prevent memory leaks.

14.109.3 Member Function Documentation

14.109.3.1 `virtual AAX_Result AAX_ITransport::GetCurrentTempo(double * TempoBPM) const [pure virtual]`

CALL: Gets the current tempo.

Returns the tempo corresponding to the current position of the transport counter

Note

The resolution of the tempo returned here is based on the host's tempo resolution, so it will match the tempo displayed in the host. Use [GetCurrentTicksPerBeat\(\)](#) to calculate the tempo resolution note.

Parameters

<code>out</code>	<code>TempoBPM</code>	The current tempo in beats per minute
------------------	-----------------------	---------------------------------------

Implemented in [AAX_VTransport](#).

14.109.3.2 `virtual AAX_Result AAX_ITransport::GetCurrentMeter(int32_t * MeterNumerator, int32_t * MeterDenominator) const [pure virtual]`

CALL: Gets the current meter.

Returns the meter corresponding to the current position of the transport counter

Parameters

<code>out</code>	<code>MeterNumerator</code>	The numerator portion of the meter
<code>out</code>	<code>MeterDenominator</code>	The denominator portion of the meter

Implemented in [AAX_VTransport](#).

14.109.3.3 `virtual AAX_Result AAX_ITransport::IsTransportPlaying(bool *.isPlaying) const [pure virtual]`

CALL: Indicates whether or not the transport is playing back.

Parameters

<code>out</code>	<code>isPlaying</code>	true if the transport is currently in playback
------------------	------------------------	--

Implemented in [AAX_VTransport](#).

14.109.3.4 `virtual AAX_Result AAX_ITransport::GetCurrentTickPosition(int64_t * TickPosition) const [pure virtual]`

CALL: Gets the current tick position.

Returns the current tick position corresponding to the current transport position. One "Tick" is represented here as 1/960000 of a quarter note. That is, there are 960,000 of these ticks in a quarter note.

Host Compatibility Notes The tick resolution here is different than that of the tick displayed in Pro Tools. "Display ticks" (as they are called) are 1/960 of a quarter note.

Parameters

out	<i>TickPosition</i>	The tick position value
-----	---------------------	-------------------------

Implemented in [AAX_VTransport](#).

14.109.3.5 virtual AAX_Result AAX_ITransport::GetCurrentLoopPosition (bool * *bLooping*, int64_t * *LoopStartTick*, int64_t * *LoopEndTick*) const [pure virtual]

CALL: Gets current information on loop playback.

Host Compatibility Notes This does not indicate anything about the status of the "Loop Record" option. Even when the host is configured to loop playback, looping may not occur if certain conditions are not met (i.e. the length of the selection is too short)

Parameters

out	<i>bLooping</i>	true if the host is configured to loop playback
out	<i>LoopStartTick</i>	The starting tick position of the selection being looped (see GetCurrentTickPosition())
out	<i>LoopEndTick</i>	The ending tick position of the selection being looped (see GetCurrentTickPosition())

Implemented in [AAX_VTransport](#).

14.109.3.6 virtual AAX_Result AAX_ITransport::GetCurrentNativeSampleLocation (int64_t * *SampleLocation*) const [pure virtual]

CALL: Gets the current playback location of the native audio engine.

When called from a ProcessProc render callback, this method will provide the absolute sample location at the beginning of the callback's audio buffers.

When called from [AAX_IEffectParameters::RenderAudio_Hybrid\(\)](#), this method will provide the absolute sample location for the samples in the method's **output** audio buffers. To calculate the absolute sample location for the samples in the method's input buffers (i.e. the timeline location where the samples originated) subtract the value provided by [AAX_IController::GetHybridSignalLatency\(\)](#) from this value.

When called from a non-real-time thread, this method will provide the current location of the samples being processed by the plug-in's ProcessProc on its real-time processing thread.

Note

This method only returns a value during playback. It cannot be used to determine, e.g., the location of the timeline selector while the host is not in playback.

Parameters

out	<i>SampleLocation</i>	Absolute sample location of the first sample in the current native processing buffer
-----	-----------------------	--

Implemented in [AAX_VTransport](#).

14.109.3.7 virtual AAX_Result AAX_ITransport::GetCustomTickPosition (int64_t * *oTickPosition*, int64_t *iSampleLocation*) const [pure virtual]

CALL: Given an absolute sample position, gets the corresponding tick position.

Host Compatibility Notes There is a minor performance cost associated with using this API in Pro Tools. It should not be used excessively without need.

Parameters

out	<i>oTickPosition</i>	the timeline tick position corresponding to <i>iSampleLocation</i>
in	<i>iSampleLocation</i>	An absolute sample location (see GetCurrentNativeSampleLocation())

Implemented in [AAX_VTransport](#).

14.109.3.8 virtual AAX_Result AAX_ITransport::GetBarBeatPosition (int32_t * *Bars*, int32_t * *Beats*, int64_t * *DisplayTicks*, int64_t *SampleLocation*) const [pure virtual]

CALL: Given an absolute sample position, gets the corresponding bar and beat position.

Host Compatibility Notes There is a minor performance cost associated with using this API in Pro Tools. It should not be used excessively without need.

Parameters

out	<i>Bars</i>	The bar corresponding to <i>SampleLocation</i>
out	<i>Beats</i>	The beat corresponding to <i>SampleLocation</i>
out	<i>DisplayTicks</i>	The ticks corresponding to <i>SampleLocation</i>
in	<i>SampleLocation</i>	An absolute sample location (see GetCurrentNativeSampleLocation())

Implemented in [AAX_VTransport](#).

14.109.3.9 virtual AAX_Result AAX_ITransport::GetTicksPerQuarter (uint32_t * *ticks*) const [pure virtual]

CALL: Retrieves the number of ticks per quarter note.

Parameters

out	<i>ticks</i>
-----	--------------

Implemented in [AAX_VTransport](#).

14.109.3.10 virtual AAX_Result AAX_ITransport::GetCurrentTicksPerBeat (uint32_t * *ticks*) const [pure virtual]

CALL: Retrieves the number of ticks per beat.

Parameters

out	<i>ticks</i>
-----	--------------

Implemented in [AAX_VTransport](#).

14.109.3.11 virtual AAX_Result AAX_ITransport::GetTimelineSelectionStartPosition (int64_t * *oSampleLocation*) const [pure virtual]

CALL: Retrieves the current absolute sample position of the beginning of the current transport selection.

Note

This method is part of the [version 2 transport interface](#)

Parameters

out	<i>oSampleLocation</i>	
-----	------------------------	--

Implemented in [AAX_VTransport](#).

```
14.109.3.12 virtual AAX_Result AAX_ITransport::GetTimeCodeInfo ( AAX_EFrameRate * oFrameRate, int32_t * oOffset ) const [pure virtual]
```

CALL: Retrieves the current time code frame rate and offset.

Note

This method is part of the [version 2 transport interface](#)

Parameters

out	<i>oFrameRate</i>	
out	<i>oOffset</i>	

Implemented in [AAX_VTransport](#).

```
14.109.3.13 virtual AAX_Result AAX_ITransport::GetFeetFramesInfo ( AAX_EFootFramesRate * oFootFramesRate, int64_t * oOffset ) const [pure virtual]
```

CALL: Retrieves the current timecode feet/frames rate and offset.

Note

This method is part of the [version 2 transport interface](#)

Parameters

out	<i>oFootFramesRate</i>	
out	<i>oOffset</i>	

Implemented in [AAX_VTransport](#).

```
14.109.3.14 virtual AAX_Result AAX_ITransport::IsMetronomeEnabled ( int32_t * isEnabled ) const [pure virtual]
```

Sets isEnabled to true if the metronome is enabled.

Note

This method is part of the [version 2 transport interface](#)

Parameters

out	<i>isEnabled</i>	
-----	------------------	--

Implemented in [AAX_VTransport](#).

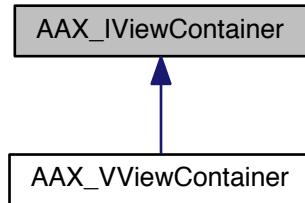
The documentation for this class was generated from the following file:

- [AAX_ITransport.h](#)

14.110 AAX_IViewContainer Class Reference

```
#include <AAx_IViewContainer.h>
```

Inheritance diagram for AAX_IViewContainer:



14.110.1 Description

Interface for the AAX host's view of a single instance of an effect. Used both by clients of the AAX host and by effect components.

:Implemented by the AAX Host

Public Member Functions

- virtual [~AAX_IViewContainer](#) (void)

View and GUI state queries

- virtual int32_t [GetType](#) ()=0
Returns the raw view type as one of [AAx_EViewContainer_Type](#).
- virtual void * [GetPtr](#) ()=0
Returns a pointer to the raw view.
- virtual [AAx_Result](#) [GetModifiers](#) (uint32_t *outModifiers)=0
Queries the host for the current [modifier keys](#).

View change requests

- virtual [AAx_Result](#) [SetViewSize](#) ([AAx_Point](#) &inSize)=0
Request a change to the main view size.

Host event handlers

These methods are used to pass plug-in GUI events to the host for handling. Events should always be passed on in this way when there is a possibility of the host overriding the event with its own behavior.

For example, in Pro Tools a command-control-option-click on any automatable plug-in parameter editor should bring up that parameter's automation pop-up menu, and a control-right click should display the parameter's automation lane in the Pro Tools Edit window. In order for Pro Tools to handle these events, the plug-in must pass them on using [HandleParameterMouseDown\(\)](#)

For each of these methods:

- *AAX_SUCCESS* is returned if the event was successfully handled by the host. In most cases, no further action will be required from the plug-in after the host successfully handles an event.
- *AAX_ERROR_UNIMPLEMENTED* is returned if the event was not handled by the host. In this case, the plug-in should perform its own event handling.
- virtual `AAX_Result HandleParameterMouseDown (AAX_CParamID inParamID, uint32_t inModifiers)=0`
Alert the host to a mouse down event.
- virtual `AAX_Result HandleParameterMouseDrag (AAX_CParamID inParamID, uint32_t inModifiers)=0`
Alert the host to a mouse drag event.
- virtual `AAX_Result HandleParameterMouseUp (AAX_CParamID inParamID, uint32_t inModifiers)=0`
Alert the host to a mouse up event.
- virtual `AAX_Result HandleMultipleParametersMouseDown (const AAX_CParamID *inParamIDs, uint32_t inNumOfParams, uint32_t inModifiers)=0`
Alert the host to a mouse down event.
- virtual `AAX_Result HandleMultipleParametersMouseDrag (const AAX_CParamID *inParamIDs, uint32_t inNumOfParams, uint32_t inModifiers)=0`
Alert the host to a mouse drag event.
- virtual `AAX_Result HandleMultipleParametersMouseUp (const AAX_CParamID *inParamIDs, uint32_t inNumOfParams, uint32_t inModifiers)=0`
Alert the host to a mouse up event.

14.110.2 Constructor & Destructor Documentation

14.110.2.1 virtual `AAX_IViewContainer::~AAX_IViewContainer (void) [inline], [virtual]`

14.110.3 Member Function Documentation

14.110.3.1 virtual `int32_t AAX_IViewContainer::GetType () [pure virtual]`

Returns the raw view type as one of *AAX_EViewContainer_Type*.

Implemented in *AAX_VViewContainer*.

14.110.3.2 virtual `void* AAX_IViewContainer::GetPtr () [pure virtual]`

Returns a pointer to the raw view.

Implemented in *AAX_VViewContainer*.

14.110.3.3 virtual `AAX_Result AAX_IViewContainer::GetModifiers (uint32_t *outModifiers) [pure virtual]`

Queries the host for the current modifier keys.

This method returns a bit mask with bits set for each of the currently active modifier keys. This method does not return the state of the *AAX_eModifiers_SecondaryButton*.

Host Compatibility Notes Although this method allows plug-ins to acquire the current state of the Windows key (normally blocked by Pro Tools), plug-ins should not use key combinations that require this key.

Parameters

<code>out</code>	<code>outModifiers</code>	Current modifiers as a bitmask of AAX_EModifiers
------------------	---------------------------	--

Implemented in [AAX_VViewContainer](#).

14.110.3.4 virtual AAX_Result AAX_IViewContainer::SetViewSize ([AAX_Point & inSize](#)) [pure virtual]

Request a change to the main view size.

Note

- For compatibility with the smallest supported displays, plug-in GUI dimensions should not exceed 749x617 pixels, or 749x565 pixels for plug-ins with sidechain support.

Parameters

<code>in</code>	<code>inSize</code>	The new size to which the plug-in view should be set
-----------------	---------------------	--

Implemented in [AAX_VViewContainer](#).

14.110.3.5 virtual AAX_Result AAX_IViewContainer::HandleParameterMouseDown ([AAX_CParamID inParamID](#), [uint32_t inModifiers](#)) [pure virtual]

Alert the host to a mouse down event.

Parameters

<code>in</code>	<code>inParamID</code>	ID of the parameter whose control is being edited
<code>in</code>	<code>inModifiers</code>	A bitmask of AAX_EModifiers values

Implemented in [AAX_VViewContainer](#).

14.110.3.6 virtual AAX_Result AAX_IViewContainer::HandleParameterMouseDrag ([AAX_CParamID inParamID](#), [uint32_t inModifiers](#)) [pure virtual]

Alert the host to a mouse drag event.

Warning

The host may return [AAX_ERROR_UNIMPLEMENTED](#) for this event even if the host did handle the corresponding mouse down event. A plug-in should ignore any following mouse drag and mouse up events that correspond to a host-managed mouse down event. ([PTSW-195209](#) / [PT-218474](#))

Parameters

<code>in</code>	<code>inParamID</code>	ID of the parameter whose control is being edited
<code>in</code>	<code>inModifiers</code>	A bitmask of AAX_EModifiers values

Implemented in [AAX_VViewContainer](#).

14.110.3.7 virtual AAX_Result AAX_IViewContainer::HandleParameterMouseUp ([AAX_CParamID inParamID](#), [uint32_t inModifiers](#)) [pure virtual]

Alert the host to a mouse up event.

Warning

The host may return [AAX_ERROR_UNIMPLEMENTED](#) for this event even if the host did handle the corresponding mouse down event. A plug-in should ignore any following mouse drag and mouse up events that correspond to a host-managed mouse down event. ([PTSW-195209](#) / [PT-218474](#))

Parameters

in	<i>inParamID</i>	ID of the parameter whose control is being edited
in	<i>inModifiers</i>	A bitmask of AAX_EModifiers values

Implemented in [AAX_VViewContainer](#).

14.110.3.8 virtual AAX_Result AAX_IViewContainer::HandleMultipleParametersMouseDown (const AAX_CParamID * *inParamIDs*, uint32_t *inNumOfParams*, uint32_t *inModifiers*) [pure virtual]

Alert the host to a mouse down event.

Parameters

in	<i>inParamIDs</i>	IDs of the parameters that belong to the same GUI element whose controls are being edited
in	<i>inNumOfParams</i>	Number of parameter IDs
in	<i>inModifiers</i>	A bitmask of AAX_EModifiers values

Implemented in [AAX_VViewContainer](#).

14.110.3.9 virtual AAX_Result AAX_IViewContainer::HandleMultipleParametersMouseDrag (const AAX_CParamID * *inParamIDs*, uint32_t *inNumOfParams*, uint32_t *inModifiers*) [pure virtual]

Alert the host to a mouse drag event.

Warning

The host may return [AAX_ERROR_UNIMPLEMENTED](#) for this event even if the host did handle the corresponding mouse down event. A plug-in should ignore any following mouse drag and mouse up events that correspond to a host-managed mouse down event. ([PTSW-195209](#) / [PT-218474](#))

Parameters

in	<i>inParamIDs</i>	IDs of the parameters that belong to the same GUI element whose controls are being edited
in	<i>inNumOfParams</i>	Number of parameter IDs
in	<i>inModifiers</i>	A bitmask of AAX_EModifiers values

Implemented in [AAX_VViewContainer](#).

14.110.3.10 virtual AAX_Result AAX_IViewContainer::HandleMultipleParametersMouseUp (const AAX_CParamID * *inParamIDs*, uint32_t *inNumOfParams*, uint32_t *inModifiers*) [pure virtual]

Alert the host to a mouse up event.

Warning

The host may return [AAX_ERROR_UNIMPLEMENTED](#) for this event even if the host did handle the corresponding mouse down event. A plug-in should ignore any following mouse drag and mouse up events that correspond to a host-managed mouse down event. ([PTSW-195209](#) / [PT-218474](#))

Parameters

in	<i>inParamIDs</i>	IDs of the parameters that belong to the same GUI element whose controls are being edited
in	<i>inNumOfParams</i>	Number of parameter IDs
in	<i>inModifiers</i>	A bitmask of AAX_EModifiers values

Implemented in [AAX_VViewContainer](#).

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

- [AAX_IViewContainer.h](#)

14.111 AAX_Map Class Reference

```
#include <AAX_Map.h>
```

Public Member Functions

- [AAX_Map \(\)](#)
- [~AAX_Map \(\)](#)
- void [SetCoefficients \(int aSize, double *alnpX, double *alnpY\)](#)
- void [GetCoefficient \(int alIndex, double *aOutX, double *aOutY\)](#)
- int [GetUpperBoundIndex \(double inp\)](#)
- double [GetX \(int alIndex\)](#)
- double [GetY \(int alIndex\)](#)
- double [GetFirstX \(\)](#)
- double [GetFirstY \(\)](#)
- double [GetLastX \(\)](#)
- double [GetLastY \(\)](#)
- int [GetSize \(\)](#)

14.111.1 Constructor & Destructor Documentation

14.111.1.1 [AAX_Map::AAX_Map \(\) \[inline\]](#)

14.111.1.2 [AAX_Map::~AAX_Map \(\) \[inline\]](#)

14.111.2 Member Function Documentation

14.111.2.1 [void AAX_Map::SetCoefficients \(int aSize, double * alnpX, double * alnpY \)](#)

14.111.2.2 [void AAX_Map::GetCoefficient \(int alIndex, double * aOutX, double * aOutY \)](#)

14.111.2.3 [int AAX_Map::GetUpperBoundIndex \(double inp \)](#)

14.111.2.4 [double AAX_Map::GetX \(int alIndex \) \[inline\]](#)

14.111.2.5 [double AAX_Map::GetY \(int alIndex \) \[inline\]](#)

14.111.2.6 [double AAX_Map::GetFirstX \(\) \[inline\]](#)

14.111.2.7 [double AAX_Map::GetFirstY \(\) \[inline\]](#)

14.111.2.8 [double AAX_Map::GetLastX \(\) \[inline\]](#)

14.111.2.9 double AAX_Map::GetLastY() [inline]

14.111.2.10 int AAX_Map::GetSize() [inline]

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

- [AAX_Map.h](#)

14.112 AAX_Point Struct Reference

```
#include <AAX_GUITypes.h>
```

14.112.1 Description

Data structure representing a two-dimensional coordinate point.

Comparison operators give preference to `vert`

Public Member Functions

- [AAX_Point \(float v, float h\)](#)
- [AAX_Point \(void\)](#)

Public Attributes

- float `vert`
- float `horz`

14.112.2 Constructor & Destructor Documentation

14.112.2.1 `AAX_Point::AAX_Point (float v, float h)` [inline]

14.112.2.2 `AAX_Point::AAX_Point (void)` [inline]

14.112.3 Member Data Documentation

14.112.3.1 float `AAX_Point::vert`

Referenced by `operator<()`, `operator<=()`, and `operator==()`.

14.112.3.2 float `AAX_Point::horz`

Referenced by `operator<()`, `operator<=()`, and `operator==()`.

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

- [AAX_GUITypes.h](#)

14.113 AAX_Rect Struct Reference

```
#include <AAX_GUITypes.h>
```

14.113.1 Description

Data structure representing a rectangle in a two-dimensional coordinate plane.

Public Member Functions

- [AAX_Rect](#) (float t, float l, float w, float h)
- [AAX_Rect](#) (void)

Public Attributes

- float [top](#)
- float [left](#)
- float [width](#)
- float [height](#)

14.113.2 Constructor & Destructor Documentation

14.113.2.1 [AAX_Rect::AAX_Rect \(float t, float l, float w, float h \)](#) [inline]

14.113.2.2 [AAX_Rect::AAX_Rect \(void \)](#) [inline]

14.113.3 Member Data Documentation

14.113.3.1 float [AAX_Rect::top](#)

Referenced by operator==().

14.113.3.2 float [AAX_Rect::left](#)

Referenced by operator==().

14.113.3.3 float [AAX_Rect::width](#)

Referenced by operator==().

14.113.3.4 float [AAX_Rect::height](#)

Referenced by operator==().

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

- [AAX_GUITypes.h](#)

14.114 AAX_SHybridRenderInfo Struct Reference

```
#include <AACFEffectParameters.h>
```

14.114.1 Description

Hybrid render processing context.

See also

AAX_IACFEfectParameters_V2::RenderAudio_Hybrid()

Public Attributes

- float ** mAudioInputs
 - int32_t * mNumAudioInputs
 - float ** mAudioOutputs
 - int32_t * mNumAudioOutputs
 - int32_t * mNumSamples
 - AAX_CTimestamp * mClock

14.114.2 Member Data Documentation

14.114.2.1 float** AAX_SHybridRenderInfo::mAUDIOInputs

14.114.2.2 int32_t* AAX_SHybridRenderInfo::mNumAudioInputs

14.114.2.3 float** AAX_SHybridRenderInfo::mAudioOutputs

14.114.2.4 int32_t* AAX_SHybridRenderInfo::mNumAudioOutputs

14.114.2.5 int32_t* AAX_SHybridRenderInfo::mNumSamples

14.114.2.6 AAX_CTimestamp* AAX_SHybridRenderInfo::mClock

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

- [AAX_IACFEfectParameters.h](#)

14.115 AAX_SInstrumentPrivateData Struct Reference

```
#include <AAX_CMonolithicParameters.h>
```

Collaboration diagram for AAX_SInstrumentPrivateData:



14.115.1 Description

Utility struct for [AAX_CMonolithicParameters](#).

This is an implementation detail of [AAX_CMonolithicParameters](#); you should never need to interact with this structure directly.

Public Attributes

- [AAX_CMonolithicParameters * mMonolithicParametersPtr](#)
A pointer to the instrument's data model.

14.115.2 Member Data Documentation

14.115.2.1 [AAX_CMonolithicParameters* AAX_SInstrumentPrivateData::mMonolithicParametersPtr](#)

A pointer to the instrument's data model.

You should never need to use this since the data model is available directly from within the virtual [AAX_CMonolithicParameters::RenderAudio\(\)](#) function.

Referenced by [AAX_CMonolithicParameters::ResetFieldData\(\)](#), and [AAX_CMonolithicParameters::StaticRenderAudio\(\)](#).

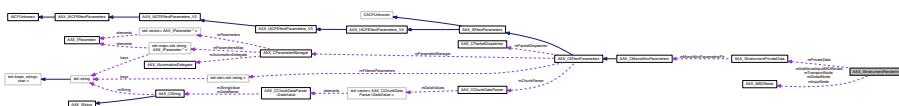
The documentation for this struct was generated from the following file:

- [AAX_CMonolithicParameters.h](#)

14.116 AAX_SInstrumentRenderInfo Struct Reference

```
#include <AAX_CMonolithicParameters.h>
```

Collaboration diagram for AAX_SInstrumentRenderInfo:



14.116.1 Description

Information used to parameterize [AAX_CMonolithicParameters::RenderAudio\(\)](#)

Public Attributes

- float ** [mAUDIOInputs](#)
Audio input buffers.
- float ** [mAUDIOOutputs](#)
Audio output buffers, including any aux output stems.
- int32_t * [mNumSamples](#)
Number of samples in each buffer. Bounded as per [AAE_EAudioBufferLengthNative](#). The exact value can vary from buffer to buffer.
- [AAX_CTimestamp * mClock](#)
Pointer to the global running time clock.
- [AAX_IMIDINode * mInputNode](#)
Buffered local MIDI input node. Used for incoming MIDI messages directed to the instrument.
- [AAX_IMIDINode * mGlobalNode](#)
Buffered global MIDI input node. Used for global events like beat updates in metronomes.
- [AAX_IMIDINode * mTransportNode](#)
Transport MIDI node. Used for querying the state of the MIDI transport.

- **AAX_IMIDINode * mAdditionalInputMIDINodes [kMaxAdditionalMIDINodes]**
List of additional input MIDI nodes, if your plugin needs them.
- **AAX_SInstrumentPrivateData * mPrivateData**
Struct containing private data relating to the instance. You should not need to use this data.
- **float ** mMeters**
Array of meter taps. One meter value should be entered per tap for each render call.
- **int64_t * mCurrentStateNum**
State counter.

14.116.2 Member Data Documentation

14.116.2.1 float** AAX_SInstrumentRenderInfo::mAUDIOInputs

Audio input buffers.

14.116.2.2 float** AAX_SInstrumentRenderInfo::mAUDIOOutputs

Audio output buffers, including any aux output stems.

14.116.2.3 int32_t* AAX_SInstrumentRenderInfo::mNumSamples

Number of samples in each buffer. Bounded as per [AAE_EAudioBufferLengthNative](#). The exact value can vary from buffer to buffer.

14.116.2.4 AAX_CTimestamp* AAX_SInstrumentRenderInfo::mClock

Pointer to the global running time clock.

14.116.2.5 AAX_IMIDINode* AAX_SInstrumentRenderInfo::mInputNode

Buffered local MIDI input node. Used for incoming MIDI messages directed to the instrument.

14.116.2.6 AAX_IMIDINode* AAX_SInstrumentRenderInfo::mGlobalNode

Buffered global MIDI input node. Used for global events like beat updates in metronomes.

14.116.2.7 AAX_IMIDINode* AAX_SInstrumentRenderInfo::mTransportNode

Transport MIDI node. Used for querying the state of the MIDI transport.

14.116.2.8 AAX_IMIDINode* AAX_SInstrumentRenderInfo::mAdditionalInputMIDINodes[kMaxAdditionalMIDINodes]

List of additional input MIDI nodes, if your plugin needs them.

14.116.2.9 AAX_SInstrumentPrivateData* AAX_SInstrumentRenderInfo::mPrivateData

Struct containing private data relating to the instance. You should not need to use this data.

14.116.2.10 float** AAX_SInstrumentRenderInfo::mMeters

Array of meter taps. One meter value should be entered per tap for each render call.

14.116.2.11 int64_t* AAX_SInstrumentRenderInfo::mCurrentStateNum

State counter.

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

- [AAX_CMonolithicParameters.h](#)

14.117 AAX_SInstrumentSetupInfo Struct Reference

```
#include <AAX_CMonolithicParameters.h>
```

14.117.1 Description

Information used to describe the instrument.

See also

[AAX_CMonolithicParameters::StaticDescribe\(\)](#)

Public Member Functions

- [AAX_SInstrumentSetupInfo \(\)](#)

Default constructor.

Public Attributes

- bool [mNeedsGlobalMIDI](#)
Does the instrument use a global MIDI input node?
- const char * [mGlobalMIDINodeName](#)
Name of the global MIDI node, if used.
- uint32_t [mGlobalMIDIEventMask](#)
Global MIDI node event mask of [AAX_EMidiGlobalNodeSelectors](#), if used.
- bool [mNeedsInputMIDI](#)
Does the instrument use a local MIDI input node?
- const char * [mInputMIDINodeName](#)
Name of the MIDI input node, if used.
- uint32_t [mInputMIDIChannelMask](#)
MIDI input node channel mask, if used.
- int32_t [mNumAdditionalInputMIDINodes](#)
Number of additional input MIDI Nodes. These will all share the same channelMask and base MIDINodeName, but the names will be appended with numbers 2,3,4,...
- bool [mNeedsTransport](#)
Does the instrument use the transport interface?
- const char * [mTransportMIDINodeName](#)
Name of the MIDI transport node, if used.
- int32_t [mNumMeters](#)

- const [AAX_CTypeID](#) * [mMeterIDs](#)
Number of meter taps used by the instrument. Must match the size of [mMeterIDs](#).
- int32_t [mNumAuxOutputStems](#)
Number of aux output stems for the plug-in.
- const char * [mAuxOutputStemNames](#) [[kMaxAuxOutputStems](#)]
Names of the aux output stems.
- [AAX_EStemFormat](#) [mAuxOutputStemFormats](#) [[kMaxAuxOutputStems](#)]
Stem formats for the output stems.
- [AAX_EStemFormat](#) [mHybridInputStemFormat](#)
Hybrid input stem format
- [AAX_EStemFormat](#) [mHybridOutputStemFormat](#)
Hybrid output stem format
- [AAX_EStemFormat](#) [mInputStemFormat](#)
Input stem format
- [AAX_EStemFormat](#) [mOutputStemFormat](#)
Output stem format
- bool [mUseHostGeneratedGUI](#)
Allow Pro Tools or other host to generate a generic GUI. This can be useful for early development.
- bool [mCanBypass](#)
Can this instrument be bypassed?
- [AAX_CTypeID](#) [mManufacturerID](#)
Manufacturer ID
- [AAX_CTypeID](#) [mProductID](#)
Product ID
- [AAX_CTypeID](#) [mPluginID](#)
Plug-In (Type) ID
- [AAX_CTypeID](#) [mAudiosuiteID](#)
AudioSuite ID
- [AAX_CBoolean](#) [mMultiMonoSupport](#)

14.117.2 Constructor & Destructor Documentation

14.117.2.1 [AAX_SInstrumentSetupInfo::AAX_SInstrumentSetupInfo\(\)](#) [inline]

Default constructor.

Use this constructor if you want to enable a sub-set of features and don't need to fill out the whole struct.

References [AAX_eStemFormat_Mono](#), [AAX_eStemFormat_None](#), and [kMaxAuxOutputStems](#).

14.117.3 Member Data Documentation

14.117.3.1 [bool AAX_SInstrumentSetupInfo::mNeedsGlobalMIDI](#)

Does the instrument use a global MIDI input node?

See also

[MIDI](#)

Referenced by [AAX_CMonolithicParameters::StaticDescribe\(\)](#).

14.117.3.2 const char* AAX_SInstrumentSetupInfo::mGlobalMIDIPropertyName

Name of the global MIDI node, if used.

See also

[MIDI](#)

Referenced by [AAX_CMonolithicParameters::StaticDescribe\(\)](#).

14.117.3.3 uint32_t AAX_SInstrumentSetupInfo::mGlobalMIDIEventMask

Global MIDI node event mask of [AAX_EMidiGlobalNodeSelectors](#), if used.

See also

[MIDI](#)

Referenced by [AAX_CMonolithicParameters::StaticDescribe\(\)](#).

14.117.3.4 bool AAX_SInstrumentSetupInfo::mNeedsInputMIDI

Does the instrument use a local MIDI input node?

See also

[MIDI](#)

Referenced by [AAX_CMonolithicParameters::StaticDescribe\(\)](#).

14.117.3.5 const char* AAX_SInstrumentSetupInfo::mInputMIDIPropertyName

Name of the MIDI input node, if used.

See also

[MIDI](#)

Referenced by [AAX_CMonolithicParameters::StaticDescribe\(\)](#).

14.117.3.6 uint32_t AAX_SInstrumentSetupInfo::mInputMIDIChannelMask

MIDI input node channel mask, if used.

See also

[MIDI](#)

Referenced by [AAX_CMonolithicParameters::StaticDescribe\(\)](#).

14.117.3.7 int32_t AAX_SInstrumentSetupInfo::mNumAdditionalInputMIDINodes

Number of additional input MIDI Nodes. These will all share the same channelMask and base MIDIPropertyName, but the names will be appended with numbers 2,3,4,...

See also

[MIDI](#)

Referenced by [AAX_CMonolithicParameters::StaticDescribe\(\)](#).

14.117.3.8 bool AAX_SInstrumentSetupInfo::mNeedsTransport

Does the instrument use the transport interface?

See also

[AAX_ITransport](#)

Referenced by [AAX_CMonolithicParameters::StaticDescribe\(\)](#).

14.117.3.9 const char* AAX_SInstrumentSetupInfo::mTransportMIDINodeName

Name of the MIDI transport node, if used.

See also

[AAX_ITransport](#)

14.117.3.10 int32_t AAX_SInstrumentSetupInfo::mNumMeters

Number of meter taps used by the instrument. Must match the size of [mMeterIDs](#).

See also

[Plug-in meters](#)

Referenced by [AAX_CMonolithicParameters::StaticDescribe\(\)](#).

14.117.3.11 const AAX_CTypeID* AAX_SInstrumentSetupInfo::mMeterIDs

Array of meter IDs.

See also

[Plug-in meters](#)

Referenced by [AAX_CMonolithicParameters::StaticDescribe\(\)](#).

14.117.3.12 int32_t AAX_SInstrumentSetupInfo::mNumAuxOutputStems

Number of aux output stems for the plug-in.

See also

[Auxiliary Output Stems](#)

Referenced by [AAX_CMonolithicParameters::StaticDescribe\(\)](#).

14.117.3.13 const char* AAX_SInstrumentSetupInfo::mAuxOutputStemNames[kMaxAuxOutputStems]

Names of the aux output stems.

See also

[Auxiliary Output Stems](#)

Referenced by [AAX_CMonolithicParameters::StaticDescribe\(\)](#).

14.117.3.14 AAX_EStemFormat AAX_SInstrumentSetupInfo::mAuxOutputStemFormats[kMaxAuxOutputStems]

Stem formats for the output stems.

See also

[Auxiliary Output Stems](#)

Referenced by [AAX_CMonolithicParameters::StaticDescribe\(\)](#).

14.117.3.15 AAX_EStemFormat AAX_SInstrumentSetupInfo::mHybridInputStemFormat

Hybrid input stem format

A plug-in that defines this value must also define `mHybridOutputStemFormat` and implement [AAX_IEffectParameters::RenderAudio_Hybrid\(\)](#)

See also

[Hybrid Processing architecture](#)

Referenced by [AAX_CMonolithicParameters::StaticDescribe\(\)](#).

14.117.3.16 AAX_EStemFormat AAX_SInstrumentSetupInfo::mHybridOutputStemFormat

Hybrid output stem format

A plug-in that defines this value must also define `mHybridInputStemFormat` and implement [AAX_IEffectParameters::RenderAudio_Hybrid\(\)](#)

See also

[Hybrid Processing architecture](#)

Referenced by [AAX_CMonolithicParameters::StaticDescribe\(\)](#).

14.117.3.17 AAX_EStemFormat AAX_SInstrumentSetupInfo::mInputStemFormat

Input stem format

Referenced by [AAX_CMonolithicParameters::StaticDescribe\(\)](#).

14.117.3.18 AAX_EStemFormat AAX_SInstrumentSetupInfo::mOutputStemFormat

Output stem format

Referenced by [AAX_CMonolithicParameters::StaticDescribe\(\)](#).

14.117.3.19 bool AAX_SInstrumentSetupInfo::mUseHostGeneratedGUI

Allow Pro Tools or other host to generate a generic GUI. This can be useful for early development.

See also

[AAX_eProperty_UsesClientGUI](#)

Referenced by [AAX_CMonolithicParameters::StaticDescribe\(\)](#).

14.117.3.20 bool AAX_SInstrumentSetupInfo::mCanBypass

Can this instrument be bypassed?

See also

[AAX_eProperty_CanBypass](#)

Referenced by [AAX_CMonolithicParameters::StaticDescribe\(\)](#).

14.117.3.21 AAX_CTypeID AAX_SInstrumentSetupInfo::mManufacturerID

[Manufacturer ID](#)

Referenced by [AAX_CMonolithicParameters::StaticDescribe\(\)](#).

14.117.3.22 AAX_CTypeID AAX_SInstrumentSetupInfo::mProductID

[Product ID](#)

Referenced by [AAX_CMonolithicParameters::StaticDescribe\(\)](#).

14.117.3.23 AAX_CTypeID AAX_SInstrumentSetupInfo::mPluginID

[Plug-In \(Type\) ID](#)

Referenced by [AAX_CMonolithicParameters::StaticDescribe\(\)](#).

14.117.3.24 AAX_CTypeID AAX_SInstrumentSetupInfo::mAudiosuiteID

[AudioSuite ID](#)

Referenced by [AAX_CMonolithicParameters::StaticDescribe\(\)](#).

14.117.3.25 AAX_CBoolean AAX_SInstrumentSetupInfo::mMultiMonoSupport

Multi-mono support

Note

It is recommended to un-set the `mMultiMonoSupport` flag for VIs and other plug-ins which rely on non-global MIDI input. For more information see [AAX_eProperty_Constraint_MultiMonoSupport](#)

Referenced by [AAX_CMonolithicParameters::StaticDescribe\(\)](#).

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

- [AAX_CMonolithicParameters.h](#)

14.118 AAX_SPlugInChunk Struct Reference

```
#include <AAX.h>
```

14.118.1 Description

Plug-in chunk header + data.

See also

[AAX_SPlugInChunkHeader](#)

Public Attributes

- **int32_t fSize**
The size of the chunk's [fData](#) member.
- **int32_t fVersion**
The chunk's version.
- **AAX_CTypeID fManufacturerID**
The Plug-In's manufacturer ID.
- **AAX_CTypeID fProductID**
The Plug-In file's product ID.
- **AAX_CTypeID fPlugInID**
The ID of a particular Plug-In within the file.
- **AAX_CTypeID fChunkID**
The ID of a particular Plug-In chunk.
- **unsigned char fName [32]**
A user defined name for this chunk.
- **char fData [1]**
The chunk's data.

14.118.2 Member Data Documentation

14.118.2.1 int32_t AAX_SPlugInChunk::fSize

The size of the chunk's [fData](#) member.

14.118.2.2 int32_t AAX_SPlugInChunk::fVersion

The chunk's version.

14.118.2.3 AAX_CTypeID AAX_SPlugInChunk::fManufacturerID

The Plug-In's manufacturer ID.

14.118.2.4 AAX_CTypeID AAX_SPlugInChunk::fProductID

The Plug-In file's product ID.

14.118.2.5 AAX_CTypeID AAX_SPlugInChunk::fPlugInID

The ID of a particular Plug-In within the file.

14.118.2.6 AAX_CTypeID AAX_SPlugInChunk::fChunkID

The ID of a particular Plug-In chunk.

14.118.2.7 unsigned char AAX_SPlugInChunk:: fName[32]

A user defined name for this chunk.

14.118.2.8 char AAX_SPlugInChunk::fData[1]

The chunk's data.

Note

The fixed-size array definition here is historical, but misleading. Plug-ins actually write off the end of this block and are allowed to do so long as they don't exceed their reported size.

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

- [AAX.h](#)

14.119 AAX_SPlugInChunkHeader Struct Reference

```
#include <AAX.h>
```

14.119.1 Description

Plug-in chunk header.

Legacy Porting Notes To ensure compatibility with TDM/RTAS plug-ins whose implementation requires `fSize` to be equal to the size of the chunk's header plus its data, AAE performs some behind-the-scenes record keeping.

The following actions are only taken for AAX plug-ins, so, e.g., if a chunk is stored by an RTAS or TDM plug-in that reports data+header size in `fSize` and this chunk is then loaded by the AAX version of the plug-in, the header size will be cached as-is from the legacy plug-in and will be subtracted out before the chunk data is passed to the AAX plug-in. If a chunk is stored by an AAX plug-in and is then loaded by a legacy plug-in, the legacy plug-in will receive the cached plug-in header with `fSize` equal to the data+header size.

These are the special actions that AAE takes to ensure backwards-compatibility when handling AAX chunk data:

- When AAE retrieves the size of a chunk from an AAX plug-in using [GetChunkSize\(\)](#), it adds the chunk header size to the amount of memory that it allocates for the chunk
- When AAE retrieves a chunk from an AAX plug-in using [GetChunk\(\)](#), it adds the chunk header size to `fChunkSize` before caching the chunk
- Before calling [SetChunk\(\)](#) or [CompareActiveChunk\(\)](#), AAE subtracts the chunk header size from the cached chunk's header's `fChunkSize` member

Public Attributes

- **int32_t fSize**
The size of the chunk's [fData](#) member.
- **int32_t fVersion**
The chunk's version.
- **AAX_CTypeID fManufacturerID**
The Plug-In's manufacturer ID.
- **AAX_CTypeID fProductID**
The Plug-In file's product ID.
- **AAX_CTypeID fPlugInID**
The ID of a particular Plug-In within the file.
- **AAX_CTypeID fChunkID**
The ID of a particular Plug-In chunk.
- **unsigned char fName [32]**
A user defined name for this chunk.

14.119.2 Member Data Documentation

14.119.2.1 int32_t AAX_SPlugInChunkHeader::fSize

The size of the chunk's [fData](#) member.

14.119.2.2 int32_t AAX_SPlugInChunkHeader::fVersion

The chunk's version.

14.119.2.3 AAX_CTypeID AAX_SPlugInChunkHeader::fManufacturerID

The Plug-In's manufacturer ID.

14.119.2.4 AAX_CTypeID AAX_SPlugInChunkHeader::fProductID

The Plug-In file's product ID.

14.119.2.5 AAX_CTypeID AAX_SPlugInChunkHeader::fPlugInID

The ID of a particular Plug-In within the file.

14.119.2.6 AAX_CTypeID AAX_SPlugInChunkHeader::fChunkID

The ID of a particular Plug-In chunk.

14.119.2.7 unsigned char AAX_SPlugInChunkHeader::fName[32]

A user defined name for this chunk.

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

- [AAX.h](#)

14.120 AAX_SPlugInIdentifierTriad Struct Reference

```
#include <AAX.h>
```

14.120.1 Description

Plug-in Identifier Triad.

This set of identifiers are what uniquely identify a particular plug-in type.

Public Attributes

- [AAX_CTypeID mManufacturerID](#)
The Plug-In's manufacturer ID.
- [AAX_CTypeID mProductID](#)
The Plug-In's product (Effect) ID.
- [AAX_CTypeID mPlugInID](#)
The ID of a specific type in the product (Effect)

14.120.2 Member Data Documentation

14.120.2.1 AAX_CTypeID AAX_SPlugInIdentifierTriad::mManufacturerID

The Plug-In's manufacturer ID.

Referenced by [AAX::AsStringIDTriad\(\)](#).

14.120.2.2 AAX_CTypeID AAX_SPlugInIdentifierTriad::mProductID

The Plug-In's product (Effect) ID.

Referenced by [AAX::AsStringIDTriad\(\)](#).

14.120.2.3 AAX_CTypeID AAX_SPlugInIdentifierTriad::mPlugInID

The ID of a specific type in the product (Effect)

Referenced by [AAX::AsStringIDTriad\(\)](#).

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

- [AAX.h](#)

14.121 AAX_StLock_Guard Class Reference

```
#include <AAX_CMutex.h>
```

14.121.1 Description

Helper class for working with mutex.

Public Member Functions

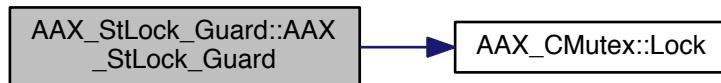
- [AAX_StLock_Guard \(AAX_CMutex &iMutex\)](#)
- [~AAX_StLock_Guard \(\)](#)

14.121.2 Constructor & Destructor Documentation

14.121.2.1 [AAX_StLock_Guard::AAX_StLock_Guard \(AAX_CMutex & iMutex \) \[inline\], \[explicit\]](#)

References [AAX_CMutex::Lock\(\)](#).

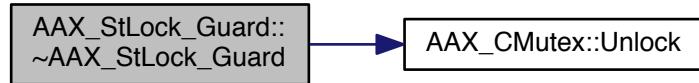
Here is the call graph for this function:



14.121.2.2 [AAX_StLock_Guard::~AAX_StLock_Guard \(\) \[inline\]](#)

References [AAX_CMutex::Unlock\(\)](#).

Here is the call graph for this function:



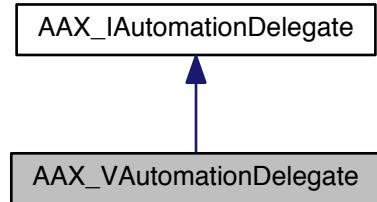
The documentation for this class was generated from the following file:

- [AAX_CMutex.h](#)

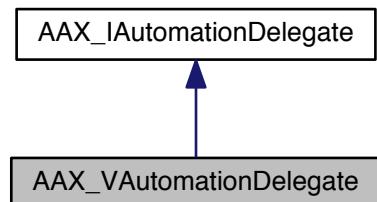
14.122 AAX_VAutomationDelegate Class Reference

```
#include <AAX_VAutomationDelegate.h>
```

Inheritance diagram for AAX_VAutomationDelegate:



Collaboration diagram for AAX_VAutomationDelegate:



14.122.1 Description

Version-managed concrete [automation delegate](#) class.

Public Member Functions

- [AAX_VAutomationDelegate \(IACFUnknown *pUnknown\)](#)
- virtual [~AAX_VAutomationDelegate \(\)](#)
- [IACFUnknown * GetUnknown \(\) const](#)
- virtual [AAX_Result RegisterParameter \(AAC_CParamID iParameterID\) AAX_OVERRIDE](#)
- virtual [AAX_Result UnregisterParameter \(AAC_CParamID iParameterID\) AAX_OVERRIDE](#)
- virtual [AAX_Result PostSetValueRequest \(AAC_CParamID iParameterID, double iNormalizedValue\) const AAX_OVERRIDE](#)
- virtual [AAX_Result PostCurrentValue \(AAC_CParamID iParameterID, double iNormalizedValue\) const AAX_OVERRIDE](#)
- virtual [AAX_Result PostTouchRequest \(AAC_CParamID iParameterID\) AAX_OVERRIDE](#)
- virtual [AAX_Result PostReleaseRequest \(AAC_CParamID iParameterID\) AAX_OVERRIDE](#)
- virtual [AAX_Result GetTouchState \(AAC_CParamID iParameterID, AAC_CBoolean *outTouched\) AAX_OVERRIDE](#)

14.122.2 Constructor & Destructor Documentation

14.122.2.1 **AAX_VAutomationDelegate::AAX_VAutomationDelegate (IACFUnknown * *pUnknown*)**

14.122.2.2 **virtual AAX_VAutomationDelegate::~AAX_VAutomationDelegate () [virtual]**

14.122.3 Member Function Documentation

14.122.3.1 **IACFUnknown* AAX_VAutomationDelegate::GetUnknown () const [inline]**

14.122.3.2 **virtual AAX_Result AAX_VAutomationDelegate::RegisterParameter (AAX_CParamID *iParameterID*) [virtual]**

Register a control with the automation system using a char* based control identifier

The automation delegate owns a list of the IDs of all of the parameters that have been registered with it. This list is used to set up listeners for all of the registered parameters such that the automation delegate may update the plug-in when the state of any of the registered parameters have been modified.

See also

[AAX_IAutomationDelegate::UnregisterParameter\(\)](#)

Parameters

in	<i>iParameterID</i>	Parameter ID that is being registered
----	---------------------	---------------------------------------

Implements [AAX_IAutomationDelegate](#).

14.122.3.3 **virtual AAX_Result AAX_VAutomationDelegate::UnregisterParameter (AAX_CParamID *iParameterID*) [virtual]**

Unregister a control with the automation system using a char* based control identifier

Note

All registered controls should be unregistered or the system might leak.

See also

[AAX_IAutomationDelegate::RegisterParameter\(\)](#)

Parameters

in	<i>iParameterID</i>	Parameter ID that is being registered
----	---------------------	---------------------------------------

Implements [AAX_IAutomationDelegate](#).

14.122.3.4 **virtual AAX_Result AAX_VAutomationDelegate::PostSetValueRequest (AAX_CParamID *iParameterID*, double *normalizedValue*) const [virtual]**

Submits a request for the given parameter's value to be changed

Parameters

in	<i>iParameterID</i>	ID of the parameter for which a change is requested
in	<i>normalizedValue</i>	The requested new parameter value, formatted as a double and normalized to [0 1]

Implements [AAX_IAutomationDelegate](#).

14.122.3.5 virtual AAX_Result AAX_VAutomationDelegate::PostCurrentValue (AAX_CParamID *iParameterID*, double *normalizedValue*) const [virtual]

Notifies listeners that a parameter's value has changed

Parameters

in	<i>iParameterID</i>	ID of the parameter that has been updated
in	<i>normalizedValue</i>	The current parameter value, formatted as a double and normalized to [0 1]

Implements [AAX_IAutomationDelegate](#).

14.122.3.6 virtual AAX_Result AAX_VAutomationDelegate::PostTouchRequest (AAX_CParamID *iParameterID*) [virtual]

Requests that the given parameter be "touched", i.e. locked for updates by the current client

Parameters

in	<i>iParameterID</i>	ID of the parameter that will be touched
----	---------------------	--

Implements [AAX_IAutomationDelegate](#).

14.122.3.7 virtual AAX_Result AAX_VAutomationDelegate::PostReleaseRequest (AAX_CParamID *iParameterID*) [virtual]

Requests that the given parameter be "released", i.e. available for updates from any client

Parameters

in	<i>iParameterID</i>	ID of the parameter that will be released
----	---------------------	---

Implements [AAX_IAutomationDelegate](#).

14.122.3.8 virtual AAX_Result AAX_VAutomationDelegate::GetTouchState (AAX_CParamID *iParameterID*, AAX_CBoolean * *oTouched*) [virtual]

Gets the current touched state of a parameter

Parameters

in	<i>iParameterID</i>	ID of the parameter that is being queried
out	<i>oTouched</i>	The current touch state of the parameter

Implements [AAX_IAutomationDelegate](#).

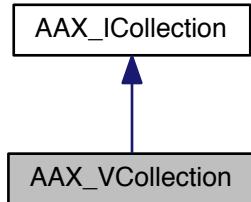
The documentation for this class was generated from the following file:

- [AAX_VAutomationDelegate.h](#)

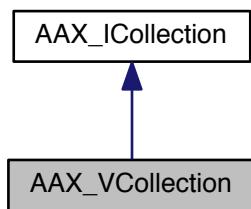
14.123 AAX_VCollection Class Reference

```
#include <AAX_VCollection.h>
```

Inheritance diagram for AAX_VCollection:



Collaboration diagram for AAX_VCollection:



14.123.1 Description

Version-managed concrete [AAX_ICollection](#) class.

Public Member Functions

- [AAX_VCollection \(IACFUnknown *pUnkHost\)](#)
- virtual [~AAX_VCollection \(\)](#)
- virtual [AAX_IEffectDescriptor * NewDescriptor \(\) AAX_OVERRIDE](#)
Create a new Effect descriptor.
- virtual [AAX_Result AddEffect \(const char *inEffectID, AAX_IEffectDescriptor *inEffectDescriptor\) AAX_OVERRIDE](#)
Add an Effect description to the collection.
- virtual [AAX_Result SetManufacturerName \(const char *inPackageName\) AAX_OVERRIDE](#)
Set the plug-in manufacturer name.
- virtual [AAX_Result AddPackageName \(const char *inPackageName\) AAX_OVERRIDE](#)
Set the plug-in package name.
- virtual [AAX_Result SetPackageVersion \(uint32_t inVersion\) AAX_OVERRIDE](#)
Set the plug-in package version number.
- virtual [AAX_IPropertyMap * NewPropertyMap \(\) AAX_OVERRIDE](#)

Create a new property map.

- virtual [AAx_Result SetProperties \(AAx_IPropertyMap *inProperties\) AAX_OVERRIDE](#)
Set the properties of the collection.
- virtual [AAx_IDescriptionHost * DescriptionHost \(\) AAX_OVERRIDE](#)
- virtual const [AAx_IDescriptionHost * DescriptionHost \(\) const AAX_OVERRIDE](#)
- virtual [IACFDefinition * HostDefinition \(\) const AAX_OVERRIDE](#)
- IACFPluginDefinition * [GetIUnknown \(void\) const](#)

14.123.2 Constructor & Destructor Documentation

14.123.2.1 [AAx_VCollection::AAx_VCollection \(IACFUnknown * pUnkHost \)](#)

14.123.2.2 virtual [AAx_VCollection::~AAx_VCollection \(\) \[virtual\]](#)

14.123.3 Member Function Documentation

14.123.3.1 virtual [AAx_IEffectDescriptor* AAX_VCollection::NewDescriptor \(\) \[virtual\]](#)

Create a new Effect descriptor.

This implementation retains each generated [AAx_IEffectDescriptor](#) and destroys the descriptor upon [AAx_VCollection](#) destructionCreate a new Effect descriptor.

Implements [AAx_ICollection](#).

14.123.3.2 virtual [AAx_Result AAX_VCollection::AddEffect \(const char * inEffectID, AAX_IEffectDescriptor * inEffectDescriptor \) \[virtual\]](#)

Add an Effect description to the collection.

Each Effect that a plug-in registers with [AAx_ICollection::AddEffect\(\)](#) is considered a completely different user-facing product. For example, in Avid's Dynamics III plug-in the Expander, Compressor, and DeEsser are each registered as separate Effects. All stem format variations within each Effect are registered within that Effect's [AAx_IEffectDescriptor](#) using [AddComponent\(\)](#).

The [AAx_eProperty_ProductID](#) value for all ProcessProcs within a single Effect must be identical.

This method passes ownership of an [AAx_IEffectDescriptor](#) object to the [AAx_ICollection](#). The [AAx_IEffectDescriptor](#) must not be deleted by the AAX plug-in, nor should it be edited in any way after it is passed to the [AAx_ICollection](#).

Parameters

in	<i>inEffectID</i>	The effect ID.
in	<i>inEffectDescriptor</i>	The Effect descriptor.

Implements [AAx_ICollection](#).

14.123.3.3 virtual [AAx_Result AAX_VCollection::SetManufacturerName \(const char * inPackageName \) \[virtual\]](#)

Set the plug-in manufacturer name.

Parameters

in	<i>inPackageName</i>	The name of the manufacturer.
----	----------------------	-------------------------------

Implements [AAX_ICollection](#).

14.123.3.4 virtual AAX_Result AAX_VCollection::AddPackageName (const char * *inPackageName*) [virtual]

Set the plug-in package name.

May be called multiple times to add abbreviated package names.

Note

Every plug-in must include at least one name variant with 16 or fewer characters, plus a null terminating character. Used for Presets folder.

Parameters

in	<i>inPackageName</i>	The name of the package.
----	----------------------	--------------------------

Implements [AAX_ICollection](#).

14.123.3.5 virtual AAX_Result AAX_VCollection::SetPackageVersion (uint32_t *inVersion*) [virtual]

Set the plug-in package version number.

Parameters

in	<i>inVersion</i>	The package version numner.
----	------------------	-----------------------------

Implements [AAX_ICollection](#).

14.123.3.6 virtual AAX_IPropertyMap* AAX_VCollection::NewPropertyMap () [virtual]

Create a new property map.

Implements [AAX_ICollection](#).

14.123.3.7 virtual AAX_Result AAX_VCollection::SetProperties (AAX_IPropertyMap * *inProperties*) [virtual]

Set the properties of the collection.

Parameters

in	<i>inProperties</i>	Collection properties
----	---------------------	-----------------------

Implements [AAX_ICollection](#).

14.123.3.8 virtual AAX_IDescriptionHost* AAX_VCollection::DescriptionHost () [virtual]

Get a pointer to an [AAX_IDescriptionHost](#), if supported by the host

This interface is served by the [AAX_ICollection](#) in order to avoid requiring a new method prototype for the [GetEffectDescriptions\(\)](#) method called from the AAX Library.

See also

[AAX_UIDs.h](#) for available feature UIDs, e.g. [AAXATTR_ClientFeature_AuxOutputStem](#)

Implements [AAX_ICollection](#).

14.123.3.9 virtual const AAX_IDescriptionHost* AAX_VCollection::DescriptionHost() const [virtual]

Get a pointer to an [AAX_IDescriptionHost](#), if supported by the host

This interface is served by the [AAX_ICollection](#) in order to avoid requiring a new method prototype for the [GetEffectDescriptions\(\)](#) method called from the AAX Library.

See also

[AAX_UIDs.h](#) for available feature UIDs, e.g. [AAXATTR_ClientFeature_AuxOutputStem](#)

Implements [AAX_ICollection](#).

14.123.3.10 virtual IACFDefinition* AAX_VCollection::HostDefinition() const [virtual]

Get a pointer to an [IACFDefinition](#), if supported by the host

This interface is served by the [AAX_ICollection](#) in order to avoid requiring a new method prototype for the [GetEffectDescriptions\(\)](#) method called from the AAX Library.

See also

[AAX_UIDs.h](#) for available host attribute UIDs, e.g. [AAXATTR_Client_Level](#)

The implementation of [AAX_ICollection](#) owns the referenced object. No AddRef occurs.

[IACFDefinition::DefineAttribute\(\)](#) is not supported on this object

Implements [AAX_ICollection](#).

14.123.3.11 IACFPluginDefinition* AAX_VCollection::GetUnknown(void) const

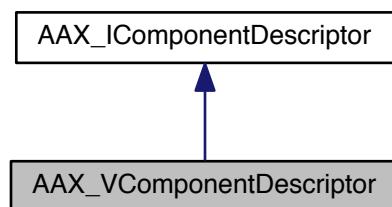
The documentation for this class was generated from the following file:

- [AAX_VCollection.h](#)

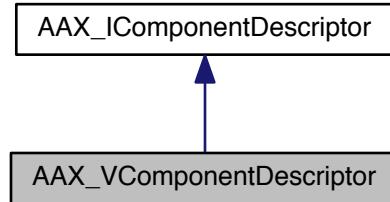
14.124 AAX_VComponentDescriptor Class Reference

```
#include <AAX_VComponentDescriptor.h>
```

Inheritance diagram for AAX_VComponentDescriptor:



Collaboration diagram for AAX_VComponentDescriptor:



14.124.1 Description

Version-managed concrete [AAX_IComponentDescriptor](#) class.

Public Member Functions

- [AAX_VComponentDescriptor \(IACFUnknown *pUnkHost\)](#)
- virtual [~AAX_VComponentDescriptor \(\)](#)
- virtual [AAAX_Result Clear \(\) AAAX_OVERRIDE](#)
Clears the descriptor.
- virtual [AAAX_Result AddReservedField \(AAAX_CFieldIndex inFieldIndex, uint32_t inFieldType\) AAAX_OVERRIDE](#)
Subscribes a context field to host-provided services or information.
- virtual [AAAX_Result AddAudioIn \(AAAX_CFieldIndex inFieldIndex\) AAAX_OVERRIDE](#)
Subscribes an audio input context field.
- virtual [AAAX_Result AddAudioOut \(AAAX_CFieldIndex inFieldIndex\) AAAX_OVERRIDE](#)
Subscribes an audio output context field.
- virtual [AAAX_Result AddAudioBufferLength \(AAAX_CFieldIndex inFieldIndex\) AAAX_OVERRIDE](#)
Subscribes a buffer length context field.
- virtual [AAAX_Result AddSampleRate \(AAAX_CFieldIndex inFieldIndex\) AAAX_OVERRIDE](#)
Subscribes a sample rate context field.
- virtual [AAAX_Result AddClock \(AAAX_CFieldIndex inFieldIndex\) AAAX_OVERRIDE](#)
Subscribes a clock context field.
- virtual [AAAX_Result AddSideChainIn \(AAAX_CFieldIndex inFieldIndex\) AAAX_OVERRIDE](#)
Subscribes a side-chain input context field.
- virtual [AAAX_Result AddDataInPort \(AAAX_CFieldIndex inFieldIndex, uint32_t inPacketSize, AAAX_EDataInPortType inPortType\) AAAX_OVERRIDE](#)
Adds a custom data port to the algorithm context.
- virtual [AAAX_Result AddAuxOutputStem \(AAAX_CFieldIndex inFieldIndex, int32_t inStemFormat, const char inNameUTF8\[\]\) AAAX_OVERRIDE](#)
Adds an auxiliary output stem for a plug-in.
- virtual [AAAX_Result AddPrivateData \(AAAX_CFieldIndex inFieldIndex, int32_t inDataSize, uint32_t inOptions\) AAAX_OVERRIDE](#)
Adds a private data port to the algorithm context.
- virtual [AAAX_Result AddTemporaryData \(AAAX_CFieldIndex inFieldIndex, uint32_t inDataElementSize\) AAAX_OVERRIDE](#)

- Adds a block of data to a context that is not saved between callbacks and is scaled by the system buffer size.
- virtual [AAx_Result AddDmaInstance \(AAx_CFieldIndex inFieldIndex, AAX_IDma::EMode inDmaMode\) AAX_OVERRIDE](#)
 - Adds a DMA field to the plug-in's context.*
- virtual [AAx_Result AddMeters \(AAx_CFieldIndex inFieldIndex, const AAX_CTypeID *inMeterIDs, const uint32_t inMeterCount\) AAX_OVERRIDE](#)
 - Adds a meter field to the plug-in's context.*
- virtual [AAx_Result AddMIDINode \(AAx_CFieldIndex inFieldIndex, AAX_EMIDINodeType inNodeType, const char inNodeName\[\], uint32_t channelMask\) AAX_OVERRIDE](#)
 - Adds a MIDI node field to the plug-in's context.*
- virtual [AAx_IPropertyMap * NewPropertyMap \(\) const AAX_OVERRIDE](#)
 - Creates a new, empty property map.*
- virtual [AAx_IPropertyMap * DuplicatePropertyMap \(AAx_IPropertyMap *inPropertyMap\) const AAX_OVERRIDE](#)
 - Creates a new property map using an existing property map.*
- virtual [AAx_Result AddProcessProc_Native \(AAx_CProcessProc inProcessProc, AAX_IPropertyMap *inProperties=NULL, AAX_CInstanceInitProc inInstanceInitProc=NULL, AAX_CBackgroundProc inBackgroundProc=NULL, AAX_CSelector *outProcID=NULL\) AAX_OVERRIDE](#)
 - Registers an algorithm processing entrypoint (process procedure) for the native architecture.*
- virtual [AAx_Result AddProcessProc_TI \(const char inDLLFileNameUTF8\[\], const char inProcessProcSymbol\[\], AAX_IPropertyMap *inProperties=NULL, const char inInstanceInitProcSymbol\[\]=NULL, const char inBackgroundProcSymbol\[\]=NULL, AAX_CSelector *outProcID=NULL\) AAX_OVERRIDE](#)
 - Registers an algorithm processing entrypoint (process procedure) for the native architecture.*
- virtual [AAx_Result AddProcessProc \(AAx_IPropertyMap *inProperties, AAX_CSelector *outProcIDs=NULL, int32_t inProcIDsSize=0\) AAX_OVERRIDE](#)
 - Registers one or more algorithm processing entrypoints (process procedures)*
- [IACFUnknown * GetIUnknown \(void\) const](#)

Friends

- class [AAx_VPropertyMap](#)

14.124.2 Constructor & Destructor Documentation

14.124.2.1 [AAx_VComponentDescriptor::AAx_VComponentDescriptor \(IACFUnknown * pUnkHost \)](#)

14.124.2.2 [virtual AAx_VComponentDescriptor::~AAx_VComponentDescriptor \(\) \[virtual\]](#)

14.124.3 Member Function Documentation

14.124.3.1 [virtual AAX_Result AAx_VComponentDescriptor::Clear \(\) \[virtual\]](#)

Clears the descriptor.

Clears the descriptor and readies it for the next algorithm description

Implements [AAx_IComponentDescriptor](#).

14.124.3.2 [virtual AAX_Result AAx_VComponentDescriptor::AddReservedField \(AAX_CFieldIndex inFieldIndex, uint32_t inFieldType \) \[virtual\]](#)

Subscribes a context field to host-provided services or information.

Note

Currently for internal use only.

Parameters

in	<i>inFieldIndex</i>	Unique identifier for the field, generated using AAX_FIELD_INDEX
in	<i>inFieldType</i>	Type of field that is being added

Implements [AAX_IComponentDescriptor](#).

14.124.3.3 virtual AAX_Result AAX_VComponentDescriptor::AddAudioIn (AAX_CFieldIndex *inFieldIndex*) [virtual]

Subscribes an audio input context field.

Defines an audio in port for host-provided information in the algorithm's context structure.

- Data type: float**
- Data kind: An array of float arrays, one for each input channel

Parameters

in	<i>inFieldIndex</i>	Unique identifier for the field, generated using AAX_FIELD_INDEX
----	---------------------	--

Implements [AAX_IComponentDescriptor](#).

14.124.3.4 virtual AAX_Result AAX_VComponentDescriptor::AddAudioOut (AAX_CFieldIndex *inFieldIndex*) [virtual]

Subscribes an audio output context field.

Defines an audio out port for host-provided information in the algorithm's context structure.

- Data type: float**
- Data kind: An array of float arrays, one for each output channel

Parameters

in	<i>inFieldIndex</i>	Unique identifier for the field, generated using AAX_FIELD_INDEX
----	---------------------	--

Implements [AAX_IComponentDescriptor](#).

14.124.3.5 virtual AAX_Result AAX_VComponentDescriptor::AddAudioBufferLength (AAX_CFieldIndex *inFieldIndex*) [virtual]

Subscribes a buffer length context field.

Defines a buffer length port for host-provided information in the algorithm's context structure.

- Data type: int32_t*
- Data kind: The number of samples in the current audio buffer

Parameters

in	<i>inFieldIndex</i>	Unique identifier for the field, generated using AAX_FIELD_INDEX
----	---------------------	--

Implements [AAX_IComponentDescriptor](#).

14.124.3.6 virtual AAX_Result AAX_VComponentDescriptor::AddSampleRate (AAX_CFieldIndex *inFieldIndex*) [virtual]

Subscribes a sample rate context field.

Defines a sample rate port for host-provided information in the algorithm's context structure.

- Data type: [AAX_CSampleRate](#) *
- Data kind: The current sample rate

Parameters

in	<i>inFieldIndex</i>	Unique identifier for the field, generated using AAX_FIELD_INDEX
----	---------------------	--

Implements [AAX_IComponentDescriptor](#).

14.124.3.7 virtual AAX_Result AAX_VComponentDescriptor::AddClock (AAX_CFieldIndex *inFieldIndex*) [virtual]

Subscribes a clock context field.

Defines a clock port for host-provided information in the algorithm's context structure.

- Data type: [AAX_CTimestamp](#) *
- Data kind: A running counter which increments even when the transport is not playing. The counter increments exactly once per sample quantum.

Host Compatibility Notes As of Pro Tools 11.1, this field may be used in both Native and DSP plug-ins. The DSP clock data is a 16-bit cycling counter. This field was only available for Native plug-ins in previous Pro Tools versions.

Parameters

in	<i>inFieldIndex</i>	Unique identifier for the field, generated using AAX_FIELD_INDEX
----	---------------------	--

Implements [AAX_IComponentDescriptor](#).

14.124.3.8 virtual AAX_Result AAX_VComponentDescriptor::AddSideChainIn (AAX_CFieldIndex *inFieldIndex*) [virtual]

Subscribes a side-chain input context field.

Defines a side-chain input port for host-provided information in the algorithm's context structure.

- Data type: int32_t*
- Data kind: The index of the plug-in's first side-chain input channel within the array of input audio buffers

Parameters

in	<i>inFieldIndex</i>	Unique identifier for the field, generated using AAX_FIELD_INDEX
----	---------------------	--

Implements [AAX_IComponentDescriptor](#).

14.124.3.9 virtual AAX_Result AAX_VComponentDescriptor::AddDataInPort (AAX_CFieldIndex *inFieldIndex*, uint32_t *inPacketSize*, AAX_EDataInPortType *inPortType*) [virtual]

Adds a custom data port to the algorithm context.

Defines a read-only data port for plug-in information in the algorithm's context structure. The plug-in can send information to this port using [AAX_IController::PostPacket\(\)](#).

The host guarantees that all packets will be delivered to this port in the order in which they were posted, up to the point of a packet buffer overflow, though some packets may be dropped depending on the *inPortType* and host implementation.

Note

When a plug-in is operating in offline (AudioSuite) mode, all data ports operate as [AAX_eDataInPortType_↔Unbuffered](#) ports

Parameters

in	<i>inFieldIndex</i>	Unique identifier for the port, generated using AAX_FIELD_INDEX
in	<i>inPacketSize</i>	Size of the data packets that will be sent to this port
in	<i>inPortType</i>	The requested packet delivery behavior for this port

Implements [AAX_IComponentDescriptor](#).

14.124.3.10 virtual AAX_Result AAX_VComponentDescriptor::AddAuxOutputStem (AAX_CFieldIndex *inFieldIndex*, int32_t *inStemFormat*, const char *inNameUTF8[]*) [virtual]

Adds an auxiliary output stem for a plug-in.

Use this method to add additional output channels to the algorithm context.

The aux output stem audio buffers will be added to the end of the audio outputs array in the order in which they are described. When writing audio data to a specific aux output, find the proper starting channel by accumulating all of the channels of the main output stem format and any previously-described aux output stems.

The plug-in is responsible for providing a meaningful name for each aux outputs. At the very least, individual outputs should be labeled "Output xx", where "xx" is the aux output number as it is defined in the plug-in. The output name should also include the words "mono" and "stereo" to support when users are looking for an output with a specific stem format.

Host Compatibility Notes There is a hard limit to the number of outputs that Pro Tools supports for a single plug-in instance. This limit is currently set at 256 channels, which includes all of the plug-in's output channels in addition to the sum total of all of its aux output stem channels.

Host Compatibility Notes Pro Tools supports only mono and stereo auxiliary output stem formats

Warning

This method will return an error code on hosts which do not support auxiliary output stems. This indicates that the host will not provide audio buffers for auxiliary output stems during processing. A plug-in must not attempt to write data into auxiliary output stem buffers which have not been provided by the host!

Parameters

in	<i>inFieldIndex</i>	DEPRECATED: This parameter is no longer needed by the host, but is included in the interface for binary compatibility
in	<i>inStemFormat</i>	The stem format of the new aux output
in	<i>inNameUTF8</i>	The name of the aux output. This name is static and cannot be changed after the descriptor is submitted to the host

Implements [AAX_IComponentDescriptor](#).

14.124.3.11 virtual AAX_Result AAX_VComponentDescriptor::AddPrivateData (AAX_CFieldIndex *inFieldIndex*, int32_t *inDataSize*, uint32_t *inOptions*) [virtual]

Adds a private data port to the algorithm context.

Defines a read/write data port for private state data. Data written to this port will be maintained by the host between calls to the algorithm context.

See also

[alg_pd_registration](#)

Parameters

in	<i>inFieldIndex</i>	Unique identifier for the port, generated using AAX_FIELD_INDEX
in	<i>inDataSize</i>	Size of the data packets that will be sent to this port
in	<i>inOptions</i>	Options that define the private data port's behavior

Implements [AAX_IComponentDescriptor](#).

14.124.3.12 virtual AAX_Result AAX_VComponentDescriptor::AddTemporaryData (AAX_CFieldIndex *inFieldIndex*, uint32_t *inDataElementSize*) [virtual]

Adds a block of data to a context that is not saved between callbacks and is scaled by the system buffer size.

This can be very useful if you use block processing and need to store intermediate results. Just specify your base element size and the system will scale the overall block size by the buffer size. For example, to create a buffer of floats that is the length of the block, specify 4 bytes as the elementsize.

This data block does not retain state across callback and can also be reused across instances on memory contrained systems.

Parameters

in	<i>inFieldIndex</i>	Unique identifier for the port, generated using AAX_FIELD_INDEX
in	<i>inDataElementSize</i>	The size of a single piece of data in the block. This number will be multiplied by the processing block size to determine total block size.

Implements [AAX_IComponentDescriptor](#).

14.124.3.13 virtual AAX_Result AAX_VComponentDescriptor::AddDmaInstance (AAX_CFieldIndex *inFieldIndex*, AAX_IDma::EMode *inDmaMode*) [virtual]

Adds a DMA field to the plug-in's context.

DMA (direct memory access) provides efficient reads from and writes to external memory on the DSP. DMA behavior is emulated in host-based plug-ins for cross-platform portability.

Note

The order in which DMA instances are added defines their priority and therefore order of execution of DMA operations. In most plug-ins, Scatter fields should be placed first in order to achieve the lowest possible access latency.

For more information, see [Direct Memory Access](#).

Todo Update the DMA system management such that operation priority can be set arbitrarily

Parameters

in	<i>inFieldIndex</i>	Unique identifier for the field, generated using AAX_FIELD_INDEX
in	<i>inDmaMode</i>	AAX_IDma::EMode that will apply to this field

Implements [AAX_IComponentDescriptor](#).

14.124.3.14 virtual [AAX_Result](#) [AAX_VComponentDescriptor::AddMeters](#) ([AAX_CFieldIndex](#) *inFieldIndex*, const [AAX_CTypeID](#) * *inMeterIDs*, const [uint32_t](#) *inMeterCount*) [virtual]

Adds a meter field to the plug-in's context.

Meter fields include an array of meter tap values, with one tap per meter per context. Only one meter field should be added per Component. Individual meter behaviors can be described at the Effect level.

For more information, see [Plug-in meters](#).

Parameters

in	<i>inFieldIndex</i>	Unique identifier for the field, generated using AAX_FIELD_INDEX
in	<i>inMeterIDs</i>	Array of 32-bit IDs, one for each meter. Meter IDs must be unique within the Effect.
in	<i>inMeterCount</i>	The number of meters included in this field

Implements [AAX_IComponentDescriptor](#).

14.124.3.15 virtual [AAX_Result](#) [AAX_VComponentDescriptor::AddMIDIINode](#) ([AAX_CFieldIndex](#) *inFieldIndex*, [AAX_EMIDIINodeType](#) *inNodeType*, const char *innodeName*[], [uint32_t](#) *channelMask*) [virtual]

Adds a MIDI node field to the plug-in's context.

- Data type: [AAX_IMIDIINode](#) *

The resulting MIDI node data will be available both in the algorithm context and in the plug-in's [data model](#) via [UpdateMIDIINodes\(\)](#).

To add a MIDI node that is only accessible to the plug-in's data model, use [AAX_IEffectDescriptor::AddControlMIDIINode\(\)](#)

Host Compatibility Notes Due to current restrictions MIDI data won't be delivered to DSP algorithms, only to AAX Native.

Parameters

in	<i>inFieldIndex</i>	The ID of the port. MIDI node ports should be formatted as a pointer to an AA_X_IMIDIINode .
----	---------------------	--

in	<i>inNodeType</i>	The type of MIDI node, as AAX_EMIDINodeType
in	<i>innodeName</i>	The name of the MIDI node as it should appear in the host's UI
in	<i>channelMask</i>	The channel mask for the MIDI node. This parameter specifies used MIDI channels. For Global MIDI nodes, use a mask of AAX_EMidiGlobalNodeSelectors

Implements [AAX_IComponentDescriptor](#).

14.124.3.16 virtual [AAX_IPropertyMap](#)* [AAX_VComponentDescriptor](#)::NewPropertyMap() const [virtual]

Creates a new, empty property map.

The component descriptor owns the reference to the resulting property map, and the underlying property map is destroyed when the component descriptor is released.

This implementation retains each generated [AAX_IPropertyMap](#) and destroys the property map upon [AAX_VComponentDescriptor](#) destruction

Implements [AAX_IComponentDescriptor](#).

14.124.3.17 virtual [AAX_IPropertyMap](#)* [AAX_VComponentDescriptor](#)::DuplicatePropertyMap([AAX_IPropertyMap](#) * *inPropertyMap*) const [virtual]

Creates a new property map using an existing property map.

The component descriptor owns the reference to the resulting property map, and the underlying property map is destroyed when the component descriptor is released.

Parameters

in	<i>inPropertyMap</i>	The property values in this map will be copied into the new map
----	----------------------	---

This implementation retains each generated [AAX_IPropertyMap](#) and destroys the property map upon [AAX_VComponentDescriptor](#) destruction

Implements [AAX_IComponentDescriptor](#).

14.124.3.18 virtual [AAX_Result](#) [AAX_VComponentDescriptor](#)::AddProcessProc_Native([AAX_CProcessProc](#) *inProcessProc*, [AAX_IPropertyMap](#) * *inProperties* = NULL, [AAX_CInstanceInitProc](#) *inInstanceInitProc* = NULL, [AAX_CBackgroundProc](#) *inBackgroundProc* = NULL, [AAX_CSelector](#) * *outProcID* = NULL) [virtual]

Registers an algorithm processing entrypoint (process procedure) for the native architecture.

Parameters

in	<i>inProcessProc</i>	Symbol for this processing callback
in	<i>inProperties</i>	A property map for this processing callback. The property map's values are copied by the host and associated with the new ProcessProc. The property map contents are unchanged and the map may be re-used when registering additional ProcessProcs.
in	<i>inInstanceInitProc</i>	Initialization routine that will be called when a new instance of the Effect is created. See Algorithm initialization .
in	<i>inBackgroundProc</i>	Background routine that will be called in an idle context within the same address space as the associated process procedure. See Background processing callback

out	outProcID	
-----	-----------	--

Todo document this parameter

Implements [AAX_IComponentDescriptor](#).

```
14.124.3.19 virtual AAX_Result AAX_VComponentDescriptor::AddProcessProc_TI ( const char inDLLFileNameUTF8[],  
                           const char inProcessProcSymbol[] , AAX_IPropertyMap * inProperties = NULL, const char  
                           inInstanceInitProcSymbol[] = NULL, const char inBackgroundProcSymbol[] = NULL, AAX_CSelector *  
                           outProcID = NULL ) [virtual]
```

Registers an algorithm processing entrypoint (process procedure) for the native architecture.

Parameters

in	<i>inDLLFileNameUTF8</i>	UTF-8 encoded filename for the ELF DLL containing the algorithm code fragment
in	<i>inProcessProcSymbol</i>	Symbol for this processing callback
in	<i>inProperties</i>	A property map for this processing callback. The property map's values are copied by the host and associated with the new ProcessProc. The property map contents are unchanged and the map may be re-used when registering additional ProcessProcs.
in	<i>inInstanceInitProcSymbol</i>	Initialization routine that will be called when a new instance of the Effect is created. Must be included in the same DLL as the main algorithm entrypoint. See Algorithm initialization .
in	<i>inBackgroundProcSymbol</i>	Background routine that will be called in an idle context within the same address space as the associated process procedure. Must be included in the same DLL as the main algorithm entrypoint. See Background processing callback
out	<i>outProcID</i>	

Todo document this parameter

Implements [AAX_IComponentDescriptor](#).

```
14.124.3.20 virtual AAX_Result AAX_VComponentDescriptor::AddProcessProc ( AAX_IPropertyMap * inProperties,  
                           AAX_CSelector * outProcIDs = NULL, int32_t inProcIDsSize = 0 ) [virtual]
```

Registers one or more algorithm processing entrypoints (process procedures)

Any non-overlapping set of processing entrypoints may be specified. Typically this can be used to specify both Native and TI entrypoints using the same call.

The AAX Library implementation of this method includes backwards compatibility logic to complete the ProcessProc registration on hosts which do not support this method. Therefore plug-in code may use this single registration routine instead of separate calls to [AddProcessProc_Native\(\)](#), [AddProcessProc_TI\(\)](#), etc. regardless of the host version.

The following properties replace the input arguments to the platform-specific registration methods:

[AddProcessProc_Native\(\)](#) ([AAX_eProperty_PlugInID_Native](#), [AAX_eProperty_PlugInID_AudioSuite](#))

- [AAX_CProcessProc](#) *iProcessProc*: [AAX_eProperty_NativeProcessProc](#) (required)
- [AAX_CInstanceInitProc](#) *iInstanceInitProc*: [AAX_eProperty_NativeInstanceInitProc](#) (optional)
- [AAX_CBackgroundProc](#) *iBackgroundProc*: [AAX_eProperty_NativeBackgroundProc](#) (optional)

[AddProcessProc_TI\(\)](#) ([AAX_eProperty_PlugInID_TI](#))

- const char inDLLFileNameUTF8[]: [AAX_eProperty_TIDLLFileName](#) (required)
- const char iProcessProcSymbol[]: [AAX_eProperty_TIProcessProc](#) (required)
- const char iInstanceInitProcSymbol[]: [AAX_eProperty_TIInstanceInitProc](#) (optional)
- const char iBackgroundProcSymbol[]: [AAX_eProperty_TIBackgroundProc](#) (optional)

If any platform-specific plug-in ID property is present in `iProperties` then [AddProcessProc\(\)](#) will check for the required properties for that platform.

Note

[AAX_eProperty_AudioBufferLength](#) will be ignored for the Native and AudioSuite ProcessProcs since it should only be used for AAX DSP.

Parameters

in	<i>inProperties</i>	A property map for this processing callback. The property map's values are copied by the host and associated with the new ProcessProc. The property map contents are unchanged and the map may be re-used when registering additional ProcessProcs.
out	<i>outProcIDs</i>	

Todo document this parameter Returned array will be NULL-terminated

Parameters

in	<i>inProcIDsSize</i>	The size of the array provided to <code>oProcIDs</code> . If <code>oProcIDs</code> is non-NULL but <code>iProcIDsSize</code> is not large enough for all of the registered ProcessProcs (plus one for NULL termination) then this method will fail with AAX_ERROR_ARGUMENT_OVERFLOW
----	----------------------	---

Implements [AAX_IComponentDescriptor](#).

14.124.3.21 IACFUnknown* AAX_VComponentDescriptor::GetUnknown (void) const

14.124.4 Friends And Related Function Documentation

14.124.4.1 friend class AAX_VPropertyMap [friend]

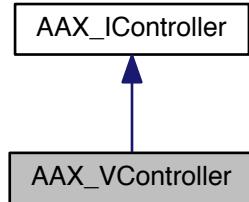
The documentation for this class was generated from the following file:

- [AAX_VComponentDescriptor.h](#)

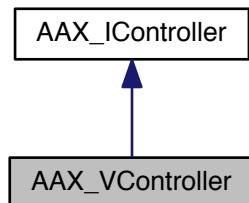
14.125 AAX_VController Class Reference

```
#include <AAX_VController.h>
```

Inheritance diagram for AAX_VController:



Collaboration diagram for AAX_VController:



14.125.1 Description

Version-managed concrete [Controller](#) class.

For usage information, see [Host-provided interfaces](#)

Public Member Functions

- [AAX_VController \(IACFUnknown *pUnknown\)](#)
- virtual [~AAX_VController \(\)](#)
- virtual [AAX_Result GetEffectID \(AAX_IString *outEffectID\) const AAX_OVERRIDE](#)
- virtual [AAX_Result GetSampleRate \(AAX_CSampleRate *outSampleRate\) const AAX_OVERRIDE](#)

CALL: Returns the current literal sample rate.
- virtual [AAX_Result GetInputStemFormat \(AAX_EStemFormat *outStemFormat\) const AAX_OVERRIDE](#)

CALL: Returns the plug-in's input stem format.
- virtual [AAX_Result GetOutputStemFormat \(AAX_EStemFormat *outStemFormat\) const AAX_OVERRIDE](#)

CALL: Returns the plug-in's output stem format.
- virtual [AAX_Result GetSignalLatency \(int32_t *outSamples\) const AAX_OVERRIDE](#)

CALL: Returns the most recent signal (algorithmic) latency that has been published by the plug-in.
- virtual [AAX_Result GetHybridSignalLatency \(int32_t *outSamples\) const AAX_OVERRIDE](#)

- virtual `AAX_Result GetPlugInTargetPlatform (AAX_CTargetPlatform *outTargetPlatform) const AAX_OVERRIDE`

CALL: Returns the latency between the algorithm normal input samples and the inputs returning from the hybrid component.
- virtual `AAX_Result GetsAudioSuite (AAX_CBoolean *outIsAudioSuite) const AAX_OVERRIDE`

CALL: Returns execution platform type, native or TI.
- virtual `AAX_Result GetCycleCount (AAX_EProperty inWhichCycleCount, AAX_CPropertyValue *outNumCycles) const AAX_OVERRIDE`

CALL: Returns true for AudioSuite instances.

CALL: returns the plug-in's current real-time DSP cycle count.
- virtual `AAX_Result GetTODLocation (AAX_CTimeOfDay *outTODLocation) const AAX_OVERRIDE`

CALL: Returns the current Time Of Day (TOD) of the system.
- virtual `AAX_Result GetCurrentAutomationTimestamp (AAX_CTransportCounter *outTimestamp) const AAX_OVERRIDE`

CALL: Returns the current automation timestamp if called during the `GenerateCoefficients()` call AND the generation of coefficients is being triggered by an automation point instead of immediate changes.
- virtual `AAX_Result GetHostName (AAX_IString *outHostNameString) const AAX_OVERRIDE`

CALL: Returns name of the host application this plug-in instance is being loaded by. This string also typically includes version information.
- virtual `AAX_Result SetSignalLatency (int32_t inNumSamples) AAX_OVERRIDE`

CALL: Submits a request to change the delay compensation value that the host uses to account for the plug-in's signal (algorithmic) latency.
- virtual `AAX_Result SetCycleCount (AAX_EProperty *inWhichCycleCounts, AAX_CPropertyValue *iValues, int32_t numValues) AAX_OVERRIDE`

CALL: Indicates a change in the plug-in's real-time DSP cycle count.
- virtual `AAX_Result PostPacket (AAX_CFieldIndex inFieldIndex, const void *inPayloadP, uint32_t inPayloadSize) AAX_OVERRIDE`

CALL: Posts a data packet to the host for routing between plug-in components.
- virtual `AAX_Result SendNotification (AAX_CTypeID inNotificationType, const void *inNotificationData, uint32_t inNotificationDataSize) AAX_OVERRIDE`

CALL: Dispatch a notification.
- virtual `AAX_Result SendNotification (AAX_CTypeID inNotificationType) AAX_OVERRIDE`

CALL: Sends an event to the GUI (no payload)

Note

Not an AAX interface method
- virtual `AAX_Result GetCurrentMeterValue (AAX_CTypeID inMeterID, float *outMeterValue) const AAX_OVERRIDE`

CALL: Retrieves the current value of a host-managed plug-in meter.
- virtual `AAX_Result GetMeterPeakValue (AAX_CTypeID inMeterID, float *outMeterPeakValue) const AAX_OVERRIDE`

CALL: Retrieves the currently held peak value of a host-managed plug-in meter.
- virtual `AAX_Result ClearMeterPeakValue (AAX_CTypeID inMeterID) const AAX_OVERRIDE`

CALL: Clears the peak value from a host-managed plug-in meter.
- virtual `AAX_Result GetMeterClipped (AAX_CTypeID inMeterID, AAX_CBoolean *outClipped) const AAX_OVERRIDE`

CALL: Retrieves the clipped flag from a host-managed plug-in meter.
- virtual `AAX_Result ClearMeterClipped (AAX_CTypeID inMeterID) const AAX_OVERRIDE`

CALL: Clears the clipped flag from a host-managed plug-in meter.
- virtual `AAX_Result GetMeterCount (uint32_t *outMeterCount) const AAX_OVERRIDE`

CALL: Retrieves the number of host-managed meters registered by a plug-in.
- virtual `AAX_Result GetNextMIDIPacket (AAX_CFieldIndex *outPort, AAX_CMidiPacket *outPacket) AAX_OVERRIDE`

CALL: Retrieves MIDI packets for described MIDI nodes.

- virtual `AAX_IPageTable * CreateTableCopyForEffect (AAX_CPropertyValue inManufacturerID, AAX_CPropertyValue inProductID, AAX_CPropertyValue inPlugInID, uint32_t inTableType, int32_t inTablePageSize)` const `AAX_OVERRIDE`
`Copy the current page table data for a particular plug-in type.`
- virtual `AAX_IPageTable * CreateTableCopyForLayout (const char *inEffectID, const char *inLayoutName, uint32_t inTableType, int32_t inTablePageSize)` const `AAX_OVERRIDE`
`Copy the current page table data for a particular plug-in effect and page table layout.`
- virtual `AAX_IPageTable * CreateTableCopyForEffectFromFile (const char *inPageTableFilePath, AAX_ETextEncoding inFilePathEncoding, AAX_CPropertyValue inManufacturerID, AAX_CPropertyValue inProductID, AAX_CPropertyValue inPlugInID, uint32_t inTableType, int32_t inTablePageSize)` const `AAX_OVERRIDE`
`Copy the current page table data for a particular plug-in type.`
- virtual `AAX_IPageTable * CreateTableCopyForLayoutFromFile (const char *inPageTableFilePath, AAX_ETextEncoding inFilePathEncoding, const char *inLayoutName, uint32_t inTableType, int32_t inTablePageSize)` const `AAX_OVERRIDE`
`Copy the current page table data for a particular plug-in effect and page table layout.`

14.125.2 Constructor & Destructor Documentation

14.125.2.1 `AAX_VController::AAX_VController (IACFUnknown * pUnknown)`

14.125.2.2 `virtual AAX_VController::~AAX_VController () [virtual]`

14.125.3 Member Function Documentation

14.125.3.1 `virtual AAX_Result AAX_VController::GetEffectID (AAX_IString * outEffectID) const [virtual]`

Implements `AAX_IController`.

14.125.3.2 `virtual AAX_Result AAX_VController::GetSampleRate (AAX_CSampleRate * outSampleRate) const [virtual]`

CALL: Returns the current literal sample rate.

Parameters

<code>out</code>	<code>outSampleRate</code>	The current sample rate
------------------	----------------------------	-------------------------

Implements `AAX_IController`.

14.125.3.3 `virtual AAX_Result AAX_VController::GetInputStemFormat (AAX_EStemFormat * outStemFormat) const [virtual]`

CALL: Returns the plug-in's input stem format.

Parameters

<code>out</code>	<code>outStemFormat</code>	The current input stem format
------------------	----------------------------	-------------------------------

Implements `AAX_IController`.

14.125.3.4 `virtual AAX_Result AAX_VController::GetOutputStemFormat (AAX_EStemFormat * outStemFormat) const [virtual]`

CALL: Returns the plug-in's output stem format.

Parameters

<code>out</code>	<code>outStemFormat</code>	The current output stem format
------------------	----------------------------	--------------------------------

Implements [AAX_IController](#).

14.125.3.5 virtual AAX_Result AAX_VController::GetSignalLatency (int32_t * *outSamples*) const [virtual]

CALL: Returns the most recent signal (algorithmic) latency that has been published by the plug-in.

This method provides the most recently published signal latency. The host may not have updated its delay compensation to match this signal latency yet, so plug-ins that dynamically change their latency using [SetSignalLatency\(\)](#) should always wait for an [AAX_eNotificationEvent_SignalLatencyChanged](#) notification before updating its algorithm to incur this latency.

See also

[SetSignalLatency\(\)](#)

Parameters

<code>out</code>	<code>outSamples</code>	The number of samples of signal delay published by the plug-in
------------------	-------------------------	--

Implements [AAX_IController](#).

14.125.3.6 virtual AAX_Result AAX_VController::GetHybridSignalLatency (int32_t * *outSamples*) const [virtual]

CALL: Returns the latency between the algorithm normal input samples and the inputs returning from the hybrid component.

This method provides the number of samples that the AAX host expects the plug-in to delay a signal. The host will use this value when accounting for latency across the system.

Note

This value will generally scale up with sample rate, although it's not a simple multiple due to some fixed overhead. This value will be fixed for any given sample rate regardless of other buffer size settings in the host app.

Parameters

<code>out</code>	<code>outSamples</code>	The number of samples of hybrid signal delay
------------------	-------------------------	--

Implements [AAX_IController](#).

14.125.3.7 virtual AAX_Result AAX_VController::GetPlugInTargetPlatform (AAX_CTargetPlatform * *outTargetPlatform*) const [virtual]

CALL: Returns execution platform type, native or TI.

Parameters

<code>out</code>	<code>outTargetPlatform</code>	The type of the current execution platform as one of AAX_ETargetPlatform .
------------------	--------------------------------	--

Implements [AAX_IController](#).

14.125.3.8 virtual AAX_Result AAX_VController::GetIsAudioSuite (AAX_CBoolean * *outIsAudioSuite*) const [virtual]

CALL: Returns true for AudioSuite instances.

Parameters

<code>out</code>	<code>outIsAudioSuite</code>	The boolean flag which indicate true for AudioSuite instances.
------------------	------------------------------	--

Implements [AAX_IController](#).

14.125.3.9 virtual [AAX_Result](#) [AAX_VController::GetCycleCount](#) ([AAX_EProperty](#) *inWhichCycleCount*, [AAX_CPropertyValue](#) * *outNumCycles*) const [virtual]

CALL: returns the plug-in's current real-time DSP cycle count.

This method provides the number of cycles that the AAX host expects the DSP plug-in to consume. The host uses this value when allocating DSP resources for the plug-in.

Note

A plug-in should never apply a DSP algorithm with more demanding resource requirements than what is currently accounted for by the host. To set a higher cycle count value, a plug-in must call [AAX_IController::SetCycleCount\(\)](#), then poll [AAX_IController::GetCycleCount\(\)](#) until the new value has been applied. Once the host has recognized the new cycle count value, the plug-in may apply the more demanding algorithm.

Parameters

<code>in</code>	<code>inWhichCycleCount</code>	Selector for the requested cycle count metric. One of: <ul style="list-style-type: none"> • AAX_eProperty_TI_SharedCycleCount • AAX_eProperty_TI_InstanceCycleCount • AAX_eProperty_TI_MaxInstancesPerChip
<code>in</code>	<code>outNumCycles</code>	The current value of the selected cycle count metric

Todo PLACEHOLDER - NOT CURRENTLY IMPLEMENTED IN HOST

Implements [AAX_IController](#).

14.125.3.10 virtual [AAX_Result](#) [AAX_VController::GetTODLocation](#) ([AAX_CTimeOfDay](#) * *outTODLocation*) const [virtual]

CALL: Returns the current Time Of Day (TOD) of the system.

This method provides a plug-in the TOD (in samples) of the current system. TOD is the number of samples that the playhead has traversed since the beginning of playback.

Note

The TOD value is the immediate value of the audio engine playhead. This value is incremented within the audio engine's real-time rendering context; it is not synchronized with non-real-time calls to plug-in interface methods.

Parameters

<code>out</code>	<code>outTODLocation</code>	The current Time Of Day as set by the host
------------------	-----------------------------	--

Implements [AAX_IController](#).

14.125.3.11 virtual [AAX_Result](#) [AAX_VController::GetCurrentAutomationTimestamp](#) ([AAX_CTransportCounter](#) * *outTimestamp*) const [virtual]

CALL: Returns the current automation timestamp if called during the [GenerateCoefficients\(\)](#) call AND the generation of coefficients is being triggered by an automation point instead of immediate changes.

Note

This function will return 0 if called from outside of [GenerateCoefficients\(\)](#) or if the [GenerateCoefficients\(\)](#) call was initiated due to a non-automated change. In those cases, you can get your sample offset from the transport start using [GetTODLocation\(\)](#).

Parameters

<code>out</code>	<code>outTimestamp</code>	The current coefficient timestamp. Sample count from transport start.
------------------	---------------------------	---

Implements [AAX_IController](#).

14.125.3.12 virtual AAX_Result AAX_VController::GetHostName (AAX_IString * *outHostNameString*) const [virtual]

CALL: Returns name of the host application this plug-in instance is being loaded by. This string also typically includes version information.

Host Compatibility Notes Pro Tools versions from Pro Tools 11.0 to Pro Tools 12.3.1 will return a generic version string to this call. This issue is resolved beginning in Pro Tools 12.4.

Parameters

<code>out</code>	<code>outHostName</code>	The name of the current host application.
------------------	--------------------------	---

Implements [AAX_IController](#).

14.125.3.13 virtual AAX_Result AAX_VController::SetSignalLatency (int32_t *inNumSamples*) [virtual]

CALL: Submits a request to change the delay compensation value that the host uses to account for the plug-in's signal (algorithmic) latency.

This method is used to request a change in the number of samples that the AAX host expects the plug-in to delay a signal.

The host is not guaranteed to immediately apply the new latency value. A plug-in should avoid incurring an actual algorithmic latency that is different than the latency accounted for by the host.

To set a new latency value, a plug-in must call [AAX_IController::SetSignalLatency\(\)](#), then wait for an [AAX_eNotificationEvent_SignalLatencyChanged](#) notification. Once this notification has been received, [AAX_IController::GetSignalLatency\(\)](#) will reflect the updated latency value and the plug-in should immediately apply any relevant algorithmic changes that alter its latency to this new value.

Warning

Parameters which affect the latency of a plug-in should not be made available for control through automation. This will result in audible glitches when delay compensation is adjusted while playing back automation for these parameters.

Parameters

<code>in</code>	<code>inNumSamples</code>	The number of samples of signal delay that the plug-in requests to incur
-----------------	---------------------------	--

Implements [AAX_IController](#).

14.125.3.14 virtual AAX_Result AAX_VController::SetCycleCount (AAX_EProperty * *inWhichCycleCounts*, AAX_CPropertyValue * *iValues*, int32_t *numValues*) [virtual]

CALL: Indicates a change in the plug-in's real-time DSP cycle count.

This method is used to request a change in the number of cycles that the AAX host expects the DSP plug-in to consume.

Note

A plug-in should never apply a DSP algorithm with more demanding resource requirements than what is currently accounted for by the host. To set a higher cycle count value, a plug-in must call [AAX_IController::SetCycleCount\(\)](#), then poll [AAX_IController::GetCycleCount\(\)](#) until the new value has been applied. Once the host has recognized the new cycle count value, the plug-in may apply the more demanding algorithm.

Parameters

in	<i>inWhichCycleCounts</i>	Array of selectors indicating the specific cycle count metrics that should be set. Each selector must be one of: <ul style="list-style-type: none"> • AAX_eProperty_TI_SharedCycleCount • AAX_eProperty_TI_InstanceCycleCount • AAX_eProperty_TI_MaxInstancesPerChip
in	<i>iValues</i>	An array of values requested, one for each of the selected cycle count metrics.
in	<i>numValues</i>	The size of <i>iValues</i>

Todo PLACEHOLDER - NOT CURRENTLY IMPLEMENTED IN HOST

Implements [AAX_IController](#).

14.125.3.15 virtual [AAX_Result](#) [AAX_VController::PostPacket](#) ([AAX_CFieldIndex](#) *inFieldIndex*, const void * *inPayloadP*, [uint32_t](#) *inPayloadSize*) [virtual]

CALL: Posts a data packet to the host for routing between plug-in components.

The posted packet is identified with a [AAX_CFieldIndex](#) packet index value, which is equivalent to the target data port's identifier. The packet's payload must have the expected size for the given packet index / data port, as defined when the port is created in [Describe](#). See [AAX_IComponentDescriptor::AddDataInPort\(\)](#).

Warning

Any data structures that will be passed between platforms (for example, sent to a TI DSP in an AAX DSP plug-in) must be properly data-aligned for compatibility across both platforms. See [AAX_ALIGN_FILE_ALG](#) for more information about guaranteeing cross-platform compatibility of data structures used for algorithm processing.

Note

All calls to this method should be made within the scope of [AAX_IEffectParameters::GenerateCoefficients\(\)](#). Calls from outside this method may result in packets not being delivered. See [PT-206161](#)

Parameters

in	<i>inFieldIndex</i>	The packet's destination port
in	<i>inPayloadP</i>	A pointer to the packet's payload data
in	<i>inPayloadSize</i>	The size, in bytes, of the payload data

Implements [AAX_IController](#).

14.125.3.16 virtual AAX_Result AAX_VController::SendNotification (**AAX_CTypeID** *inNotificationType*, const void * *inNotificationData*, uint32_t *inNotificationDataSize*) [virtual]

CALL: Dispatch a notification.

The notification is handled by the host and may be delivered back to other plug-in components such as the G↔UI or data model (via [AAX_IEffectGUI::NotificationReceived\(\)](#) or [AAX_IEffectParameters::NotificationReceived\(\)](#), respectively) depending on the notification type.

The host may choose to dispatch the posted notification either synchronously or asynchronously.

See the [AAX_ENotificationEvent](#) documentation for more information.

This method is supported by AAX V2 Hosts only. Check the return code on the return of this function. If the error is [AAX_ERROR_UNIMPLEMENTED](#), your plug-in is being loaded into a host that doesn't support this feature.

Parameters

in	<i>inNotificationType</i>	Type of notification to send
in	<i>inNotificationData</i>	Block of notification data
in	<i>inNotificationDataSize</i>	Size of <i>inNotificationData</i> , in bytes

Implements [AAX_IController](#).

14.125.3.17 virtual AAX_Result AAX_VController::SendNotification (**AAX_CTypeID** *inNotificationType*) [virtual]

CALL: Sends an event to the GUI (no payload)

Note

Not an [AAX](#) interface method

This version of the notification method is a convenience for notifications which do not take any payload data. Internally, it simply calls [AAX_IController::SendNotification\(AAX_CTypeID, const void*, uint32_t\)](#) with a null payload.

Parameters

in	<i>inNotificationType</i>	Type of notification to send
----	---------------------------	------------------------------

Note

Not an [AAX](#) interface method

Implements [AAX_IController](#).

14.125.3.18 virtual AAX_Result AAX_VController::GetCurrentMeterValue (**AAX_CTypeID** *inMeterID*, float * *outMeterValue*) const [virtual]

CALL: Retrieves the current value of a host-managed plug-in meter.

Parameters

in	<i>inMeterID</i>	ID of the meter that is being queried
out	<i>outMeterValue</i>	The queried meter's current value

Implements [AAX_IController](#).

14.125.3.19 virtual AAX_Result AAX_VController::GetMeterPeakValue (AAX_CTypeID *inMeterID*, float * *outMeterPeakValue*) const [virtual]

CALL: Retrieves the currently held peak value of a host-managed plug-in meter.

Parameters

in	<i>inMeterID</i>	ID of the meter that is being queried
out	<i>outMeterPeakValue</i>	The queried meter's currently held peak value

Implements [AAX_IController](#).

14.125.3.20 virtual AAX_Result AAX_VController::ClearMeterPeakValue (AAX_CTypeID *inMeterID*) const [virtual]

CALL: Clears the peak value from a host-managed plug-in meter.

Parameters

in	<i>inMeterID</i>	ID of the meter that is being cleared
----	------------------	---------------------------------------

Implements [AAX_IController](#).

14.125.3.21 virtual AAX_Result AAX_VController::GetMeterClipped (AAX_CTypeID *inMeterID*, AAX_CBoolean * *outClipped*) const [virtual]

CALL: Retrieves the clipped flag from a host-managed plug-in meter.

See [AAX_IComponentDescriptor::AddMeters\(\)](#).

Parameters

in	<i>inMeterID</i>	ID of the meter that is being queried.
out	<i>outClipped</i>	The queried meter's clipped flag.

Implements [AAX_IController](#).

14.125.3.22 virtual AAX_Result AAX_VController::ClearMeterClipped (AAX_CTypeID *inMeterID*) const [virtual]

CALL: Clears the clipped flag from a host-managed plug-in meter.

See [AAX_IComponentDescriptor::AddMeters\(\)](#).

Parameters

in	<i>inMeterID</i>	ID of the meter that is being cleared.
----	------------------	--

Implements [AAX_IController](#).

14.125.3.23 virtual AAX_Result AAX_VController::GetMeterCount (uint32_t * *outMeterCount*) const [virtual]

CALL: Retrieves the number of host-managed meters registered by a plug-in.

See [AAX_IComponentDescriptor::AddMeters\(\)](#).

Parameters

<code>out</code>	<code>outMeterCount</code>	The number of registered plug-in meters.
------------------	----------------------------	--

Implements [AAX_IController](#).

14.125.3.24 virtual **AAX_Result** **AAX_VController::GetNextMIDIpacket** (**AAX_CFieldIndex** * *outPort*, **AAX_CMidiPacket** * *outPacket*) [virtual]

CALL: Retrieves MIDI packets for described MIDI nodes.

Parameters

<code>out</code>	<code>outPort</code>	port ID of the MIDI node that has unhandled packet
<code>out</code>	<code>outPacket</code>	The MIDI packet

Implements [AAX_IController](#).

14.125.3.25 virtual **AAX_IPageTable*** **AAX_VController::CreateTableCopyForEffect** (**AAX_CPropertyValue** *inManufacturerID*, **AAX_CPropertyValue** *inProductID*, **AAX_CPropertyValue** *inPlugInID*, **uint32_t** *inTableType*, **int32_t** *inTablePageSize*) const [virtual]

Copy the current page table data for a particular plug-in type.

The host may restrict plug-ins to only copying page table data from certain plug-in types, such as plug-ins from the same manufacturer or plug-in types within the same effect.

See [Page Table Guide](#) for more information about page tables.

Returns

A new page table object to which the requested page table data has been copied. Ownership of this object passes to the caller.

a null pointer if the requested plug-in type is unknown, if *inTableType* is unknown or if *inTablePageSize* is not a supported size for the given table type.

Parameters

<code>in</code>	<i>inManufacturerID</i>	Manufacturer ID of the desired plug-in type
<code>in</code>	<i>inProductID</i>	Product ID of the desired plug-in type
<code>in</code>	<i>inPlugInID</i>	Type ID of the desired plug-in type (AAX_eProperty_PlugInID_Native , AAX_eProperty_PlugInID_TI)
<code>in</code>	<i>inTableType</i>	Four-char type identifier for the requested table type (e.g. 'PgTL', 'Av81', etc.)
<code>in</code>	<i>inTablePageSize</i>	Page size for the requested table. Some tables support multiple page sizes.

Implements [AAX_IController](#).

14.125.3.26 virtual **AAX_IPageTable*** **AAX_VController::CreateTableCopyForLayout** (`const char` * *inEffectID*, `const char` * *inLayoutName*, `uint32_t` *inTableType*, `int32_t` *inTablePageSize*) const [virtual]

Copy the current page table data for a particular plug-in effect and page table layout.

The host may restrict plug-ins to only copying page table data from certain effects, such as effects registered within the current [AAX](#) plug-in bundle.

See [Page Table Guide](#) for more information about page tables.

Returns

A new page table object to which the requested page table data has been copied. Ownership of this object passes to the caller.
 a null pointer if the requested effect ID is unknown or if `inLayoutName` is not a valid layout name for the page tables registered for the effect.

Parameters

in	<code>inEffectID</code>	Effect ID for the desired effect. See AAX_ICollection::AddEffect()
in	<code>inLayoutName</code>	Page table layout name ("name" attribute of the PTLayout XML tag)
in	<code>inTableType</code>	Four-char type identifier for the requested table type (e.g. 'PgTL', 'Av81', etc.)
in	<code>inTablePageSize</code>	Page size for the requested table. Some tables support multiple page sizes.

Implements [AAX_IController](#).

14.125.3.27 virtual [AAX_IPageTable](#)* [AAX_VController::CreateTableCopyForEffectFromFile](#) (const char * `inPageTableFilePath`, [AAX_ETextEncoding](#) `inFilePathEncoding`, [AAX_CPropertyValue](#) `inManufacturerID`, [AAX_CPropertyValue](#) `inProductID`, [AAX_CPropertyValue](#) `inPlugInID`, uint32_t `inTableType`, int32_t `inTablePageSize`) const [virtual]

Copy the current page table data for a particular plug-in type.

Returns

A new page table object to which the requested page table data has been copied. Ownership of this object passes to the caller.
 a null pointer if the requested plug-in type is unknown, if `inTableType` is unknown or if `inTablePage↔Size` is not a supported size for the given table type.

Parameters

in	<code>inPageTable↔FilePath</code>	Path to XML page table file.
in	<code>inFilePath↔Encoding</code>	File path text encoding.
in	<code>in↔ManufacturerID</code>	Manufacturer ID of the desired plug-in type
in	<code>inProductID</code>	Product ID of the desired plug-in type
in	<code>inPlugInID</code>	Type ID of the desired plug-in type (AAX_eProperty_PlugInID_Native , AAX_eProperty_PlugInID_TI)
in	<code>inTableType</code>	Four-char type identifier for the requested table type (e.g. 'PgTL', 'Av81', etc.)
in	<code>inTablePageSize</code>	Page size for the requested table. Some tables support multiple page sizes.

Implements [AAX_IController](#).

14.125.3.28 virtual [AAX_IPageTable](#)* [AAX_VController::CreateTableCopyForLayoutFromFile](#) (const char * `inPageTableFilePath`, [AAX_ETextEncoding](#) `inFilePathEncoding`, const char * `inLayoutName`, uint32_t `inTableType`, int32_t `inTablePageSize`) const [virtual]

Copy the current page table data for a particular plug-in effect and page table layout.

Returns

A new page table object to which the requested page table data has been copied. Ownership of this object passes to the caller.
 a null pointer if `inLayoutName` is not a valid layout name for the page tables file.

Parameters

in	<i>inPageTableFilePath</i>	Path to XML page table file.
in	<i>inFilePathEncoding</i>	File path text encoding.
in	<i>inLayoutName</i>	Page table layout name ("name" attribute of the <code>PTLayout</code> XML tag)
in	<i>inTableType</i>	Four-char type identifier for the requested table type (e.g. 'PgTL', 'Av81', etc.)
in	<i>inTablePageSize</i>	Page size for the requested table. Some tables support multiple page sizes.

Implements [AAX_IController](#).

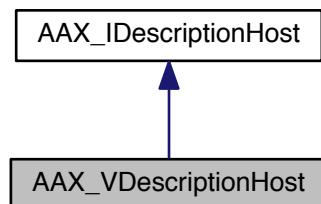
The documentation for this class was generated from the following file:

- [AAX_VController.h](#)

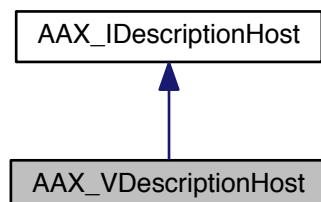
14.126 AAX_VDescriptionHost Class Reference

```
#include <AAX_VDescriptionHost.h>
```

Inheritance diagram for AAX_VDescriptionHost:



Collaboration diagram for AAX_VDescriptionHost:



14.126.1 Description

Versioned wrapper for access to host service interfaces provided during plug-in description

This object aggregates access to [AAX_IACFDescriptionHost](#) and [IACFDefinition](#), with support depending on the interface support level of the [IACFUnknown](#) which is passed to this object upon creation.

Public Member Functions

- [AAX_VDescriptionHost \(IACFUnknown *pUnknown\)](#)
- virtual [~AAX_VDescriptionHost \(\)](#)
- virtual const [AAX_IFeatureInfo * AcquireFeatureProperties \(const AAX_Feature_UID &inFeatureID\) const AAX_OVERRIDE](#)
- bool [Supported \(\) const](#)
- [AAX_IACFDescriptionHost * DescriptionHost \(\)](#)
- const [AAX_IACFDescriptionHost * DescriptionHost \(\) const](#)
- [IACFDefinition * HostDefinition \(\) const](#)

14.126.2 Constructor & Destructor Documentation

14.126.2.1 [AAX_VDescriptionHost::AAX_VDescriptionHost \(IACFUnknown * pUnknown \) \[explicit\]](#)

14.126.2.2 [virtual AAX_VDescriptionHost::~AAX_VDescriptionHost \(\) \[virtual\]](#)

14.126.3 Member Function Documentation

14.126.3.1 [virtual const AAX_IFeatureInfo* AAX_VDescriptionHost::AcquireFeatureProperties \(const AAX_Feature_UID & inFeatureID \) const \[virtual\]](#)

Get the client's feature object for a given feature ID

Similar to [QueryInterface \(\)](#) but uses a feature identifier rather than a true IID

Ownership of the returned object is passed to the caller; the caller is responsible for destroying the object, e.g. by capturing the returned object in a smart pointer.

```
// AAX_IDescriptionHost* descHost
std::unique_ptr<const AAX_IFeatureInfo> featureInfoPtr (descHost->AcquireFeatureProperties(someFeatureUID);
```

Returns

An [AAX_IFeatureInfo](#) interface with access to the host's feature properties for this feature.
NULL if the desired feature was not found or if an error occurred

Note

May return an [AAX_IFeatureInfo](#) object with limited method support, which would return an error such as [AAX_ERROR_NULL_OBJECT](#) or [AAX_ERROR_UNIMPLEMENTED](#) to interface calls.

If no [AAX_IFeatureInfo](#) is provided then that may mean that the host is unaware of the feature, or it may mean that the host is aware of the feature but has not implemented the AAX feature support interface for this feature yet.

Parameters

in	<i>inFeatureID</i>	Identifier of the requested feature
----	--------------------	-------------------------------------

Implements [AAX_IDescriptionHost](#).

14.126.3.2 bool [AAX_VDescriptionHost::Supported\(\)](#) const [inline]

14.126.3.3 [AAX_IACFDescriptionHost*](#) [AAX_VDescriptionHost::DescriptionHost\(\)](#) [inline]

14.126.3.4 const [AAX_IACFDescriptionHost*](#) [AAX_VDescriptionHost::DescriptionHost\(\)](#) const [inline]

14.126.3.5 [IACFDefinition*](#) [AAX_VDescriptionHost::HostDefinition\(\)](#) const [inline]

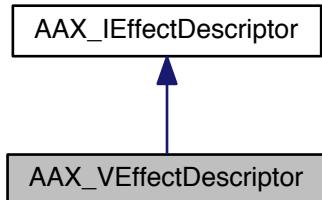
The documentation for this class was generated from the following file:

- [AAX_VDescriptionHost.h](#)

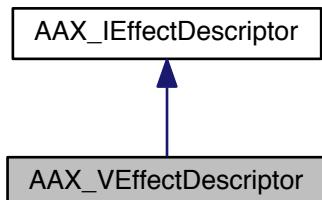
14.127 AAX_VEffectDescriptor Class Reference

#include <[AAX_VEffectDescriptor.h](#)>

Inheritance diagram for AAX_VEffectDescriptor:



Collaboration diagram for AAX_VEffectDescriptor:



14.127.1 Description

Version-managed concrete [AAX_VEffectDescriptor](#) class.

Public Member Functions

- [AAX_VEffectDescriptor](#) ([IACFUnknown](#) **pUnkHost*)
- virtual [~AAX_VEffectDescriptor](#) ()
- virtual [AAX_IComponentDescriptor](#) * [NewComponentDescriptor](#) () [AAX_OVERRIDE](#)
Create an instance of a component descriptor.
- virtual [AAX_Result](#) [AddComponent](#) ([AAX_IComponentDescriptor](#) **inComponentDescriptor*) [AAX_OVERRIDE](#)
Add a component to an instance of a component descriptor.
- virtual [AAX_Result](#) [AddName](#) (const char **inPlugInName*) [AAX_OVERRIDE](#)
Add a name to the Effect.
- virtual [AAX_Result](#) [AddCategory](#) (uint32_t *inCategory*) [AAX_OVERRIDE](#)
Add a category to your plug-in. See [AAX_EPlugInCategory](#).
- virtual [AAX_Result](#) [AddCategoryBypassParameter](#) (uint32_t *inCategory*, [AAX_CParamID](#) *inParamID*) [AAX_OVERRIDE](#)
Add a category to your plug-in. See [AAX_EPlugInCategory](#).
- virtual [AAX_Result](#) [AddProcPtr](#) (void **inProcPtr*, [AAX_CProcPtrID](#) *inProcID*) [AAX_OVERRIDE](#)
Add a process pointer.
- virtual [AAX_IPropertyMap](#) * [NewPropertyMap](#) () [AAX_OVERRIDE](#)
Create a new property map.
- virtual [AAX_Result](#) [SetProperties](#) ([AAX_IPropertyMap](#) **inProperties*) [AAX_OVERRIDE](#)
Set the properties of a new property map.
- virtual [AAX_Result](#) [AddResourceInfo](#) ([AAX_EResourceType](#) *inResourceType*, const char **inInfo*) [AAX_OVERRIDE](#)
Set resource file info.
- virtual [AAX_Result](#) [AddMeterDescription](#) ([AAX_CTypeID](#) *inMeterID*, const char **inMeterName*, [AAX_IPropertyMap](#) **inProperties*) [AAX_OVERRIDE](#)
Add name and property map to meter with given ID.
- virtual [AAX_Result](#) [AddControlMIDI](#) ([AAX_CTypeID](#) *inNodeID*, [AAX_EMIDINodeType](#) *inNodeType*, const char **innodeName*[], uint32_t *inChannelMask*) [AAX_OVERRIDE](#)
Add a control MIDI node to the plug-in data model.
- [IACFUnknown](#) * [GetIUnknown](#) (void) const

14.127.2 Constructor & Destructor Documentation

14.127.2.1 [AAX_VEffectDescriptor::AAX_VEffectDescriptor](#) ([IACFUnknown](#) **pUnkHost*)

14.127.2.2 virtual [AAX_VEffectDescriptor](#)::[~AAX_VEffectDescriptor](#) () [virtual]

14.127.3 Member Function Documentation

14.127.3.1 virtual [AAX_IComponentDescriptor](#)* [AAX_VEffectDescriptor::NewComponentDescriptor](#) () [virtual]

Create an instance of a component descriptor.

This implementation retains each generated [AAX_IComponentDescriptor](#) and destroys the property map upon [AAX_VEffectDescriptor](#) destruction

Implements [AAX_IEffectDescriptor](#).

14.127.3.2 virtual AAX_Result AAX_VEffectDescriptor::AddComponent (AAX_IComponentDescriptor * *inComponentDescriptor*) [virtual]

Add a component to an instance of a component descriptor.

Unlike with [AAX_ICollection::AddEffect\(\)](#), the [AAX_IEffectDescriptor](#) does not take ownership of the [AAX_IComponentDescriptor](#) that is passed to it in this method. The host copies out the contents of this descriptor, and thus the plug-in may re-use the same descriptor object when creating additional similar components.

Parameters

in	<i>inComponentDescriptor</i>	
----	------------------------------	--

Implements [AAX_IEffectDescriptor](#).

14.127.3.3 virtual AAX_Result AAX_VEffectDescriptor::AddName (const char * *inPlugInName*) [virtual]

Add a name to the Effect.

May be called multiple times to add abbreviated Effect names.

Note

Every Effect must include at least one name variant with 31 or fewer characters, plus a null terminating character

Parameters

in	<i>inPlugInName</i>	The name assigned to the plug-in.
----	---------------------	-----------------------------------

Implements [AAX_IEffectDescriptor](#).

14.127.3.4 virtual AAX_Result AAX_VEffectDescriptor::AddCategory (uint32_t *inCategory*) [virtual]

Add a category to your plug-in. See [AAX_EPlugInCategory](#).

Parameters

in	<i>inCategory</i>	One of the categories for the plug-in.
----	-------------------	--

Implements [AAX_IEffectDescriptor](#).

14.127.3.5 virtual AAX_Result AAX_VEffectDescriptor::AddCategoryBypassParameter (uint32_t *inCategory*, AAX_CParamID *inParamID*) [virtual]

Add a category to your plug-in. See [AAX_EPlugInCategory](#).

Parameters

in	<i>inCategory</i>	One of the categories for the plug-in.
in	<i>inParamID</i>	The parameter ID of the parameter used to bypass the category section of the plug-in.

Implements [AAX_IEffectDescriptor](#).

14.127.3.6 virtual AAX_Result AAX_VEffectDescriptor::AddProcPtr (void * *inProcPtr*, AAX_CProcPtrID *inProcID*) [virtual]

Add a process pointer.

Parameters

in	<i>inProcPtr</i>	A process pointer.
in	<i>inProcID</i>	A process ID.

Implements [AAX_IEffectDescriptor](#).

14.127.3.7 virtual AAX_IPropertyMap* AAX_VEffectDescriptor::NewPropertyMap() [virtual]

Create a new property map.

This implementation retains each generated [AAX_IPropertyMap](#) and destroys the property map upon [AAX_VEffectDescriptor](#) destruction

Implements [AAX_IEffectDescriptor](#).

14.127.3.8 virtual AAX_Result AAX_VEffectDescriptor::SetProperties(AAX_IPropertyMap * *inProperties*) [virtual]

Set the properties of a new property map.

Parameters

in	<i>inProperties</i>	Description
----	---------------------	-------------

Implements [AAX_IEffectDescriptor](#).

14.127.3.9 virtual AAX_Result AAX_VEffectDescriptor::AddResourceInfo(AAX_EResourceType *in ResourceType*, const char * *inInfo*) [virtual]

Set resource file info.

Parameters

in	<i>in ResourceType</i>	See AAX_EResourceType .
in	<i>inInfo</i>	Definition varies on the resource type.

Implements [AAX_IEffectDescriptor](#).

14.127.3.10 virtual AAX_Result AAX_VEffectDescriptor::AddMeterDescription(AAX_CTypeID *in MeterID*, const char * *in MeterName*, AAX_IPropertyMap * *inProperties*) [virtual]

Add name and property map to meter with given ID.

Parameters

in	<i>in MeterID</i>	The ID of the meter being described.
in	<i>in MeterName</i>	The name of the meter.
in	<i>inProperties</i>	The property map containing meter related data such as meter type, orientation, etc.

Implements [AAX_IEffectDescriptor](#).

14.127.3.11 virtual AAX_Result AAX_VEffectDescriptor::AddControlMIDINode(AAX_CTypeID *in NodeID*, AAX_EMIDINodeType *in NodeType*, const char *in NodeName*[], uint32_t *in ChannelMask*) [virtual]

Add a control MIDI node to the plug-in data model.

- This MIDI node may receive note data as well as control data.

- To send MIDI data to the plug-in's algorithm, use [AAX_IComponentDescriptor::AddMIDINode\(\)](#).

See also

[AAX_IACFEffectorParameters_V2::UpdateControlMIDINodes\(\)](#)

Parameters

in	<i>inNodeID</i>	The ID for the new control MIDI node.
in	<i>inNodeType</i>	The type of the node.
in	<i>innodeName</i>	The name of the node.
in	<i>inChannelMask</i>	The bit mask for required nodes channels (up to 16) or required global events for global node.

Implements [AAX_IEffectDescriptor](#).

14.127.3.12 IACFUnknown* AAX_VEffectDescriptor::GetIUnknown (void) const

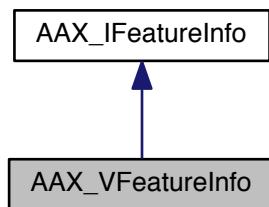
The documentation for this class was generated from the following file:

- [AAX_VEffectDescriptor.h](#)

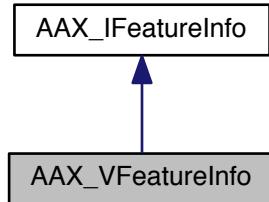
14.128 AAX_VFeatureInfo Class Reference

```
#include <AAX_VFeatureInfo.h>
```

Inheritance diagram for AAX_VFeatureInfo:



Collaboration diagram for AAX_VFeatureInfo:



14.128.1 Description

Concrete implementation of [AAX_IFeatureInfo](#), which provides a version-controlled interface to host feature information

Public Member Functions

- [AAX_VFeatureInfo \(IACFUnknown *pUnknown, const AAX_Feature_UID &inFeatureID\)](#)
- virtual [~AAX_VFeatureInfo \(\)](#)
- virtual [AAX_Result SupportLevel \(AAX_ESupportLevel &oSupportLevel\) const AAX_OVERRIDE](#)
- virtual const [AAX_IPropertyMap * AcquireProperties \(\) const AAX_OVERRIDE](#)
- virtual const [AAX_Feature_UID & ID \(\) const AAX_OVERRIDE](#)

14.128.2 Constructor & Destructor Documentation

14.128.2.1 AAX_VFeatureInfo::AAX_VFeatureInfo (IACFUnknown * pUnknown, const AAX_Feature_UID & inFeatureID) [explicit]

14.128.2.2 virtual AAX_VFeatureInfo::~AAX_VFeatureInfo () [virtual]

14.128.3 Member Function Documentation

14.128.3.1 virtual AAX_Result AAX_VFeatureInfo::SupportLevel (AAX_ESupportLevel & oSupportLevel) const [virtual]

Determine the level of support for this feature by the host

Note

The host will not provide an underlying [AAX_IACFFeatureInfo](#) interface for features which it does not recognize at all, resulting in a [AAX_ERROR_NULL_OBJECT](#) error code

Implements [AAX_IFeatureInfo](#).

14.128.3.2 virtual const AAX_IPropertyMap* AAX_VFeatureInfo::AcquireProperties () const [virtual]

Additional properties providing details of the feature support

See the feature's UID for documentation of which features provide additional properties

Ownership of the returned object is passed to the caller; the caller is responsible for destroying the object, e.g. by capturing the returned object in a smart pointer.

```
// AAX_IFeatureInfo* featureInfo
std::unique_ptr<const AAX_IPropertyMap> featurePropertiesPtr(featureInfo->AcquireProperties());
```

Returns

An [AAX_IPropertyMap](#) interface with access to the host's properties for this feature.
NULL if the desired feature was not found or if an error occurred

Note

May return an [AAX_IPropertyMap](#) object with limited method support, which would return an error such as [AAX_ERROR_NULL_OBJECT](#) or [AAX_ERROR_UNIMPLEMENTED](#) to interface calls.

Implements [AAX_IFeatureInfo](#).

14.128.3.3 virtual const AAX_Feature_UID& AAX_VFeatureInfo::ID() const [virtual]

Returns the ID of the feature which this object represents

Implements [AAX_IFeatureInfo](#).

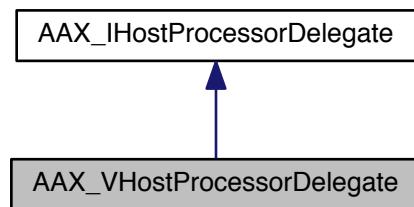
The documentation for this class was generated from the following file:

- [AAX_VFeatureInfo.h](#)

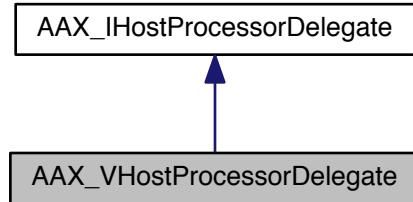
14.129 AAX_VHostProcessorDelegate Class Reference

```
#include <AAX_VHostProcessorDelegate.h>
```

Inheritance diagram for AAX_VHostProcessorDelegate:



Collaboration diagram for AAX_VHostProcessorDelegate:



14.129.1 Description

Version-managed concrete [Host Processor delegate](#) class.

Public Member Functions

- [AAX_VHostProcessorDelegate \(IACFUnknown *pUnknown\)](#)
- virtual [AAX_Result GetAudio \(const float *const inAudioIns\[\], int32_t inAudioInCount, int64_t inLocation, int32_t *ioNumSamples\) AAX_OVERRIDE](#)
CALL: Randomly access audio from the timeline.
- virtual [int32_t GetSideChainInputNum \(\) AAX_OVERRIDE](#)
CALL: Returns the index of the side chain input buffer.
- virtual [AAX_Result ForceAnalyze \(\) AAX_OVERRIDE](#)
CALL: Request an analysis pass.
- virtual [AAX_Result ForceProcess \(\) AAX_OVERRIDE](#)
CALL: Request a process pass.

14.129.2 Constructor & Destructor Documentation

14.129.2.1 [AAX_VHostProcessorDelegate::AAX_VHostProcessorDelegate \(IACFUnknown * pUnknown \)](#)

14.129.3 Member Function Documentation

14.129.3.1 [virtual AAX_Result AAX_VHostProcessorDelegate::GetAudio \(const float *const inAudioIns\[\], int32_t inAudioInCount, int64_t inLocation, int32_t * ioNumSamples \) \[virtual\]](#)

CALL: Randomly access audio from the timeline.

Called from within [AAX_IHostProcessor::RenderAudio\(\)](#), this method fills a buffer of samples with randomly-accessed data from the current input processing region on the timeline, including any extra samples such as processing "handles".

Note

Plug-ins that use this feature must set [AAX_eProperty_UsesRandomAccess](#) to true
It is not possible to retrieve samples from outside of the current input processing region
Always check the return value of this method before using the randomly-accessed samples

Parameters

in	<i>inAudioIns</i>	Timeline audio buffer(s). This must be set to <i>inAudioIns</i> from AAX_IHostProcessor::RenderAudio()
----	-------------------	--

Parameters

in	<i>inAudioInCount</i>	Number of buffers in <i>inAudioIns</i> . This must be set to <i>inAudioInCount</i> from AAX_IHostProcessor::RenderAudio()
----	-----------------------	---

Parameters

in	<i>inLocation</i>	A sample location relative to the beginning of the currently processed region, e.g. a value of 0 corresponds to the timeline location returned by AAX_CHostProcessor::GetSrcStart()
in, out	<i>ioNumSamples</i>	<ul style="list-style-type: none"> • Input: The maximum number of samples to read. • Output: The actual number of samples that were read from the timeline

Implements [AAX_IHostProcessorDelegate](#).

14.129.3.2 virtual int32_t AAX_VHostProcessorDelegate::GetSideChainInputNum() [virtual]

CALL: Returns the index of the side chain input buffer.

Called from within [AAX_IHostProcessor::RenderAudio\(\)](#), this method returns the index of the side chain input sample buffer within *inAudioIns*.

Implements [AAX_IHostProcessorDelegate](#).

14.129.3.3 virtual AAX_Result AAX_VHostProcessorDelegate::ForceAnalyze() [virtual]

CALL: Request an analysis pass.

Call this method to request an analysis pass from within the plug-in. Most plug-ins should rely on the host to trigger analysis passes when appropriate. However, plug-ins that require an analysis pass a) outside of the context of host-driven render or analysis, or b) when internal plug-in data changes may need to call [ForceAnalyze\(\)](#).

Implements [AAX_IHostProcessorDelegate](#).

14.129.3.4 virtual AAX_Result AAX_VHostProcessorDelegate::ForceProcess() [virtual]

CALL: Request a process pass.

Call this method to request a process pass from within the plug-in. If [AAX_eProperty_RequiresAnalysis](#) is defined, the resulting process pass will be preceded by an analysis pass. This method should only be used in rare circumstances by plug-ins that must launch processing outside of the normal host AudioSuite workflow.

Implements [AAX_IHostProcessorDelegate](#).

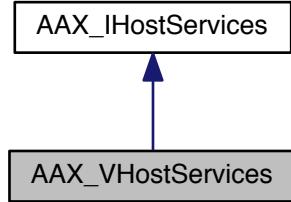
The documentation for this class was generated from the following file:

- [AAX_VHostProcessorDelegate.h](#)

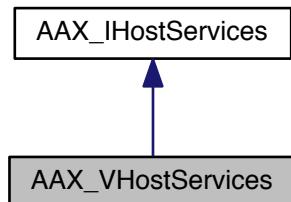
14.130 AAX_VHostServices Class Reference

```
#include <AAX_VHostServices.h>
```

Inheritance diagram for AAX_VHostServices:



Collaboration diagram for AAX_VHostServices:



14.130.1 Description

Version-managed concrete [AAX_IHostServices](#) class.

Public Member Functions

- [AAX_VHostServices \(IACFUnknown *pUnkHost\)](#)
- [~AAX_VHostServices \(\)](#)
- virtual [AAX_Result HandleAssertFailure \(const char *iFile, int32_t iLine, const char *iNote, int32_t iFlags\) const AAX_OVERRIDE](#)
Handle an assertion failure.
- virtual [AAX_Result Trace \(int32_t iPriority, const char *iMessage\) const AAX_OVERRIDE](#)
Log a trace message.
- virtual [AAX_Result StackTrace \(int32_t iTracePriority, int32_t iStackTracePriority, const char *iMessage\) const AAX_OVERRIDE](#)
Log a trace message or a stack trace.

14.130.2 Constructor & Destructor Documentation

14.130.2.1 AAX_VHostServices::AAX_VHostServices (IACFUnknown * pUnkHost)

14.130.2.2 AAX_VHostServices::~AAX_VHostServices()

14.130.3 Member Function Documentation

14.130.3.1 virtual AAX_Result AAX_VHostServices::HandleAssertFailure (const char * *iFile*, int32_t *iLine*, const char * *iNote*, int32_t *iFlags*) const [virtual]

Handle an assertion failure.

Use this method to delegate assertion failure handling to the host

Use *iFlags* to request that specific behavior be included when handling the failure. This request may not be fulfilled by the host, and absence of a flag does not preclude the host from using that behavior when handling the failure.

Parameters

in	<i>iFile</i>	The name of the file containing the assert check. Usually <code>__FILE__</code>
in	<i>iLine</i>	The line number of the assert check. Usually <code>__LINE__</code>
in	<i>iNote</i>	Text to display related to the assert. Usually the condition which failed
in	<i>iFlags</i>	Bitfield of AAX_EAssertFlags to request specific handling behavior

Implements [AAX_IHostServices](#).

14.130.3.2 virtual AAX_Result AAX_VHostServices::Trace (int32_t *iPriority*, const char * *iMessage*) const [virtual]

Log a trace message.

Parameters

in	<i>iPriority</i>	Priority of the trace, used for log filtering. One of <code>kAAX_Trace_Priority_Low</code> , <code>kAAX_Trace_Priority_Normal</code> , <code>kAAX_Trace_Priority_High</code>
in	<i>iMessage</i>	Message string to log

Implements [AAX_IHostServices](#).

14.130.3.3 virtual AAX_Result AAX_VHostServices::StackTrace (int32_t *iTracePriority*, int32_t *iStackTracePriority*, const char * *iMessage*) const [virtual]

Log a trace message or a stack trace.

If the logging output filtering is set to include logs with *iStackTracePriority* then both the logging message and a stack trace will be emitted, regardless of *iTracePriority*.

If the logging output filtering is set to include logs with *iTracePriority* but to exclude logs with *iStackTracePriority* then this will emit a normal log with no stack trace.

Parameters

in	<i>iTracePriority</i>	Priority of the trace, used for log filtering. One of <code>kAAX_Trace_Priority_Low</code> , <code>kAAX_Trace_Priority_Normal</code> , <code>kAAX_Trace_Priority_High</code>
in	<i>iStackTracePriority</i>	Priority of the stack trace, used for log filtering. One of <code>kAAX_Trace_Priority_Low</code> , <code>kAAX_Trace_Priority_Normal</code> , <code>kAAX_Trace_Priority_High</code>
in	<i>iMessage</i>	Message string to log

Implements [AAX_IHostServices](#).

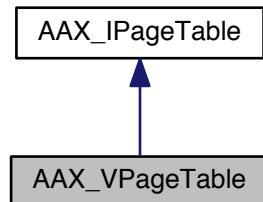
The documentation for this class was generated from the following file:

- [AAX_VHostServices.h](#)

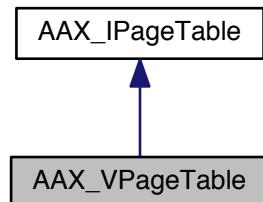
14.131 AAX_VPageTable Class Reference

```
#include <AAx_VPageTable.h>
```

Inheritance diagram for AAX_VPageTable:



Collaboration diagram for AAX_VPageTable:



14.131.1 Description

Version-managed concrete [AAX_IPageTable](#) class.

Public Member Functions

- [AAX_VPageTable \(IACFUnknown *pUnknown\)](#)
- virtual [~AAX_VPageTable \(\)](#)
- virtual [AAx_Result Clear \(\) AAx_OVERRIDE](#)
Clears all parameter mappings from the table.
- virtual [AAx_Result Empty \(AAx_CBoolean &oEmpty\) const AAx_OVERRIDE](#)
Indicates whether the table is empty.
- virtual [AAx_Result GetNumPages \(int32_t &oNumPages\) const AAx_OVERRIDE](#)
Get the number of pages currently in this table.
- virtual [AAx_Result InsertPage \(int32_t iPage\) AAx_OVERRIDE](#)
*Insert a new empty page before the page at index *iPage*.*
- virtual [AAx_Result RemovePage \(int32_t iPage\) AAx_OVERRIDE](#)

- Remove the page at index *iPage*.
 - virtual [AAX_Result GetNumMappedParameterIDs](#) (int32_t *iPage*, int32_t &*oNumParameterIdentifiers*) const [AAX_OVERRIDE](#)

Returns the total number of parameter IDs which are mapped to a page.
 - virtual [AAX_Result ClearMappedParameter](#) (int32_t *iPage*, int32_t *iIndex*) [AAX_OVERRIDE](#)

Clear the parameter at a particular index in this table.
 - virtual [AAX_Result GetMappedParameterID](#) (int32_t *iPage*, int32_t *iIndex*, [AAX_IString &oParameterIdentifier](#)) const [AAX_OVERRIDE](#)

Get the parameter identifier which is currently mapped to an index in this table.
 - virtual [AAX_Result MapParameterID](#) ([AAC_ParParamID](#) *iParameterIdentifier*, int32_t *iPage*, int32_t *iIndex*) [AAX_OVERRIDE](#)

Map a parameter to this table.
 - virtual [AAX_Result GetNumParametersWithNameVariations](#) (int32_t &*oNumParameterIdentifiers*) const [AAX_OVERRIDE](#)
 - virtual [AAX_Result GetNameVariationParameterIDAtIndex](#) (int32_t *iIndex*, [AAX_IString &oParameterIdentifier](#)) const [AAX_OVERRIDE](#)
 - virtual [AAX_Result GetNumNameVariationsForParameter](#) ([AAC_CPageTableParamID](#) *iParameterIdentifier*, int32_t &*oNumVariations*) const [AAX_OVERRIDE](#)
 - virtual [AAX_Result GetParameterNameVariationAtIndex](#) ([AAC_CPageTableParamID](#) *iParameterIdentifier*, int32_t *iIndex*, [AAX_IString &oNameVariation](#), int32_t &*oLength*) const [AAX_OVERRIDE](#)
 - virtual [AAX_Result GetParameterNameVariationOfLength](#) ([AAC_CPageTableParamID](#) *iParameterIdentifier*, int32_t *iLength*, [AAX_IString &oNameVariation](#)) const [AAX_OVERRIDE](#)
 - virtual [AAX_Result ClearParameterNameVariations](#) () [AAX_OVERRIDE](#)
 - virtual [AAX_Result ClearNameVariationsForParameter](#) ([AAC_CPageTableParamID](#) *iParameterIdentifier*) [AAX_OVERRIDE](#)
 - virtual [AAX_Result SetParameterNameVariation](#) ([AAC_CPageTableParamID](#) *iParameterIdentifier*, const [AAX_IString &iNameVariation](#), int32_t *iLength*) [AAX_OVERRIDE](#)
 - const [IACFUnknown * AsUnknown](#) () const
 - [IACFUnknown * AsUnknown](#) ()
 - bool [IsSupported](#) () const

14.131.2 Constructor & Destructor Documentation

14.131.2.1 [AAX_VPageTable::AAX_VPageTable](#) ([IACFUnknown * pUnknown](#))

14.131.2.2 virtual [AAX_VPageTable::~AAX_VPageTable](#) () [virtual]

14.131.3 Member Function Documentation

14.131.3.1 virtual [AAX_Result AAX_VPageTable::Clear](#) () [virtual]

Clears all parameter mappings from the table.

This method does not clear any parameter name variations from the table. For that, use [AAX_IPageTable::ClearNameVariations\(\)](#) or [AAX_IPageTable::ClearNameVariationsForParameter\(\)](#)

Implements [AAX_IPageTable](#).

14.131.3.2 virtual [AAX_Result AAX_VPageTable::Empty](#) ([AAC_CBoolean & oEmpty](#)) const [virtual]

Indicates whether the table is empty.

A table is empty if it contains no pages. A table which contains pages but no parameter assignments is not empty. A table which has associated parameter name variations but no pages is empty.

Parameters

out	<i>oEmpty</i>	true if this table is empty
-----	---------------	-----------------------------

Implements [AAX_IPageTable](#).

14.131.3.3 virtual AAX_Result AAX_VPageTable::GetNumPages (int32_t & *oNumPages*) const [virtual]

Get the number of pages currently in this table.

Parameters

out	<i>oNumPages</i>	The number of pages which are present in the page table. Some pages might not contain any parameter assignments.
-----	------------------	--

Implements [AAX_IPageTable](#).

14.131.3.4 virtual AAX_Result AAX_VPageTable::InsertPage (int32_t *iPage*) [virtual]

Insert a new empty page before the page at index *iPage*.

Returns

[AAX_ERROR_INVALID_ARGUMENT](#) if *iPage* is greater than the total number of pages

Parameters

in	<i>iPage</i>	The insertion point page index
----	--------------	--------------------------------

Implements [AAX_IPageTable](#).

14.131.3.5 virtual AAX_Result AAX_VPageTable::RemovePage (int32_t *iPage*) [virtual]

Remove the page at index *iPage*.

Returns

[AAX_ERROR_INVALID_ARGUMENT](#) if *iPage* is greater than the index of the last existing page

Parameters

in	<i>iPage</i>	The target page index
----	--------------	-----------------------

Implements [AAX_IPageTable](#).

14.131.3.6 virtual AAX_Result AAX_VPageTable::GetNumMappedParameterIDs (int32_t *iPage*, int32_t & *oNumParameterIdentifiers*) const [virtual]

Returns the total number of parameter IDs which are mapped to a page.

Note

The number of mapped parameter IDs does not correspond to the actual slot indices of the parameter assignments. For example, a page could have three total parameter assignments with parameters mapped to slots 2, 4, and 6.

Returns

[AAX_ERROR_INVALID_ARGUMENT](#) if *iPage* is greater than the index of the last existing page

Parameters

in	<i>iPage</i>	The target page index
out	<i>oNum</i> ↪ <i>Parameter</i> ↪ <i>Identifiers</i>	The number of parameter identifiers which are mapped to the target page

Implements [AAX_IPageTable](#).

14.131.3.7 virtual **AAX_Result** AAX_VPageTable::ClearMappedParameter (**int32_t** *iPage*, **int32_t** *iIndex*) [virtual]

Clear the parameter at a particular index in this table.

Returns

[AAX_SUCCESS](#) even if no parameter was mapped at the given index (the index is still clear)

Parameters

in	<i>iPage</i>	The target page index
in	<i>iIndex</i>	The target parameter slot index within the target page

Implements [AAX_IPageTable](#).

14.131.3.8 virtual **AAX_Result** AAX_VPageTable::GetMappedParameterID (**int32_t** *iPage*, **int32_t** *iIndex*, **AAX_IString** & *oParameterIdentifier*) const [virtual]

Get the parameter identifier which is currently mapped to an index in this table.

Returns

[AAX_ERROR_INVALID_ARGUMENT](#) if no parameter is mapped at the specified page/index location

Parameters

in	<i>iPage</i>	The target page index
in	<i>iIndex</i>	The target parameter slot index within the target page
out	<i>oParameter</i> ↪ <i>Identifier</i>	The identifier used for the mapped parameter in the page table (may be parameter name or ID)

Implements [AAX_IPageTable](#).

14.131.3.9 virtual **AAX_Result** AAX_VPageTable::MapParameterID (**AAX_CParamID** *iParameterIdentifier*, **int32_t** *iPage*, **int32_t** *iIndex*) [virtual]

Map a parameter to this table.

If *iParameterIdentifier* is an empty string then the parameter assignment will be cleared

Returns

[AAX_ERROR_NULL_ARGUMENT](#) if *iParameterIdentifier* is null

[AAX_ERROR_INVALID_ARGUMENT](#) if *iPage* is greater than the index of the last existing page

[AAX_ERROR_INVALID_ARGUMENT](#) if *iIndex* is negative

Parameters

in	<i>iParameterIdentifier</i>	The identifier for the parameter which will be mapped
in	<i>iPage</i>	The target page index
in	<i>iIndex</i>	The target parameter slot index within the target page

Implements [AAX_IPageTable](#).

14.131.3.10 virtual AAX_Result AAX_VPageTable::GetNumParametersWithNameVariations (int32_t & oNumParameterIdentifiers) const [virtual]

Get the number of parameters with name variations defined for the current table type

Provides the number of parameters with `lt;ControlNameVariations`; which are explicitly defined for the current page table type.

Note

Normally parameter name variations are only used with the 'PgTL' table type

See also

- [AAX_IPageTable::GetNameVariationParameterIDAtIndex\(\)](#)

Parameters

out	<i>oNumParameterIdentifiers</i>	The number of parameters with name variations explicitly associated with the current table type.
-----	---------------------------------	--

Implements [AAX_IPageTable](#).

14.131.3.11 virtual AAX_Result AAX_VPageTable::GetNameVariationParameterIDAtIndex (int32_t iIndex, AAX_IString & oParameterIdentifier) const [virtual]

Get the identifier for a parameter with name variations defined for the current table type

Note

Normally parameter name variations are only used with the 'PgTL' table type

See also

- [AAX_IPageTable::GetNumParametersWithNameVariations\(\)](#)

Parameters

in	<i>iIndex</i>	The target parameter index within the list of parameters with explicit name variations defined for this table type.
out	<i>oParameterIdentifier</i>	The identifier used for the parameter in the page table name variations list (may be parameter name or ID)

Implements [AAX_IPageTable](#).

14.131.3.12 virtual AAX_Result AAX_VPageTable::GetNumNameVariationsForParameter (AAX_CPageTableParamID iParameterIdentifier, int32_t & oNumVariations) const [virtual]

Get the number of name variations defined for a parameter

Provides the number of `lt;ControlNameVariations`; which are explicitly defined for `iParameterIdentifier` for the current page table type. No fallback logic is used to resolve this to the list of variations which would actually be used for an attached control surface if no explicit variations are defined for the current table type.

Note

Normally parameter name variations are only used with the 'PgTL' table type

See also

- [AAX_IPageTable::GetParameterNameVariationAtIndex\(\)](#)

Returns

`AAX_SUCCESS` and provides zero to `oNumVariations` if `iParameterIdentifier` is not found

Parameters

in	<i>iParameterIdentifier</i>	The identifier for the parameter
out	<i>oNumVariations</i>	The number of name variations which are defined for this parameter and explicitly associated with the current table type.

Implements [AAX_IPageTable](#).

14.131.3.13 virtual AAX_Result AAX_VPageTable::GetParameterNameVariationAtIndex (AAX_CPageTableParamID *iParameterIdentifier*, int32_t *iIndex*, AAX_IString & *oNameVariation*, int32_t & *oLength*) const [virtual]

Get a parameter name variation from the page table

Only returns `lt;ControlNameVariations` which are explicitly defined for the current page table type. No fallback logic is used to resolve this to the abbreviation which would actually be shown on an attached control surface if no explicit variation is defined for the current table type.

Note

Normally parameter name variations are only used with the 'PgTL' table type

See also

- [AAX_IPageTable::GetNumNameVariationsForParameter\(\)](#)

See also

- [AAX_IPageTable::GetParameterNameVariationOfLength\(\)](#)

Returns

`AAX_ERROR_NO_ABBREVIATED_PARAMETER_NAME` if no suitable variation is defined for this table

`AAX_ERROR_ARGUMENT_OUT_OF_RANGE` if `iIndex` is out of range

Parameters

in	<i>iParameterIdentifier</i>	The identifier for the parameter
in	<i>iIndex</i>	Index of the name variation
out	<i>oNameVariation</i>	The name variation, if one is explicitly defined for this table type
out	<i>oLength</i>	The length value for this name variation. This corresponds to the variation's <code>sZ</code> attribute in the page table XML and may be different from the string length of <code>iNameVariation</code> .

Implements [AAX_IPageTable](#).

14.131.3.14 virtual AAX_Result AAX_VPageTable::GetParameterNameVariationOfLength (AAX_CPageTableParamID iParameterIdentifier, int32_t iLength, AAX_IString & oNameVariation) const [virtual]

Get a parameter name variation of a particular length from the page table

Only returns `lt;ControlNameVariations[lt;` which are explicitly defined of `iLength` for the current page table type. No fallback logic is used to resolve this to the abbreviation which would actually be shown on an attached control surface if no explicit variation is defined for the specified length or current table type.

Note

Normally parameter name variations are only used with the 'PgTL' table type

See also

- [AAX_IPageTable::GetParameterNameVariationAtIndex\(\)](#)

Returns

[AAX_ERROR_NO_ABBREVIATED_PARAMETER_NAME](#) if no suitable variation is defined for this table

Parameters

in	<i>iParameterIdentifier</i>	The identifier for the parameter
in	<i>iLength</i>	The variation length to check, i.e. the <code>sz</code> attribute for the name variation in the page table XML
out	<i>oNameVariation</i>	The name variation, if one is explicitly defined for this table type and <code>iLength</code>

Implements [AAX_IPageTable](#).

14.131.3.15 virtual AAX_Result AAX_VPageTable::ClearParameterNameVariations () [virtual]

Clears all name variations for the current page table type

Note

Normally parameter name variations are only used with the 'PgTL' table type

See also

- [AAX_IPageTable::Clear\(\)](#)
- [AAX_IPageTable::ClearNameVariationsForParameter\(\)](#)

Implements [AAX_IPageTable](#).

14.131.3.16 virtual AAX_Result AAX_VPageTable::ClearNameVariationsForParameter (AAX_CPageTableParamID iParameterIdentifier) [virtual]

Clears all name variations for a single parameter for the current page table type

Warning

This will invalidate the list of parameter name variations indices, i.e. the parameter identifier associated with each index by [AAX_IPageTable::GetNameVariationParameterIDAtIndex\(\)](#)

Note

Normally parameter name variations are only used with the 'PgTL' table type

See also

[AAX_IPageTable::Clear\(\)](#)
[AAX_IPageTable::ClearParameterNameVariations\(\)](#)

Returns

[AAX_SUCCESS](#) and provides zero to *oNumVariations* if *iParameterIdentifier* is not found

Parameters

in	<i>iParameterIdentifier</i>	The identifier for the parameter
----	-----------------------------	----------------------------------

Implements [AAX_IPageTable](#).

14.131.3.17 virtual AAX_Result AAX_VPageTable::SetParameterNameVariation ([AAX_CPageTableParamID](#) *iParameterIdentifier*, const [AAX_IString](#) & *iNameVariation*, int32_t *iLength*) [virtual]

Sets a name variation explicitly for the current page table type

This will add a new name variation or overwrite the existing name variation with the same length which is defined for the current table type.

Warning

If no name variation previously existed for this parameter then this will invalidate the list of parameter name variations indices, i.e. the parameter identifier associated with each index by [AAX_IPageTable::GetNameVariationParameterIDAtIndex\(\)](#)

Note

Normally parameter name variations are only used with the 'PgTL' table type

Returns

[AAX_ERROR_INVALID_ARGUMENT](#) if *iNameVariation* is empty or if *iLength* is less than zero

Parameters

in	<i>iParameterIdentifier</i>	The identifier for the parameter
in	<i>iNameVariation</i>	The new parameter name variation
in	<i>iLength</i>	The length value for this name variation. This corresponds to the variation's sz attribute in the page table XML and is not required to match the length of <i>iNameVariation</i> .

Implements [AAX_IPageTable](#).

14.131.3.18 `const IACFUnknown* AAX_VPageTable::AsUnknown() const [inline]`

Returns the latest supported versioned ACF interface (e.g. an [AAX_IACFPageTable](#)) which is wrapped by this [AAX_IPageTable](#)

14.131.3.19 `IACFUnknown* AAX_VPageTable::AsUnknown() [inline]`

Returns the latest supported versioned ACF interface (e.g. an [AAX_IACFPageTable](#)) which is wrapped by this [AAX_IPageTable](#)

14.131.3.20 `bool AAX_VPageTable::IsSupported() const [inline]`

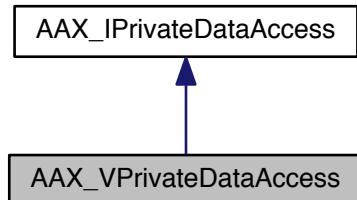
The documentation for this class was generated from the following file:

- [AAX_VPageTable.h](#)

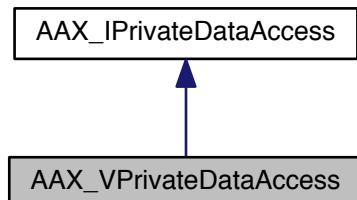
14.132 AAX_VPrivateDataAccess Class Reference

#include <AAX_VPrivateDataAccess.h>

Inheritance diagram for AAX_VPrivateDataAccess:



Collaboration diagram for AAX_VPrivateDataAccess:



14.132.1 Description

Version-managed concrete [AAX_IPrivateDataAccess](#) class.

Public Member Functions

- [AAX_VPrivateDataAccess \(IACFUnknown *pUnknown\)](#)
- virtual [~AAX_VPrivateDataAccess \(\)](#)
- virtual [AAX_Result ReadPortDirect \(AAX_CFieldIndex inFieldIndex, const uint32_t inOffset, const uint32_t inSize, void *outBuffer\) AAX_OVERRIDE](#)
Read data directly from DSP at the given port.
- virtual [AAX_Result WritePortDirect \(AAX_CFieldIndex inFieldIndex, const uint32_t inOffset, const uint32_t inSize, const void *inBuffer\) AAX_OVERRIDE](#)
Write data directly to DSP at the given port.

14.132.2 Constructor & Destructor Documentation

14.132.2.1 [AAX_VPrivateDataAccess::AAX_VPrivateDataAccess \(IACFUnknown * pUnknown \)](#)

14.132.2.2 virtual [AAX_VPrivateDataAccess::~AAX_VPrivateDataAccess \(\) \[virtual\]](#)

14.132.3 Member Function Documentation

14.132.3.1 virtual [AAX_Result AAX_VPrivateDataAccess::ReadPortDirect \(AAX_CFieldIndex inFieldIndex, const uint32_t inOffset, const uint32_t inSize, void * outBuffer \) \[virtual\]](#)

Read data directly from DSP at the given port.

Note

Blocking

Parameters

in	<i>inFieldIndex</i>	The port to read from.
in	<i>inOffset</i>	Offset into data to start reading.
in	<i>inSize</i>	Amount of data to read (in bytes).
out	<i>outBuffer</i>	Pointer to storage for data to be read into.

Implements [AAX_IPrivateDataAccess](#).

14.132.3.2 virtual [AAX_Result AAX_VPrivateDataAccess::WritePortDirect \(AAX_CFieldIndex inFieldIndex, const uint32_t inOffset, const uint32_t inSize, const void * inBuffer \) \[virtual\]](#)

Write data directly to DSP at the given port.

Note

Blocking

Parameters

in	<i>inFieldIndex</i>	The port to write to.
in	<i>inOffset</i>	Offset into data to begin writing.
in	<i>inSize</i>	Amount of data to write (in bytes).
in	<i>inBuffer</i>	Pointer to data being written.

Implements [AAX_IPrivateDataAccess](#).

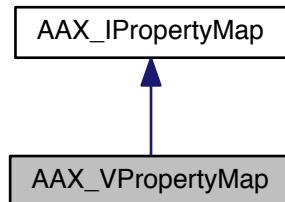
The documentation for this class was generated from the following file:

- [AAX_VPrivateDataAccess.h](#)

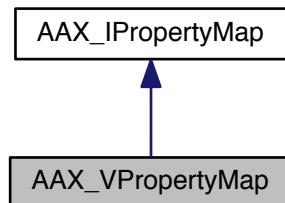
14.133 AAX_VPropertyMap Class Reference

```
#include <AAX_VPropertyMap.h>
```

Inheritance diagram for AAX_VPropertyMap:



Collaboration diagram for AAX_VPropertyMap:



14.133.1 Description

Version-managed concrete [AAX_IPropertyMap](#) class.

Public Member Functions

- virtual [~AAX_VPropertyMap](#) (void)

- virtual `AAX_CBoolean GetProperty (AAX_EProperty inProperty, AAX_CPropertyValue *outValue) const AAX_OVERRIDE`
Get a property value from a property map.
- virtual `AAX_CBoolean GetPointerProperty (AAX_EProperty inProperty, const void **outValue) const AAX_OVERRIDE`
Get a property value from a property map with a pointer-sized value.
- virtual `AAX_Result AddProperty (AAX_EProperty inProperty, AAX_CPropertyValue inValue) AAX_OVERRIDE`
Add a property to a property map.
- virtual `AAX_Result AddPointerProperty (AAX_EProperty inProperty, const void *inValue) AAX_OVERRIDE`
Add a property to a property map with a pointer-sized value.
- virtual `AAX_Result AddPointerProperty (AAX_EProperty inProperty, const char *inValue) AAX_OVERRIDE`
Add a property to a property map with a pointer-sized value.
- virtual `AAX_Result RemoveProperty (AAX_EProperty inProperty) AAX_OVERRIDE`
Remove a property from a property map.
- virtual `AAX_Result AddPropertyWithIDArray (AAX_EProperty inProperty, const AAX_SPlugInIdentifierTriad *inPluginIDs, uint32_t inNumPluginIDs) AAX_OVERRIDE`
Add an array of plug-in IDs to a property map.
- virtual `AAX_CBoolean GetPropertyWithIDArray (AAX_EProperty inProperty, const AAX_SPlugInIdentifierTriad **outPluginIDs, uint32_t *outNumPluginIDs) const AAX_OVERRIDE`
Get an array of plug-in IDs from a property map.
- virtual `IACFUnknown * GetIUnknown () AAX_OVERRIDE`

Static Public Member Functions

- static `AAX_VPropertyMap * Create (IACFUnknown *inComponentFactory)`
inComponentFactory must support IID_IACFComponentFactory - otherwise NULL is returned
- static `AAX_VPropertyMap * Acquire (IACFUnknown *inPropertyMapUnknown)`
inPropertyMapUnknown must support at least one AAX_IPROPERTYMAP interface - otherwise an AAX_VPROPERTYMAP object with no backing interface is returned

14.133.2 Constructor & Destructor Documentation

14.133.2.1 virtual `AAX_VPropertyMap::~AAX_VPropertyMap (void) [virtual]`

14.133.3 Member Function Documentation

14.133.3.1 static `AAX_VPropertyMap* AAX_VPropertyMap::Create (IACFUnknown * inComponentFactory) [static]`

`inComponentFactory` must support `IID_IACFComponentFactory` - otherwise NULL is returned

14.133.3.2 static `AAX_VPropertyMap* AAX_VPropertyMap::Acquire (IACFUnknown * inPropertyMapUnknown) [static]`

`inPropertyMapUnknown` must support at least one `AAX_IPROPERTYMAP` interface - otherwise an `AAX_VPROPERTYMAP` object with no backing interface is returned

14.133.3.3 virtual `AAX_CBoolean AAX_VPropertyMap::GetProperty (AAX_EProperty inProperty, AAX_CPropertyValue *outValue) const [virtual]`

Get a property value from a property map.

Returns true if the selected property is supported, false if it is not

Parameters

in	<i>inProperty</i>	The property ID
out	<i>outValue</i>	The property value

Implements [AAX_IPropertyMap](#).

14.133.3.4 virtual AAX_CBoolean AAX_VPropertyMap::GetPointerProperty (AAX_EProperty *inProperty*, const void ** *outValue*) const [virtual]

Get a property value from a property map with a pointer-sized value.

Returns true if the selected property is supported, false if it is not

Parameters

in	<i>inProperty</i>	The property ID
out	<i>outValue</i>	The property value

Implements [AAX_IPropertyMap](#).

14.133.3.5 virtual AAX_Result AAX_VPropertyMap::AddProperty (AAX_EProperty *inProperty*, AAX_CPropertyValue *inValue*) [virtual]

Add a property to a property map.

Note

This method may return an error if adding the property was unsuccessful. If there is a failure when adding a required property then registration of the relevant description element must be abandoned and the plug-in's description logic should proceed to the next element.

Parameters

in	<i>inProperty</i>	The property ID.
in	<i>inValue</i>	

Implements [AAX_IPropertyMap](#).

14.133.3.6 virtual AAX_Result AAX_VPropertyMap::AddPointerProperty (AAX_EProperty *inProperty*, const void * *inValue*) [virtual]

Add a property to a property map with a pointer-sized value.

Use this method to add properties which require a pointer-sized value. Do not use this method to add a property unless a pointer-sized value is explicitly specified in the property documentation.

Note

This method may return an error if adding the property was unsuccessful. If there is a failure when adding a required property then registration of the relevant description element must be abandoned and the plug-in's description logic should proceed to the next element.

Parameters

in	<i>inProperty</i>	The property ID.
in	<i>inValue</i>	

in	<i>inValue</i>	
----	----------------	--

Implements [AAX_IPropertyMap](#).

14.133.3.7 virtual AAX_Result AAX_VPropertyMap::AddPointerProperty (**AAX_EProperty** *inProperty*, const char * *inValue*) [virtual]

Add a property to a property map with a pointer-sized value.

Use this method to add properties which require a pointer-sized value. Do not use this method to add a property unless a pointer-sized value is explicitly specified in the property documentation.

Note

This method may return an error if adding the property was unsuccessful. If there is a failure when adding a required property then registration of the relevant description element must be abandoned and the plug-in's description logic should proceed to the next element.

Parameters

in	<i>inProperty</i>	The property ID.
in	<i>inValue</i>	

Implements [AAX_IPropertyMap](#).

14.133.3.8 virtual AAX_Result AAX_VPropertyMap::RemoveProperty (**AAX_EProperty** *inProperty*) [virtual]

Remove a property from a property map.

Parameters

in	<i>inProperty</i>	The property ID.
----	-------------------	------------------

Implements [AAX_IPropertyMap](#).

14.133.3.9 virtual AAX_Result AAX_VPropertyMap::AddPropertyWithIDArray (**AAX_EProperty** *inProperty*, const **AAX_SPlugInIdentifierTriad** * *inPluginIDs*, uint32_t *inNumPluginIDs*) [virtual]

Add an array of plug-in IDs to a property map.

Parameters

in	<i>inProperty</i>	The property ID.
in	<i>inPluginIDs</i>	An array of AAX_SPlugInIdentifierTriad
in	<i>inNumPluginIDs</i>	The length of <i>iPluginIDs</i>

Implements [AAX_IPropertyMap](#).

14.133.3.10 virtual AAX_CBoolean AAX_VPropertyMap::GetPropertyWithIDArray (**AAX_EProperty** *inProperty*, const **AAX_SPlugInIdentifierTriad** ** *outPluginIDs*, uint32_t * *outNumPluginIDs*) const [virtual]

Get an array of plug-in IDs from a property map.

Parameters

in	<i>inProperty</i>	The property ID.
----	-------------------	------------------

out	<i>outPluginIDs</i>	A pointer that will be set to reference an array of AAX_SPlugInIdentifierTriad
in	<i>outNumPluginIDs</i>	The length of <i>oPluginIDs</i>

Implements [AAX_IPropertyMap](#).

14.133.3.11 virtual IACFUnknown* AAX_VPropertyMap::GetIUnknown() [virtual]

Returns the most up-to-date underlying interface

Implements [AAX_IPropertyMap](#).

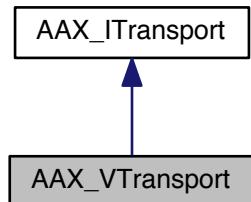
The documentation for this class was generated from the following file:

- [AAX_VPropertyMap.h](#)

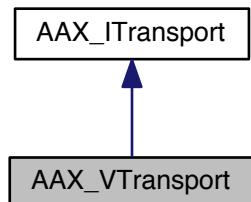
14.134 AAX_VTransport Class Reference

```
#include <AAX_VTransport.h>
```

Inheritance diagram for AAX_VTransport:



Collaboration diagram for AAX_VTransport:



14.134.1 Description

Version-managed concrete [AAX_ITransport](#) class.

Public Member Functions

- [AAX_VTransport \(IACFUnknown *pUnknown\)](#)
- [virtual ~AAX_VTransport \(\)](#)
- [virtual AAX_Result GetCurrentTempo \(double *TempoBPM\) const \[AAX_OVERRIDE\]](#)
CALL: Gets the current tempo.
- [virtual AAX_Result GetCurrentMeter \(int32_t *MeterNumerator, int32_t *MeterDenominator\) const \[AAX_OVERRIDE\]](#)
CALL: Gets the current meter.
- [virtual AAX_Result IsTransportPlaying \(bool *isPlaying\) const \[AAX_OVERRIDE\]](#)
CALL: Indicates whether or not the transport is playing back.
- [virtual AAX_Result GetCurrentTickPosition \(int64_t *TickPosition\) const \[AAX_OVERRIDE\]](#)
CALL: Gets the current tick position.
- [virtual AAX_Result GetCurrentLoopPosition \(bool *bLooping, int64_t *LoopStartTick, int64_t *LoopEndTick\) const \[AAX_OVERRIDE\]](#)
CALL: Gets current information on loop playback.
- [virtual AAX_Result GetCurrentNativeSampleLocation \(int64_t *SampleLocation\) const \[AAX_OVERRIDE\]](#)
CALL: Gets the current playback location of the native audio engine.
- [virtual AAX_Result GetCustomTickPosition \(int64_t *oTickPosition, int64_t iSampleLocation\) const \[AAX_OVERRIDE\]](#)
CALL: Given an absolute sample position, gets the corresponding tick position.
- [virtual AAX_Result GetBarBeatPosition \(int32_t *Bars, int32_t *Beats, int64_t *DisplayTicks, int64_t SampleLocation\) const \[AAX_OVERRIDE\]](#)
CALL: Given an absolute sample position, gets the corresponding bar and beat position.
- [virtual AAX_Result GetTicksPerQuarter \(uint32_t *ticks\) const \[AAX_OVERRIDE\]](#)
CALL: Retrieves the number of ticks per quarter note.
- [virtual AAX_Result GetCurrentTicksPerBeat \(uint32_t *ticks\) const \[AAX_OVERRIDE\]](#)
CALL: Retrieves the number of ticks per beat.
- [virtual AAX_Result GetTimelineSelectionStartPosition \(int64_t *oSampleLocation\) const \[AAX_OVERRIDE\]](#)
CALL: Retrieves the current absolute sample position of the beginning of the current transport selection.
- [virtual AAX_Result GetTimeCodeInfo \(AAX_EFrameRate *oFrameRate, int32_t *oOffset\) const \[AAX_OVERRIDE\]](#)
CALL: Retrieves the current time code frame rate and offset.
- [virtual AAX_Result GetFeetFramesInfo \(AAX_EFootFramesRate *oFootFramesRate, int64_t *oOffset\) const \[AAX_OVERRIDE\]](#)
CALL: Retrieves the current timecode feet/frames rate and offset.
- [virtual AAX_Result IsMetronomeEnabled \(int32_t *isEnabled\) const \[AAX_OVERRIDE\]](#)
Sets isEnabled to true if the metronome is enabled.

14.134.2 Constructor & Destructor Documentation

14.134.2.1 [AAX_VTransport::AAX_VTransport \(IACFUnknown * pUnknown \)](#)

14.134.2.2 [virtual AAX_VTransport::~AAX_VTransport \(\) \[virtual\]](#)

14.134.3 Member Function Documentation

14.134.3.1 [virtual AAX_Result AAX_VTransport::GetCurrentTempo \(double * TempoBPM \) const \[virtual\]](#)

CALL: Gets the current tempo.

Returns the tempo corresponding to the current position of the transport counter

Note

The resolution of the tempo returned here is based on the host's tempo resolution, so it will match the tempo displayed in the host. Use [GetCurrentTicksPerBeat\(\)](#) to calculate the tempo resolution note.

Parameters

<code>out</code>	<code>TempoBPM</code>	The current tempo in beats per minute
------------------	-----------------------	---------------------------------------

Implements [AAX_ITransport](#).

14.134.3.2 virtual AAX_Result AAX_VTransport::GetCurrentMeter (int32_t * MeterNumerator, int32_t * MeterDenominator) const [virtual]

CALL: Gets the current meter.

Returns the meter corresponding to the current position of the transport counter

Parameters

<code>out</code>	<code>MeterNumerator</code>	The numerator portion of the meter
<code>out</code>	<code>MeterDenominator</code>	The denominator portion of the meter

Implements [AAX_ITransport](#).

14.134.3.3 virtual AAX_Result AAX_VTransport::IsTransportPlaying (bool *.isPlaying) const [virtual]

CALL: Indicates whether or not the transport is playing back.

Parameters

<code>out</code>	<code>isplaying</code>	true if the transport is currently in playback
------------------	------------------------	--

Implements [AAX_ITransport](#).

14.134.3.4 virtual AAX_Result AAX_VTransport::GetCurrentTickPosition (int64_t * TickPosition) const [virtual]

CALL: Gets the current tick position.

Returns the current tick position corresponding to the current transport position. One "Tick" is represented here as 1/960000 of a quarter note. That is, there are 960,000 of these ticks in a quarter note.

Host Compatibility Notes The tick resolution here is different than that of the tick displayed in Pro Tools. "Display ticks" (as they are called) are 1/960 of a quarter note.

Parameters

<code>out</code>	<code>TickPosition</code>	The tick position value
------------------	---------------------------	-------------------------

Implements [AAX_ITransport](#).

14.134.3.5 virtual AAX_Result AAX_VTransport::GetCurrentLoopPosition (bool * bLooping, int64_t * LoopStartTick, int64_t * LoopEndTick) const [virtual]

CALL: Gets current information on loop playback.

Host Compatibility Notes This does not indicate anything about the status of the "Loop Record" option. Even when the host is configured to loop playback, looping may not occur if certain conditions are not met (i.e. the length of the selection is too short)

Parameters

out	<i>bLooping</i>	true if the host is configured to loop playback
out	<i>LoopStartTick</i>	The starting tick position of the selection being looped (see GetCurrentTickPosition())
out	<i>LoopEndTick</i>	The ending tick position of the selection being looped (see GetCurrentTickPosition())

Implements [AAX_ITransport](#).

14.134.3.6 virtual AAX_Result AAX_VTransport::GetCurrentNativeSampleLocation (int64_t * *SampleLocation*) const [virtual]

CALL: Gets the current playback location of the native audio engine.

When called from a ProcessProc render callback, this method will provide the absolute sample location at the beginning of the callback's audio buffers.

When called from [AAX_IEffectParameters::RenderAudio_Hybrid\(\)](#), this method will provide the absolute sample location for the samples in the method's **output** audio buffers. To calculate the absolute sample location for the samples in the method's input buffers (i.e. the timeline location where the samples originated) subtract the value provided by [AAX_IController::GetHybridSignalLatency\(\)](#) from this value.

When called from a non-real-time thread, this method will provide the current location of the samples being processed by the plug-in's ProcessProc on its real-time processing thread.

Note

This method only returns a value during playback. It cannot be used to determine, e.g., the location of the timeline selector while the host is not in playback.

Parameters

out	<i>SampleLocation</i>	Absolute sample location of the first sample in the current native processing buffer
-----	-----------------------	--

Implements [AAX_ITransport](#).

14.134.3.7 virtual AAX_Result AAX_VTransport::GetCustomTickPosition (int64_t * *oTickPosition*, int64_t *iSampleLocation*) const [virtual]

CALL: Given an absolute sample position, gets the corresponding tick position.

Host Compatibility Notes There is a minor performance cost associated with using this API in Pro Tools. It should not be used excessively without need.

Parameters

out	<i>oTickPosition</i>	the timeline tick position corresponding to <i>iSampleLocation</i>
in	<i>iSampleLocation</i>	An absolute sample location (see GetCurrentNativeSampleLocation())

Implements [AAX_ITransport](#).

14.134.3.8 virtual AAX_Result AAX_VTransport::GetBarBeatPosition (int32_t * *Bars*, int32_t * *Beats*, int64_t * *DisplayTicks*, int64_t *SampleLocation*) const [virtual]

CALL: Given an absolute sample position, gets the corresponding bar and beat position.

Host Compatibility Notes There is a minor performance cost associated with using this API in Pro Tools. It should not be used excessively without need.

Parameters

<i>out</i>	<i>Bars</i>	The bar corresponding to <i>SampleLocation</i>
<i>out</i>	<i>Beats</i>	The beat corresponding to <i>SampleLocation</i>
<i>out</i>	<i>DisplayTicks</i>	The ticks corresponding to <i>SampleLocation</i>
<i>in</i>	<i>SampleLocation</i>	An absolute sample location (see GetCurrentNativeSampleLocation())

Implements [AAX_ITransport](#).

14.134.3.9 virtual AAX_Result AAX_VTransport::GetTicksPerQuarter (uint32_t * *ticks*) const [virtual]

CALL: Retrieves the number of ticks per quarter note.

Parameters

<i>out</i>	<i>ticks</i>	
------------	--------------	--

Implements [AAX_ITransport](#).

14.134.3.10 virtual AAX_Result AAX_VTransport::GetCurrentTicksPerBeat (uint32_t * *ticks*) const [virtual]

CALL: Retrieves the number of ticks per beat.

Parameters

<i>out</i>	<i>ticks</i>	
------------	--------------	--

Implements [AAX_ITransport](#).

14.134.3.11 virtual AAX_Result AAX_VTransport::GetTimelineSelectionStartPosition (int64_t * *oSampleLocation*) const [virtual]

CALL: Retrieves the current absolute sample position of the beginning of the current transport selection.

Note

This method is part of the [version 2 transport interface](#)

Parameters

<i>out</i>	<i>oSampleLocation</i>	
------------	------------------------	--

Implements [AAX_ITransport](#).

14.134.3.12 virtual AAX_Result AAX_VTransport::GetTimeCodeInfo (AAX_EFrameRate * *oFrameRate*, int32_t * *oOffset*) const [virtual]

CALL: Retrieves the current time code frame rate and offset.

Note

This method is part of the [version 2 transport interface](#)

Parameters

out	<i>oFrameRate</i>	
out	<i>oOffset</i>	

Implements [AAX_ITransport](#).

14.134.3.13 virtual **AAX_Result** **AAX_VTransport::GetFeetFramesInfo** (**AAX_EFeetFramesRate** * *oFeetFramesRate*, **int64_t** * *oOffset*) const [virtual]

CALL: Retrieves the current timecode feet/frames rate and offset.

Note

This method is part of the [version 2 transport interface](#)

Parameters

out	<i>oFeetFramesRate</i>	
out	<i>oOffset</i>	

Implements [AAX_ITransport](#).

14.134.3.14 virtual **AAX_Result** **AAX_VTransport::IsMetronomeEnabled** (**int32_t** * *isEnabled*) const [virtual]

Sets *isEnabled* to true if the metronome is enabled.

Note

This method is part of the [version 2 transport interface](#)

Parameters

out	<i>isEnabled</i>	
-----	------------------	--

Implements [AAX_ITransport](#).

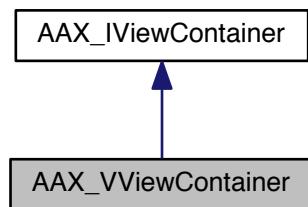
The documentation for this class was generated from the following file:

- [AAX_VTransport.h](#)

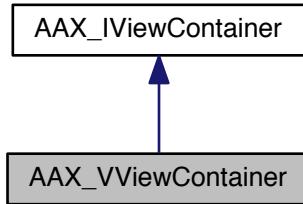
14.135 AAX_VViewContainer Class Reference

```
#include <AAX_VViewContainer.h>
```

Inheritance diagram for AAX_VViewContainer:



Collaboration diagram for AAX_VViewContainer:



14.135.1 Description

Version-managed concrete [AAX_IVViewContainer](#) class.

Public Member Functions

- **AAX_VViewContainer (IACFUnknown *pUnknown)**
- virtual **~AAX_VViewContainer ()**
- virtual int32_t **GetType () AAX_OVERRIDE**
Returns the raw view type as one of [AAX_EViewContainer_Type](#).
- virtual void * **GetPtr () AAX_OVERRIDE**
Returns a pointer to the raw view.
- virtual **AAX_Result GetModifiers (uint32_t *outModifiers) AAX_OVERRIDE**
Queries the host for the current modifier keys.
- virtual **AAX_Result SetViewSize (AAX_Point &inSize) AAX_OVERRIDE**
Request a change to the main view size.
- virtual **AAX_Result HandleParameterMouseDown (AAX_CParamID inParamID, uint32_t inModifiers) AAX_OVERRIDE**
Alert the host to a mouse down event.
- virtual **AAX_Result HandleParameterMouseDrag (AAX_CParamID inParamID, uint32_t inModifiers) AAX_OVERRIDE**
Alert the host to a mouse drag event.
- virtual **AAX_Result HandleParameterMouseUp (AAX_CParamID inParamID, uint32_t inModifiers) AAX_OVERRIDE**
Alert the host to a mouse up event.
- virtual **AAX_Result HandleMultipleParametersMouseDown (const AAX_CParamID *inParamIDs, uint32_t inNumOfParams, uint32_t inModifiers) AAX_OVERRIDE**
Alert the host to a mouse down event.
- virtual **AAX_Result HandleMultipleParametersMouseDrag (const AAX_CParamID *inParamIDs, uint32_t inNumOfParams, uint32_t inModifiers) AAX_OVERRIDE**
Alert the host to a mouse drag event.
- virtual **AAX_Result HandleMultipleParametersMouseUp (const AAX_CParamID *inParamIDs, uint32_t inNumOfParams, uint32_t inModifiers) AAX_OVERRIDE**
Alert the host to a mouse up event.

14.135.2 Constructor & Destructor Documentation

14.135.2.1 `AAX_VViewContainer::AAX_VViewContainer (IACFUnknown * pUnknown)`

14.135.2.2 `virtual AAX_VViewContainer::~AAX_VViewContainer () [virtual]`

14.135.3 Member Function Documentation

14.135.3.1 `virtual int32_t AAX_VViewContainer::GetType () [virtual]`

Returns the raw view type as one of [AAX_EViewContainer_Type](#).

Implements [AAX_IViewContainer](#).

14.135.3.2 `virtual void* AAX_VViewContainer::GetPtr () [virtual]`

Returns a pointer to the raw view.

Implements [AAX_IViewContainer](#).

14.135.3.3 `virtual AAX_Result AAX_VViewContainer::GetModifiers (uint32_t * outModifiers) [virtual]`

Queries the host for the current [modifier keys](#).

This method returns a bit mask with bits set for each of the currently active modifier keys. This method does not return the state of the [AAX_eModifiers_SecondaryButton](#).

Host Compatibility Notes Although this method allows plug-ins to acquire the current state of the Windows key (normally blocked by Pro Tools), plug-ins should not use key combinations that require this key.

Parameters

<code>out</code>	<code>outModifiers</code>	Current modifiers as a bitmask of AAX_EModifiers
------------------	---------------------------	--

Implements [AAX_IViewContainer](#).

14.135.3.4 `virtual AAX_Result AAX_VViewContainer::SetViewSize (AAX_Point & inSize) [virtual]`

Request a change to the main view size.

Note

- For compatibility with the smallest supported displays, plug-in GUI dimensions should not exceed 749x617 pixels, or 749x565 pixels for plug-ins with sidechain support.

Parameters

<code>in</code>	<code>inSize</code>	The new size to which the plug-in view should be set
-----------------	---------------------	--

Implements [AAX_IViewContainer](#).

14.135.3.5 `virtual AAX_Result AAX_VViewContainer::HandleParameterMouseDown (AAX_CParamID inParamID, uint32_t inModifiers) [virtual]`

Alert the host to a mouse down event.

Parameters

in	<i>inParamID</i>	ID of the parameter whose control is being edited
in	<i>inModifiers</i>	A bitmask of AAX_EModifiers values

Implements [AAX_IViewContainer](#).

14.135.3.6 virtual AAX_Result AAX_VViewContainer::HandleParameterMouseDrag (AAX_CParamID *inParamID*, uint32_t *inModifiers*) [virtual]

Alert the host to a mouse drag event.

Warning

The host may return [AAX_ERROR_UNIMPLEMENTED](#) for this event even if the host did handle the corresponding mouse down event. A plug-in should ignore any following mouse drag and mouse up events that correspond to a host-managed mouse down event. ([PTSW-195209 / PT-218474](#))

Parameters

in	<i>inParamID</i>	ID of the parameter whose control is being edited
in	<i>inModifiers</i>	A bitmask of AAX_EModifiers values

Implements [AAX_IViewContainer](#).

14.135.3.7 virtual AAX_Result AAX_VViewContainer::HandleParameterMouseUp (AAX_CParamID *inParamID*, uint32_t *inModifiers*) [virtual]

Alert the host to a mouse up event.

Warning

The host may return [AAX_ERROR_UNIMPLEMENTED](#) for this event even if the host did handle the corresponding mouse down event. A plug-in should ignore any following mouse drag and mouse up events that correspond to a host-managed mouse down event. ([PTSW-195209 / PT-218474](#))

Parameters

in	<i>inParamID</i>	ID of the parameter whose control is being edited
in	<i>inModifiers</i>	A bitmask of AAX_EModifiers values

Implements [AAX_IViewContainer](#).

14.135.3.8 virtual AAX_Result AAX_VViewContainer::HandleMultipleParametersMouseDown (const AAX_CParamID * *inParamIDs*, uint32_t *inNumOfParams*, uint32_t *inModifiers*) [virtual]

Alert the host to a mouse down event.

Parameters

in	<i>inParamIDs</i>	IDs of the parameters that belong to the same GUI element whose controls are being edited
in	<i>inNumOfParams</i>	Number of parameter IDs
in	<i>inModifiers</i>	A bitmask of AAX_EModifiers values

Implements [AAX_IViewContainer](#).

**14.135.3.9 virtual AAX_Result AAX_VViewContainer::HandleMultipleParametersMouseDown (const AAX_CParamID *
inParamIDs, uint32_t inNumOfParams, uint32_t inModifiers) [virtual]**

Alert the host to a mouse drag event.

Warning

The host may return [AAX_ERROR_UNIMPLEMENTED](#) for this event even if the host did handle the corresponding mouse down event. A plug-in should ignore any following mouse drag and mouse up events that correspond to a host-managed mouse down event. ([PTSW-195209 / PT-218474](#))

Parameters

in	<i>inParamIDs</i>	IDs of the parameters that belong to the same GUI element whose controls are being edited
in	<i>inNumOfParams</i>	Number of parameter IDs
in	<i>inModifiers</i>	A bitmask of AAX_EModifiers values

Implements [AAX_IViewContainer](#).

**14.135.3.10 virtual AAX_Result AAX_VViewContainer::HandleMultipleParametersMouseUp (const AAX_CParamID *
inParamIDs, uint32_t inNumOfParams, uint32_t inModifiers) [virtual]**

Alert the host to a mouse up event.

Warning

The host may return [AAX_ERROR_UNIMPLEMENTED](#) for this event even if the host did handle the corresponding mouse down event. A plug-in should ignore any following mouse drag and mouse up events that correspond to a host-managed mouse down event. ([PTSW-195209 / PT-218474](#))

Parameters

in	<i>inParamIDs</i>	IDs of the parameters that belong to the same GUI element whose controls are being edited
in	<i>inNumOfParams</i>	Number of parameter IDs
in	<i>inModifiers</i>	A bitmask of AAX_EModifiers values

Implements [AAX_IViewContainer](#).

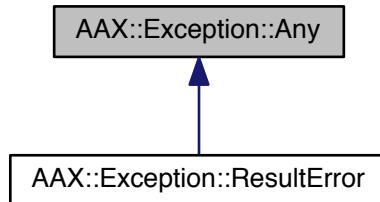
The documentation for this class was generated from the following file:

- [AAX_VViewContainer.h](#)

14.136 AAX::Exception::Any Class Reference

```
#include <AAX_Exception.h>
```

Inheritance diagram for AAX::Exception::Any:



14.136.1 Description

Base class for AAX exceptions

This class is defined within the AAX Library and is always handled within the AAX plug-in. Objects of this class are never passed between the plug-in and the AAX host.

The definition of this class may change between versions of the AAX SDK. This class does not include any form of version safety for cross-version compatibility.

Warning

Do not use multiple inheritance in any sub-classes within the [AAX::Exception::Any](#) inheritance tree
Never pass exceptions across the library boundary to the AAX host

Public Member Functions

- virtual [~Any \(\)](#)
- template<class C > [Any \(const C &inWhat\)](#)
- template<class C1 , class C2 , class C3 > [Any \(const C1 &inWhat, const C2 &inFunction, const C3 &inLine\)](#)
- [Any & operator= \(const Any &inOther\)](#)
- [AAX_DEFAULT_MOVE_CTOR \(Any\)](#)
- [AAX_DEFAULT_MOVE_OPER \(Any\)](#)
- const std::string & [What \(\) const](#)
- const std::string & [Desc \(\) const](#)
- const std::string & [Function \(\) const](#)
- const std::string & [Line \(\) const](#)

14.136.2 Constructor & Destructor Documentation

14.136.2.1 virtual AAX::Exception::Any::~Any () [inline], [virtual]

14.136.2.2 template<class C > AAX::Exception::Any::Any (const C & inWhat) [inline], [explicit]

Explicit conversion from a string-like object

14.136.2.3 template<class C1 , class C2 , class C3 > AAX::Exception::Any::Any (const C1 & inWhat, const C2 & inFunction, const C3 & inLine) [inline], [explicit]

Explicit conversion from a string-like object with function name and line number

14.136.3 Member Function Documentation

14.136.3.1 `Any& AAX::Exception::Any::operator= (const Any & inOther) [inline]`

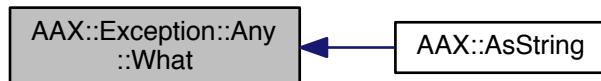
14.136.3.2 `AAX::Exception::Any::AAX_DEFAULT_MOVECTOR (Any)`

14.136.3.3 `AAX::Exception::Any::AAX_DEFAULT_MOVE_OPER (Any)`

14.136.3.4 `const std::string& AAX::Exception::Any::What () const [inline]`

Referenced by `AAX::AsString()`.

Here is the caller graph for this function:



14.136.3.5 `const std::string& AAX::Exception::Any::Desc () const [inline]`

14.136.3.6 `const std::string& AAX::Exception::Any::Function () const [inline]`

14.136.3.7 `const std::string& AAX::Exception::Any::Line () const [inline]`

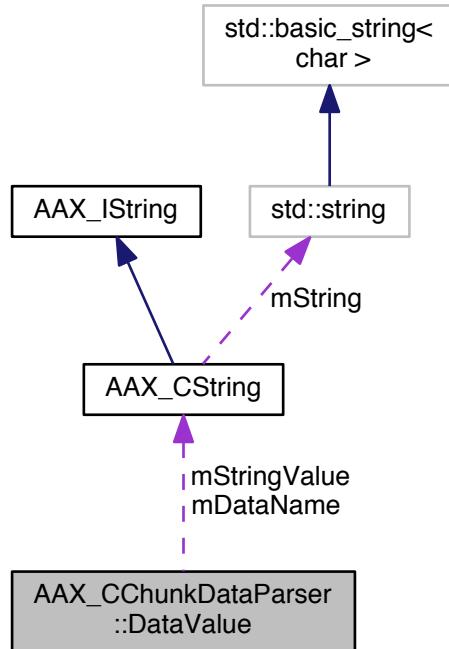
The documentation for this class was generated from the following file:

- [AAX_Exception.h](#)

14.137 AAX_CChunkDataParser::DataValue Struct Reference

```
#include <AAX_CChunkDataParser.h>
```

Collaboration diagram for AAX_CChunkDataParser::DataValue:



Public Member Functions

- [DataValue \(\)](#)

Public Attributes

- `int32_t mDataType`
- [AAX_CString mDataName](#)
name of the stored data
- `int64_t mIntValue`
used if this `DataValue` is not a string
- [AAX_CString mStringValue](#)
used if this `DataValue` is a string

14.137.1 Constructor & Destructor Documentation

14.137.1.1 `AAX_CChunkDataParser::DataValue::DataValue()` [inline]

14.137.2 Member Data Documentation

14.137.2.1 `int32_t AAX_CChunkDataParser::DataValue::mDataType`

14.137.2.2 AAX_CString AAX_CChunkDataParser::DataValue::mDataName

name of the stored data

14.137.2.3 int64_t AAX_CChunkDataParser::DataValue::mIntValue

used if this [DataValue](#) is not a string

14.137.2.4 AAX_CString AAX_CChunkDataParser::DataValue::mStringValue

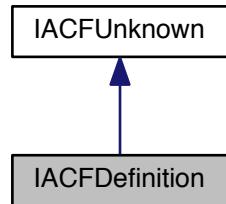
used if this [DataValue](#) is a string

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

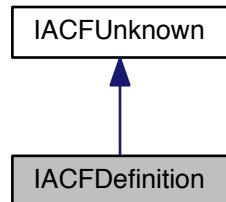
- [AAX_CChunkDataParser.h](#)

14.138 IACFDefinition Interface Reference

Inheritance diagram for IACFDefinition:



Collaboration diagram for IACFDefinition:



14.138.1 Description

Publicly inherits from IACFUnknown. This abstract interface is used to identify all of the plug-in components in the host.

Remarks

This interface is the base class for both plug-in and component definitions. All defined attributes are read only.

Note

This interface does not provide any attribute enumeration. You must know the uid of the associated with the attribute that you need to find.

This interface is implemented by the host. The plug-in will use this interface to define optional attributes for both plug-in and component implementations classes.

Public Member Functions

- virtual ACFRESULT ACFMETHODCALLTYPE [DefineAttribute](#) (const `acfUID &attributeID`, const `acfUID &typeID`, const void *`attrData`, `acfUInt32 attrDataSize`)=0

Add a read only attribute to the definition.
- virtual ACFRESULT ACFMETHODCALLTYPE [GetAttributeInfo](#) (const `acfUID &attributeID`, `acfUID *typeID`, `acfUInt32 *attrDataSize`)=0

Returns information about the given attribute.
- virtual ACFRESULT ACFMETHODCALLTYPE [CopyAttribute](#) (const `acfUID &attributeID`, const `acfUID &typeID`, void *`attrData`, `acfUInt32 attrDataSize`)=0

Copy the a given attribute.

14.138.2 Member Function Documentation

14.138.2.1 virtual ACFRESULT ACFMETHODCALLTYPE IACFDefinition::DefineAttribute (const acfUID & attributeID, const acfUID & typeID, const void * attrData, acfUInt32 attrDataSize) [pure virtual]

Add a read only attribute to the definition.

DefineAttribute

Remarks

Use the method to define additional global attributes for your component. This method will fail if the attribute has already been defined.

Parameters

<code>attributeID</code>	Unique identifier for attribute
<code>typeID</code>	Indicates the type of the attribute data
<code>attrData</code>	Pointer to buffer that contains the attribute data
<code>attrDataSize</code>	Size of the attribute buffer

14.138.2.2 virtual ACFRESULT ACFMETHODCALLTYPE IACFDefinition::GetAttributeInfo (const acfUID & attributeID, acfUID * typeID, acfUInt32 * attrDataSize) [pure virtual]

Returns information about the given attribute.

Remarks

Use this method to retrieve the type and size of a given attribute.

Parameters

<i>attributeID</i>	Unique identifier for attribute
<i>typeID</i>	Indicates the type of the attribute data
<i>attrDataSize</i>	Size of the attribute data

14.138.2.3 virtual ACFRESULT ACFMETHODCALLTYPE IACFDefinition::CopyAttribute (const acfUID & *attributeID*, const acfUID & *typeID*, void * *attrData*, acfUInt32 *attrDataSize*) [pure virtual]

Copy the a given attribute.

CopyAttribute

Remarks

Use this method to access the contents of a given attribute.

Parameters

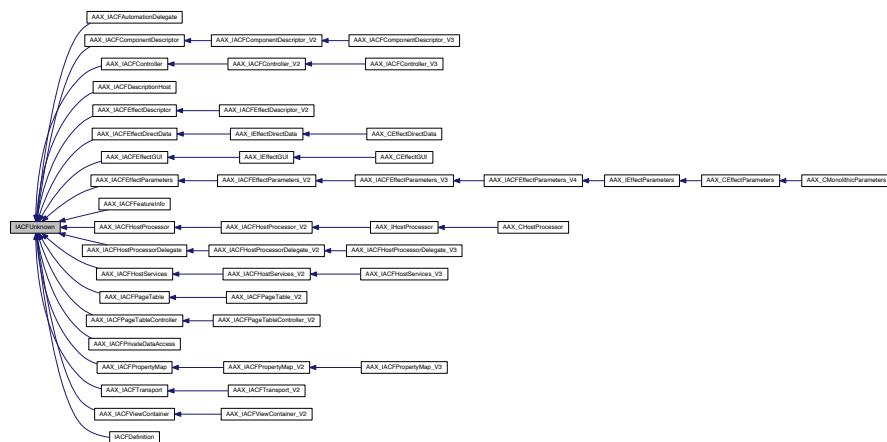
<i>attributeID</i>	Unique identifier for attribute
<i>typeID</i>	Indicates the type of the attribute data
<i>attrData</i>	Pointer to buffer to copy the attribute data
<i>attrDataSize</i>	Size of the attribute buffer

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

- [AAX_ACFInterface.dox](#)

14.139 IACFUnknown Interface Reference

Inheritance diagram for IACFUnknown:



14.139.1 Description

COM compatible IUnknown C++ interface.

Remarks

The methods of the [IACFUnknown](#) interface, implemented by all ACF objects, supports general inter-object protocol negotiation via the `QueryInterface` method, and object lifetime management with the `AddRef` and `Release` methods.

Note

Because `AddRef` and `Release` are not required to return accurate values, callers of these methods must not use the return values to determine if an object is still valid or has been destroyed. (Standard Microsoft disclaimer)

For further information please refer to the Microsoft documentation for `IUnknown`.

Note

This class will work only with compilers that can produce COM-compatible object layouts for C++ classes. egcs can not do this. Metrowerks can do this (if you subclass from `__comobject`).

Public Member Functions

- virtual BEGIN_ACFINTERFACE ACFRESULT ACFMETHODCALLTYPE [QueryInterface](#) (const acfIID &*iid*, void ***ppOut*)=0
Returns pointers to supported interfaces.
- virtual acfUInt32 ACFMETHODCALLTYPE [AddRef](#) (void)=0
Increments reference count.
- virtual acfUInt32 ACFMETHODCALLTYPE [Release](#) (void)=0
Decrements reference count.

14.139.2 Member Function Documentation**14.139.2.1 virtual BEGIN_ACFINTERFACE ACFRESULT ACFMETHODCALLTYPE IACFUnknown::QueryInterface (const acfIID & *iid*, void ** *ppOut*) [pure virtual]**

Returns pointers to supported interfaces.

Remarks

The `QueryInterface` method gives a client access to alternate interfaces implemented by an object. The returned interface pointer will have already had its reference count incremented so the caller will be required to call the `Release` method.

Parameters

<i>iid</i>	Identifier of the requested interface
<i>ppOut</i>	Address of variable that receives the interface pointer associated with <i>iid</i> .

14.139.2.2 virtual acfUInt32 ACFMETHODCALLTYPE IACFUnknown::AddRef (void) [pure virtual]

Increments reference count.

Remarks

The `AddRef` method should be called every time a new copy of an interface is made. When this copy is no longer referenced it must be released with the `Release` method.

14.139.2.3 virtual acfUInt32 ACFMETHODCALLTYPE IACFUnknown::Release(void) [pure virtual]

Decrements reference count.

Remarks

Use this method to decrement the reference count. When the reference count reaches zero the object that implements the interface will be deleted.

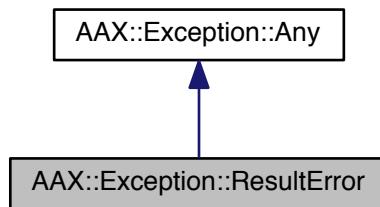
The documentation for this interface was generated from the following file:

- [AAX_ACFInterface.dox](#)

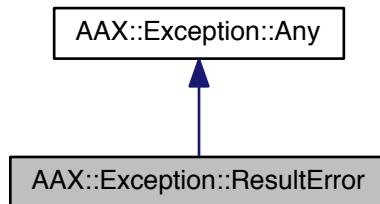
14.140 AAX::Exception::ResultError Class Reference

```
#include <AAX_Exception.h>
```

Inheritance diagram for AAX::Exception::ResultError:



Collaboration diagram for AAX::Exception::ResultError:



14.140.1 Description

Exception class for [AAX_EError](#) results

Public Member Functions

- [ResultError \(AAx_Result inWhatResult\)](#)
- template<class C> [ResultError \(AAx_Result inWhatResult, const C &inFunction\)](#)
- template<class C1, class C2> [ResultError \(AAx_Result inWhatResult, const C1 &inFunction, const C2 &inLine\)](#)
- [AAx_Result Result \(\) const](#)

Static Public Member Functions

- static std::string [FormatResult \(AAx_Result inResult\)](#)

14.140.2 Constructor & Destructor Documentation

14.140.2.1 [AAx::Exception::ResultError::ResultError \(AAX_Result inWhatResult \) \[inline\], \[explicit\]](#)

14.140.2.2 [template<class C> AAx::Exception::ResultError::ResultError \(AAX_Result inWhatResult, const C &inFunction \) \[inline\], \[explicit\]](#)

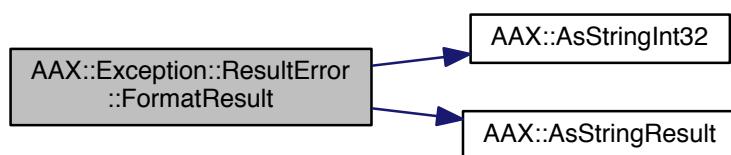
14.140.2.3 [template<class C1, class C2> AAx::Exception::ResultError::ResultError \(AAX_Result inWhatResult, const C1 &inFunction, const C2 &inLine \) \[inline\], \[explicit\]](#)

14.140.3 Member Function Documentation

14.140.3.1 [static std::string AAx::Exception::ResultError::FormatResult \(AAX_Result inResult \) \[inline\], \[static\]](#)

References AAx::AsStringInt32(), and AAx::AsStringResult().

Here is the call graph for this function:



14.140.3.2 [AAx_Result AAx::Exception::ResultError::Result \(\) const \[inline\]](#)

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

- [AAx_Exception.h](#)

14.141 SAutoArray< T > Struct Template Reference

Public Member Functions

- [SAutoArray \(\)](#)

- [~SAutoArray \(\)](#)
- void [Reset \(T *inData\)](#)
- T * [Get \(\)](#)

14.141.1 Constructor & Destructor Documentation

14.141.1.1 template<typename T> SAutoArray< T >::SAutoArray() [inline]

14.141.1.2 template<typename T> SAutoArray< T >::~SAutoArray() [inline]

References SAutoArray< T >::Reset().

Here is the call graph for this function:



14.141.2 Member Function Documentation

14.141.2.1 template<typename T> void SAutoArray< T >::Reset(T * inData) [inline]

Referenced by SAutoArray< T >::~SAutoArray().

Here is the caller graph for this function:



14.141.2.2 template<typename T> T* SAutoArray< T >::Get() [inline]

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

- [AAX_MIDILogging.cpp](#)

Chapter 15

File Documentation

15.1 AAX.h File Reference

```
#include <stdint.h>#include <stddef.h>#include "AAX_Version.h"#include "AAX_Enums.h"#include "AAX_Errors.h"
```

```
#include "AAX_Properties.h"
#include "AAX_Push2ByteStructAlignment.h"
#include "AAX_PopStructAlignment.h"
```

15.1.1 Description

Various utility definitions for AAX.

Classes

- struct [AAX_SPlugInChunkHeader](#)
Plug-in chunk header.
- struct [AAX_SPlugInChunk](#)
Plug-in chunk header + data.
- struct [AAX_SPlugInIdentifierTriad](#)
Plug-in Identifier Triad.
- struct [AAX_CMidiPacket](#)
Packet structure for MIDI data.
- struct [AAX_CMidiStream](#)
MIDI stream data structure used by [AAX_IMIDINode](#).

Macros

- #define [AAX_CALLBACK](#)
- #define [AAX_PREPROCESSOR_CONCAT_HELPER](#)(X, Y) X ## Y
- #define [AAX_PREPROCESSOR_CONCAT](#)(X, Y) [AAX_PREPROCESSOR_CONCAT_HELPER](#)(X,Y)
- #define [AAX_FIELD_INDEX](#)(aContextType, aMember) (([AAX_CFieldIndex](#)) (offsetof (aContextType, a← Member) / sizeof (void *)))
Compute the index used to address a context field.

C++ compiler macros

- #define [TL_VERSION](#) 0
Preprocessor flag indicating compilation for TI.
- #define [AAX_CPP11_SUPPORT](#) 1
Preprocessor toggle for code which requires C++11 compiler support.

C++ keyword macros

Use these macros for keywords which may not be supported on all compilers

Warning

Be careful when using these macros; they are a workaround and the fallback versions of the macros are not guaranteed to provide identical behavior to the fully-supported versions. Always consider the code which will be generated in each case!

If your code is protected with PACE Fusion and you are using a PACE SDK prior to v4 then you must explicitly define [AAX_CPP11_SUPPORT](#) 0 in your project's preprocessor settings to avoid encountering source failover caused by AAX header includes with exotic syntax.

- #define [AAX_OVERRIDE](#) override
override keyword macro
- #define [AAX_FINAL](#) final
final keyword macro
- #define [AAX_DEFAULTCTOR](#)(X) X() = default
default keyword macro for a class default constructor

- `#define AAX_DEFAULT_COPY_CTOR(X) X(const X&) = default`
default keyword macro for a class copy constructor
- `#define AAX_DEFAULT_ASgn_OPER(X) X& operator=(const X&) = default`
default keyword macro for a class assignment operator
- `#define AAX_DELETE(X) X = delete`
delete keyword macro
- `#define AAX_DEFAULT_MOVE_CTOR(X) X(X&&) = default`
default keyword macro for a class move constructor
- `#define AAX_DEFAULT_MOVE_OPER(X) X& operator=(X&&) = default`
default keyword macro for a class move-assignment operator
- `#define AAX_CONSTEXPR constexpr`
constexpr keyword macro

Pointer definitions

- `#define AAXPointer_32bit 1`
When AAX_PointerSize == AAXPointer_32bit this is a 32-bit build.
- `#define AAXPointer_64bit 2`
When AAX_PointerSize == AAXPointer_64bit this is a 64-bit build.
- `#define AAX_PointerSize AAXPointer_32bit`
Use this definition to check the pointer size in the current build.

Alignment macros

Use these macros to define struct packing alignment for data structures that will be sent across binary or platform boundaries.

```
#include AAX_ALIGN_FILE_HOST
// Structure definition
#include AAX_ALIGN_FILE_RESET
```

See the documentation for each macro for individual usage notes and warnings

- `#define AAX_ALIGN_FILE_HOST "AAX_Push2ByteStructAlignment.h"`
Macro to set alignment for data structures that are shared with the host.
- `#define AAX_ALIGN_FILE_ALG "AAX_Push8ByteStructAlignment.h"`
Macro to set alignment for data structures that are used in the alg.
- `#define AAX_ALIGN_FILE_RESET "AAX_PopStructAlignment.h"`
Macro to reset alignment back to default.

Typedefs

- `typedef int32_t AAX_CIndex`
- `typedef AAX_CIndex AAX_CCount`
- `typedef uint8_t AAX_CBoolean`
Cross-compiler boolean type used by AAX interfaces.
- `typedef uint32_t AAX_CSelector`
- `typedef int64_t AAX_CTimestamp`
Time stamp value. Measured against the DAE clock (see [AAX_IComponentDescriptor::AddClock\(\)](#))
- `typedef int64_t AAX_CTimeOfDay`
Hardware running clock value. MIDI packet time stamps are measured against this clock. This is actually the same as TransportCounter, but kept for compatibility.
- `typedef int64_t AAX_CTransportCounter`
Offset of samples from transport start. Same as TimeOfDay, but added for new interfaces as TimeOfDay is a confusing name.
- `typedef float AAX_CSAMPLERate`
Literal sample rate value used by the [sample rate field](#). For [AAX_eProperty_SampleRate](#), use a mask of [AAX_E_SampleRateMask](#).

- **typedef uint32_t AAX_CTypeID**
Matches type of OSType used in classic plugins.
- **typedef int32_t AAX_Result**
- **typedef int32_t AAX_CPropertyValue**
32-bit property values
- **typedef int64_t AAX_CPropertyValue64**
64-bit property values
- **typedef AAX_CPropertyValue AAX_CPointerPropertyValue**
Pointer-sized property values.
- **typedef int32_t AAX_CTargetPlatform**
Matches type of target platform.
- **typedef AAX_CIndex AAX_CFieldIndex**
Not used by AAX plug-ins (except in `AAX_FIELD_INDEX` macro)
- **typedef AAX_CSelector AAX_CComponentID**
- **typedef AAX_CSelector AAX_CMeterID**
- **typedef const char * AAX_CParamID**
Parameter identifier.
- **typedef AAX_CParamID AAX_CPageTableParamID**
Parameter identifier used in a page table.
- **typedef const char * AAX_CEffectID**
URL-style Effect identifier. Must be unique among all registered effects in the collection.
- **typedef _acfUID acfUID**
- **typedef acfUID AAX_Feature_UID**
- **typedef const float *const * AAX_CAudioInPort**
AAX algorithm audio input port data type
- **typedef float *const * AAX_CAudioOutPort**
AAX algorithm audio output port data type
- **typedef float *const AAX_CMeterPort**
AAX algorithm meter port data type
- **typedef struct AAX_SPlugInChunkHeader AAX_SPlugInChunkHeader**
- **typedef struct AAX_SPlugInChunk AAX_SPlugInChunk**
- **typedef struct AAX_SPlugInChunk * AAX_SPlugInChunkPtr**
- **typedef struct AAX_SPlugInIdentifierTriad AAX_SPlugInIdentifierTriad**
- **typedef struct AAX_SPlugInIdentifierTriad * AAX_SPlugInIdentifierTriadPtr**

Functions

- **AAX_CBoolean sampleRateInMask (AAX_CSampleRate inSR, uint32_t iMask)**
Determines whether a particular `AAX_CSampleRate` is present in a given mask of `AAX_ESampleRateMask`.
- **AAX_CSampleRate getLowestSampleRateInMask (uint32_t iMask)**
Converts from a mask of `AAX_ESampleRateMask` to the lowest supported `AAX_CSampleRate` value in Hz.
- **uint32_t getMaskForSampleRate (float inSR)**
Returns the `AAX_ESampleRateMask` selector for a literal sample rate.

15.1.2 Macro Definition Documentation

15.1.2.1 #define TI_VERSION 0

Preprocessor flag indicating compilation for TI.

15.1.2.2 #define AAX_CPP11_SUPPORT 1

Preprocessor toggle for code which requires C++11 compiler support.

15.1.2.3 #define AAX_OVERRIDE override

override keyword macro

15.1.2.4 #define AAX_FINAL final

final keyword macro

15.1.2.5 #define AAX_DEFAULT_CTOR(X) X() = default

default keyword macro for a class default constructor

15.1.2.6 #define AAX_DEFAULT_COPY_CTOR(X) X(const X&) = default

default keyword macro for a class copy constructor

15.1.2.7 #define AAX_DEFAULT_ASGN_OPER(X) X& operator=(const X&) = default

default keyword macro for a class assignment operator

15.1.2.8 #define AAX_DELETE(X) X = delete

delete keyword macro

Warning

The non-C++11 version of this macro assumes public declaration access

15.1.2.9 #define AAX_DEFAULT_MOVE_CTOR(X) X(X&&) = default

default keyword macro for a class move constructor

15.1.2.10 #define AAX_DEFAULT_MOVE_OPER(X) X& operator=(X&&) = default

default keyword macro for a class move-assignment operator

15.1.2.11 #define AAX_CONSTEXPR constexpr

constexpr keyword macro

15.1.2.12 #define AAXPointer_32bit 1

When AAX_PointerSize == AAXPointer_32bit this is a 32-bit build.

15.1.2.13 #define AAXPointer_64bit 2

When AAX_PointerSize == AAXPointer_64bit this is a 64-bit build.

15.1.2.14 #define AAX_PointerSize AAXPointer_32bit

Use this definition to check the pointer size in the current build.

See also

[AAXPointer_32bit](#)
[AAXPointer_64bit](#)

15.1.2.15 #define AAX_ALIGN_FILE_HOST "AAX_Push2ByteStructAlignment.h"

Macro to set alignment for data structures that are shared with the host.

This macro is used to set alignment for data structures that are part of the AAX ABI. You should not need to use this macro for any custom data structures in your plug-in.

15.1.2.16 #define AAX_ALIGN_FILE_ALG "AAX_Push8ByteStructAlignment.h"

Macro to set alignment for data structures that are used in the alg.

IMPORTANT: Be very careful to maintain correct data alignment when sending data structures between platforms.

Warning

- This macro does not guarantee data alignment compatibility for data structures which include base classes/structs or virtual functions. The MSVC, GCC and LLVM/clang, and CCS (TI) compilers do not support data structure cross-compatibility for these types of structures. clang will now present a warning when these macros are used on any such structures: `#pragma ms_struct` can not be used with dynamic classes or structures
- Struct Member Alignment (/Zp) on Microsoft compilers must be set to a minimum of 8-byte packing in order for this macro to function properly. For more information, see this MSDN article:
<http://msdn.microsoft.com/en-us/library/ms253935.aspx>

15.1.2.17 #define AAX_ALIGN_FILE_RESET "AAX_PopStructAlignment.h"

Macro to reset alignment back to default.

15.1.2.18 #define AAX_CALLBACK**15.1.2.19 #define AAX_PREPROCESSOR_CONCAT_HELPER(X, Y) X ## Y****15.1.2.20 #define AAX_PREPROCESSOR_CONCAT(X, Y) AAX_PREPROCESSOR_CONCAT_HELPER(X,Y)****15.1.2.21 #define AAX_FIELD_INDEX(aContextType, aMember) ((AAX_CFieldIndex)(offsetof(aContextType,aMember) / sizeof(void *)))**

Compute the index used to address a context field.

This macro expands to a constant expression suitable for use in enumerator definitions and case labels so `int32_t` as `aMember` is a constant specifier.

Parameters

in	<i>aContextType</i>	The name of context type
in	<i>aMember</i>	The name or other specifier of a field of that context type

Referenced by `AAX_CMonolithicParameters::GenerateCoefficients()`, `AAX_CMonolithicParameters::ResetFieldData()`, and `AAX_CMonolithicParameters::StaticDescribe()`.

15.1.3 Typedef Documentation

15.1.3.1 `typedef int32_t AAX_CIndex`

Todo Not used by AAX plug-ins (except as [AAX_CFieldIndex](#))

15.1.3.2 `typedef AAX_CIndex AAX_CCount`

Todo Not used by AAX plug-ins

15.1.3.3 `typedef uint8_t AAX_CBoolean`

Cross-compiler boolean type used by AAX interfaces.

15.1.3.4 `typedef uint32_t AAX_CSelector`

Todo Clean up usage; currently used for a variety of ID-related values

15.1.3.5 `typedef int64_t AAX_CTimestamp`

Time stamp value. Measured against the DAE clock (see [AAX_IComponentDescriptor::AddClock\(\)](#))

15.1.3.6 `typedef int64_t AAX_CTimeOfDay`

Hardware running clock value. MIDI packet time stamps are measured against this clock. This is actually the same as `TransportCounter`, but kept for compatibility.

15.1.3.7 `typedef int64_t AAX_CTransportCounter`

Offset of samples from transport start. Same as `TimeOfDay`, but added for new interfaces as `TimeOfDay` is a confusing name.

15.1.3.8 `typedef float AAX_CSAMPLERate`

Literal sample rate value used by the [sample rate field](#). For `AAX_eProperty_SampleRate`, use a mask of [AAX_E_SampleRateMask](#).

See also

[sampleRateInMask](#)

15.1.3.9 `typedef uint32_t AAX_CTypeID`

Matches type of OSType used in classic plugins.

15.1.3.10 `typedef int32_t AAX_Result`**15.1.3.11 `typedef int32_t AAX_CPropertyValue`**

32-bit property values

Use this property value type for all properties unless otherwise specified by the property documentation

15.1.3.12 `typedef int64_t AAX_CPropertyValue64`

64-bit property values

Do not use this value type unless specified explicitly in the property documentation

15.1.3.13 `typedef AAX_CPropertyValue AAX_CPointerPropertyValue`

Pointer-sized property values.

Do not use this value type unless specified explicitly in the property documentation

15.1.3.14 `typedef int32_t AAX_CTargetPlatform`

Matches type of [target platform](#).

15.1.3.15 `typedef AAX_CIndex AAX_CFieldIndex`

Not used by AAX plug-ins (except in [AAX_FIELD_INDEX](#) macro)

15.1.3.16 `typedef AAX_CSelector AAX_CComponentID`

Todo Not used by AAX plug-ins

15.1.3.17 `typedef AAX_CSelector AAX_CMeterID`

Todo Not used by AAX plug-ins

15.1.3.18 `typedef const char* AAX_CParamID`

Parameter identifier.

Note

While this is a string, it must be less than 32 characters in length. (strlen of 31 or less)

15.1.3.19 `typedef AAX_CParamID AAX_CPageTableParamID`

Parameter identifier used in a page table.

May be a parameter ID or a parameter name string depending on the page table formatting. Must be less than 32 characters in length (strlen of 31 or less.)

See also

[Parameter identifiers in the Page Table Guide](#)

15.1.3.20 `typedef const char* AAX_CEffectID`

URL-style Effect identifier. Must be unique among all registered effects in the collection.

15.1.3.21 `typedef _acfUID acfUID`**15.1.3.22 `typedef acfUID AAX_Feature_UID`**

Identifier for AAX features

See [AAX_IDescriptionHost::AcquireFeatureProperties\(\)](#) and [AAX_IFeatureInfo](#)

15.1.3.23 `typedef const float* const* AAX_CAudioInPort`

AAX algorithm audio input port data type

Audio input ports are provided with a pointer to an array of const audio buffers, with one buffer provided per input or side chain channel.

Todo Not used directly by AAX plug-ins

15.1.3.24 `typedef float* const* AAX_CAudioOutPort`

AAX algorithm audio output port data type

Audio output ports are provided with a pointer to an array of audio buffers, with one buffer provided per output or auxiliary output channel.

Todo Not used directly by AAX plug-ins

15.1.3.25 `typedef float* const AAX_CMeterPort`

AAX algorithm meter port data type

Meter output ports are provided with a pointer to an array of floats, with one float provided per meter tap. The algorithm is responsible for setting these to the corresponding per-buffer peak sample values.

Todo Not used directly by AAX plug-ins

15.1.3.26 `typedef struct AAX_SPlugInChunkHeader AAX_SPlugInChunkHeader`

15.1.3.27 `typedef struct AAX_SPlugInChunk AAX_SPlugInChunk`

15.1.3.28 `typedef struct AAX_SPlugInChunk * AAX_SPlugInChunkPtr`

15.1.3.29 `typedef struct AAX_SPlugInIdentifierTriad AAX_SPlugInIdentifierTriad`

15.1.3.30 `typedef struct AAX_SPlugInIdentifierTriad * AAX_SPlugInIdentifierTriadPtr`

15.1.4 Function Documentation

15.1.4.1 `AAX_CBoolean sampleRateInMask(AAX_CSampleRate inSR, uint32_t iMask) [inline]`

Determines whether a particular `AAX_CSampleRate` is present in a given mask of `AAX_ESampleRateMask`.

See also

`kAAX_Property_SampleRate`

References `AAX_eSampleRateMask_176400`, `AAX_eSampleRateMask_192000`, `AAX_eSampleRateMask_44100`, `AAX_eSampleRateMask_48000`, `AAX_eSampleRateMask_88200`, and `AAX_eSampleRateMask_96000`.

15.1.4.2 `AAX_CSampleRate getLowestSampleRateInMask(uint32_t iMask) [inline]`

Converts from a mask of `AAX_ESampleRateMask` to the lowest supported `AAX_CSampleRate` value in Hz.

References `AAX_eSampleRateMask_176400`, `AAX_eSampleRateMask_192000`, `AAX_eSampleRateMask_44100`, `AAX_eSampleRateMask_48000`, `AAX_eSampleRateMask_88200`, and `AAX_eSampleRateMask_96000`.

15.1.4.3 `uint32_t getMaskForSampleRate(float inSR) [inline]`

Returns the `AAX_ESampleRateMask` selector for a literal sample rate.

The given rate must be an exact match with one of the available selectors. If no exact match is found then `AAX_eSampleRateMask_No` is returned.

References `AAX_eSampleRateMask_176400`, `AAX_eSampleRateMask_192000`, `AAX_eSampleRateMask_44100`, `AAX_eSampleRateMask_48000`, `AAX_eSampleRateMask_88200`, `AAX_eSampleRateMask_96000`, and `AAX_eSampleRateMask_No`.

15.2 AAX_ACFInterface.dox File Reference

Classes

- `struct _acfUID`
- `interface IACFUnknown`

COM compatible IUnknown C++ interface.

- `interface IACFDefinition`

Publicly inherits from IACFUnknown. This abstract interface is used to identify all of the plug-in components in the host.

Typedefs

- `typedef struct _acfUID acfUID`

- **GUID compatible structure for ACF.**
- **typedef acfUID acfIID**
IID compatible structure for ACF.

15.2.1 TYPEDef Documentation

15.2.1.1 acfUID

GUID compatible structure for ACF.

15.2.1.2 acfIID

IID compatible structure for ACF.

15.3 AAX_AdditionalFeatures_Algorithm.dox File Reference

15.4 AAX_AdditionalFeatures_AOSandSidechain.dox File Reference

15.5 AAX_AdditionalFeatures_CurveDisplays.dox File Reference

15.6 AAX_AdditionalFeatures_Hybrid.dox File Reference

15.7 AAX_AdditionalFeatures_Meters.dox File Reference

15.8 AAX_AdditionalFeatures_MIDI.dox File Reference

15.9 AAX_Alignment.h File Reference

```
#include <stddef.h>
```

15.9.1 Description

Alignment malloc and free methods for optimization.

Namespaces

- [AAX](#)

Functions

- `void AAX::alignFree (void *p)`
- `template<class T > T * AAX::alignMalloc (int iArraySize, int iAlignment)`

15.10 AAX_Assert.h File Reference

```
#include "AAX.Enums.h" #include "AAX_CHostServices.h"
```

15.10.1 Description

Declarations for cross-platform AAX_ASSERT, AAX_TRACE and related facilities.

- [AAX_ASSERT\(condition \)](#) - If the condition is `false` triggers some manner of warning, e.g. a dialog in a developer build or a DigiTrace log in a shipping build. May be used on host or TI.
- [AAX_DEBUGASSERT\(condition \)](#) - Variant of [AAX_ASSERT](#) which is only active in debug builds of the plug-in.
- [AAX_TRACE_RELEASE\(iPriority, iMessageStr \[,params...\] \)](#) - Traces a printf- style message to the DigiTrace log file. Enabled using the `DTF_AAXPLUGINS` DigiTrace facility.
- [AAX_TRACE\(iPriority, iMessageStr \[, params...\] \)](#) - Variant of [AAX_TRACE_RELEASE](#) which only emits logs in debug builds of the plug-in.
- [AAX_STACKTRACE_RELEASE\(iPriority, iMessageStr \[,params...\] \)](#) - Similar to [AAX_TRACE_RELEASE](#) but prints a stack trace as well as a log message
- [AAX_STACKTRACE\(iPriority, iMessageStr \[,params...\] \)](#) - Variant of [AAX_STACKTRACE_RELEASE](#) which only emits logs in debug builds of the plug-in.
- [AAX_TRACEORSTACKTRACE_RELEASE\(iTracePriority, iStackTracePriority, iMessageStr \[,params...\] \)](#) - Combination of [AAX_TRACE_RELEASE](#) and [AAX_STACKTRACE_RELEASE](#); a stack trace is emitted if logging is enabled at `iStackTracePriority`. Otherwise, if logging is enabled at `iTracePriority` then emits a log.

For all trace macros:

`inPriority` is one of

- [kAAX_Trace_Priority_Low](#)
- [kAAX_Trace_Priority_Normal](#)
- [kAAX_Trace_Priority_High](#)

These correspond to how the trace messages are filtered using [DigiTrace](#).

Note

Disabling the `DTF_AAXPLUGINS` facility will slightly reduce the overhead of trace statements and chip communication on HDX systems.

Macros

- `#define kAAX_Trace_Priority_None AAX_eTracePriorityHost_None`
- `#define kAAX_Trace_Priority_High AAX_eTracePriorityHost_High`
- `#define kAAX_Trace_Priority_Normal AAX_eTracePriorityHost_Normal`
- `#define kAAX_Trace_Priority_Low AAX_eTracePriorityHost_Low`
- `#define kAAX_Trace_Priority_Lowest AAX_eTracePriorityHost_Lowest`
- `#define AAX_TRACE_RELEASE(iPriority, ...)`

Print a trace statement to the log.

- `#define AAX_STACKTRACE_RELEASE(iPriority, ...)`

Print a stack trace statement to the log.

- `#define AAX_TRACEORSTACKTRACE_RELEASE(iTracePriority, iStackTracePriority, ...)`

Print a trace statement with an optional stack trace to the log.

- `#define AAX_ASSERT(condition)`
Asserts that a condition is true and logs an error if the condition is false.
- `#define AAX_DEBUGASSERT(condition) do { ; } while (0)`
Asserts that a condition is true and logs an error if the condition is false (debug plug-in builds only)
- `#define AAX_TRACE(iPriority, ...)` `do { ; } while (0)`
Print a trace statement to the log (debug plug-in builds only)
- `#define AAX_STACKTRACE(iPriority, ...)` `do { ; } while (0)`
Print a stack trace statement to the log (debug builds only)
- `#define AAX_TRACEORSTACKTRACE(iTracePriority, iStackTracePriority, ...)` `do { ; } while (0)`
Print a trace statement with an optional stack trace to the log (debug builds only)

Typedefs

- `typedef AAX_ETracePriorityHost AAX_ETracePriority`

15.10.2 Macro Definition Documentation

15.10.2.1 `#define kAAX_Trace_Priority_None AAX_eTracePriorityHost_None`

15.10.2.2 `#define kAAX_Trace_Priority_High AAX_eTracePriorityHost_High`

15.10.2.3 `#define kAAX_Trace_Priority_Normal AAX_eTracePriorityHost_Normal`

15.10.2.4 `#define kAAX_Trace_Priority_Low AAX_eTracePriorityHost_Low`

15.10.2.5 `#define kAAX_Trace_Priority_Lowest AAX_eTracePriorityHost_Lowest`

15.10.2.6 `#define AAX_TRACE_RELEASE(iPriority, ...)`

Value:

```
{ \
    AAX_CHostServices::Trace ( iPriority, __VA_ARGS__ ); \
};
```

Print a trace statement to the log.

Use this macro to print a trace statement to the log file. This macro will be included in all builds of the plug-in.

Notes

- This macro is compatible with host host and embedded (AAX DSP) environments
- Subject to a total line limit of 256 chars

Usage Each invocation of this macro takes a trace priority and a printf-style logging string.

Because output from this macro will be enabled on end users' systems under certain tracing configurations, logs should always be formatted with some standard information to avoid confusion between logs from different plug-ins. This is the recommended formatting for AAX_TRACE_RELEASE logs:

[Manufacturer name] [Plug-in name] [Plug-in version] [logging text (indented)]

For example:

```
1 AAX_TRACE_RELEASE(kAAX_Trace_Priority_Normal, "%s %s %s;\tMy float: %f, My C-string: %s",
2     "MyCompany", "MyPlugIn", "1.0.2", myFloat, myCString);
```

See also

[DigiTrace Guide](#)

15.10.2.7 #define AAX_STACKTRACE_RELEASE(*iPriority*, ...)

Value:

```
{ \
    AAX_CHostServices::StackTrace ( iPriority, iPriority,
__VA_ARGS__ ); \
};
```

Print a stack trace statement to the log.

See also

[AAX_TRACE_RELEASE](#)

15.10.2.8 #define AAX_TRACEORSTACKTRACE_RELEASE(*iTracePriority*, *iStackTracePriority*, ...)

Value:

```
{ \
    AAX_CHostServices::StackTrace ( iTracePriority,
iStackTracePriority, __VA_ARGS__ ); \
};
```

Print a trace statement with an optional stack trace to the log.

Parameters

in	<i>iTracePriority</i>	The log priority at which the trace statement will be printed
in	<i>iStackTracePriority</i>	The log priority at which the stack trace will be printed

See also

[AAX_TRACE_RELEASE](#)

15.10.2.9 #define AAX_ASSERT(*condition*)

Value:

```
{ \
    if( ! ( condition ) ) { \
        AAX_CHostServices::HandleAssertFailure(
__FILE__, __LINE__, #condition, (int32_t)AAX_eAssertFlags_Log ); \
    } \
};
```

Asserts that a condition is true and logs an error if the condition is false.

Notes

- This macro will be compiled out of release builds.
- This macro is compatible with host and embedded ([AAX](#) DSP) environments.

Usage Each invocation of this macro takes a single argument, which is interpreted as a `bool`.

```
1 AAX_ASSERT(desiredValue == variableUnderTest);
```

Referenced by AAX_CMonolithicParameters::AddSynchronizedParameter(), AAX_CMonolithicParameters::GenerateCoefficients(), AAX::GetCStringOfLength(), AAX_CMonolithicParameters::ResetFieldData(), AAX_CParameter< T >::SetNumberOfSteps(), AAX_CMonolithicParameters::StaticDescribe(), AAX_CMonolithicParameters::StaticRenderAudio(), and AAX::String2Binary().

15.10.2.10 #define AAX_DEBUGASSERT(*condition*) do { ; } while (0)

Asserts that a condition is true and logs an error if the condition is false (debug plug-in builds only)

See also

[AAX_ASSERT](#)

15.10.2.11 #define AAX_TRACE(*iPriority*, ...) do { ; } while (0)

Print a trace statement to the log (debug plug-in builds only)

Use this macro to print a trace statement to the log file from debug builds of a plug-in.

Notes

- This macro will be compiled out of release builds
- This macro is compatible with host and embedded ([AAX](#) DSP) environments
- Subject to a total line limit of 256 chars

Usage Each invocation of this macro takes a trace priority and a printf-style logging string. For example:

```
1 AAX_TRACE(kAAX_Trace_Priority_Normal, "My float: %f, My C-string: %s", myFloat, myCString);
```

See also

[DigiTrace Guide](#)

15.10.2.12 #define AAX_STACKTRACE(*iPriority*, ...) do { ; } while (0)

Print a stack trace statement to the log (debug builds only)

See also

[AAX_TRACE](#)

15.10.2.13 #define AAX_TRACEORSTACKTRACE(*iTracePriority*, *iStackTracePriority*, ...) do { ; } while (0)

Print a trace statement with an optional stack trace to the log (debug builds only)

Parameters

in	<i>iTracePriority</i>	The log priority at which the trace statement will be printed
in	<i>iStackTracePriority</i>	The log priority at which the stack trace will be printed

See also[AAX_TRACE](#)**15.10.3 Typedef Documentation****15.10.3.1 `typedef AAX_ETracePriorityHost AAX_ETracePriority`****15.11 AAX_Atomic.h File Reference**

```
#include "AAX.h" #include <stdint.h>
```

15.11.1 Description

Atomic operation utilities.

Macros

- `#define _AAX_ATOMIC_H_`

Functions

- `uint32_t AAX_CALLBACK AAX_Atomic_IncThenGet_32 (register uint32_t &ioData)`
Increments a 32-bit value and returns the result.
- `uint32_t AAX_CALLBACK AAX_Atomic_DecThenGet_32 (register uint32_t &ioData)`
Decrements a 32-bit value and returns the result.
- `uint32_t AAX_CALLBACK AAX_Atomic_Exchange_32 (volatile uint32_t &ioValue, uint32_t inExchangeValue)`
Return the original value of ioValue and then set it to inExchangeValue.
- `uint64_t AAX_CALLBACK AAX_Atomic_Exchange_64 (volatile uint64_t &ioValue, uint64_t inExchangeValue)`
Return the original value of ioValue and then set it to inExchangeValue.
- `template<typename TPointer > TPointer *AAX_CALLBACK AAX_Atomic_Exchange_Pointer (TPointer *&ioValue, TPointer *inExchangeValue)`
Perform an exchange operation on a pointer value.
- `bool AAX_CALLBACK AAX_Atomic_CompareAndExchange_32 (volatile uint32_t &ioValue, uint32_t inCompareValue, uint32_t inExchangeValue)`
Perform a compare and exchange operation on a 32-bit value.
- `bool AAX_CALLBACK AAX_Atomic_CompareAndExchange_64 (volatile uint64_t &ioValue, uint64_t inCompareValue, uint64_t inExchangeValue)`
Perform a compare and exchange operation on a 64-bit value.
- `template<typename TPointer > bool AAX_CALLBACK AAX_Atomic_CompareAndExchange_Pointer (TPointer *&ioValue, TPointer *inCompareValue, TPointer *inExchangeValue)`
Perform a compare and exchange operation on a pointer value.
- `template<typename TPointer > TPointer *AAX_CALLBACK AAX_Atomic_Load_Pointer (TPointer const *const volatile *inValue)`
Atomically loads a pointer value.

15.11.2 Macro Definition Documentation

15.11.2.1 `#define _AAX_ATOMIC_H_`

15.11.3 Function Documentation

15.11.3.1 `uint32_t AAX_CALLBACK AAX_Atomic_IncThenGet_32(register uint32_t & ioData)`

Increments a 32-bit value and returns the result.

15.11.3.2 `uint32_t AAX_CALLBACK AAX_Atomic_DecThenGet_32(register uint32_t & ioData)`

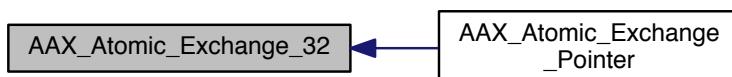
Decrements a 32-bit value and returns the result.

15.11.3.3 `uint32_t AAX_CALLBACK AAX_Atomic_Exchange_32(volatile uint32_t & ioValue, uint32_t inExchangeValue)`

Return the original value of ioValue and then set it to inExchangeValue.

Referenced by `AAX_Atomic_Exchange_Pointer()`.

Here is the caller graph for this function:

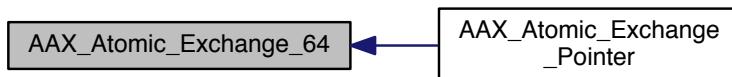


15.11.3.4 `uint64_t AAX_CALLBACK AAX_Atomic_Exchange_64(volatile uint64_t & ioValue, uint64_t inExchangeValue)`

Return the original value of ioValue and then set it to inExchangeValue.

Referenced by `AAX_Atomic_Exchange_Pointer()`.

Here is the caller graph for this function:

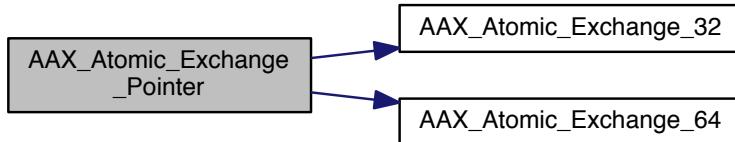


15.11.3.5 `template<typename TPointer > TPointer* AAX_CALLBACK AAX_Atomic_Exchange_Pointer(TPointer *& ioValue, TPointer * inExchangeValue)`

Perform an exchange operation on a pointer value.

References AAX_Atomic_Exchange_32(), and AAX_Atomic_Exchange_64().

Here is the call graph for this function:

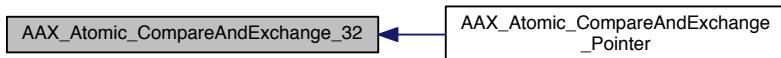


15.11.3.6 `bool AAX_CALLBACK AAX_Atomic_CompareAndExchange_32 (volatile uint32_t & ioValue, uint32_t inCompareValue, uint32_t inExchangeValue)`

Perform a compare and exchange operation on a 32-bit value.

Referenced by AAX_Atomic_CompareAndExchange_Pointer().

Here is the caller graph for this function:

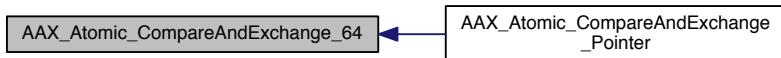


15.11.3.7 `bool AAX_CALLBACK AAX_Atomic_CompareAndExchange_64 (volatile uint64_t & ioValue, uint64_t inCompareValue, uint64_t inExchangeValue)`

Perform a compare and exchange operation on a 64-bit value.

Referenced by AAX_Atomic_CompareAndExchange_Pointer().

Here is the caller graph for this function:

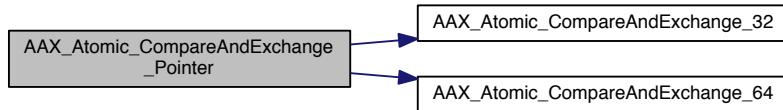


15.11.3.8 `template<typename TPointer> bool AAX_CALLBACK AAX_Atomic_CompareAndExchange_Pointer (TPointer *& ioValue, TPointer * inCompareValue, TPointer * inExchangeValue)`

Perform a compare and exchange operation on a pointer value.

References AAX_Atomic_CompareAndExchange_32(), and AAX_Atomic_CompareAndExchange_64().

Here is the call graph for this function:



15.11.3.9 template<typename TPointer > TPointer* AAX_CALLBACK AAX_Atomic_Load_Pointer (TPointer const *const volatile * *inValue*)

Atomically loads a pointer value.

15.12 AAX_AuxInterface_DirectData.dox File Reference

15.13 AAX_AuxInterface_HostProcessor.dox File Reference

15.14 AAX_BugList.dox File Reference

15.15 AAX_Callbacks.h File Reference

```
#include "AAX.h"
```

15.15.1 Description

AAX callback prototypes and ProcPtr definitions

Classes

- class [AAX_Component< aContextType >](#)
Empty class containing type declarations for the AAX algorithm and associated callbacks.

TypeDefs

- typedef IACFUnknown *AAX_CALLBACK * AAXCreateObjectProc(void)
A user-defined callback that AAX calls to process data packets and/or audio.
- typedef AAX_Component< void >::CProcessProc [AAX_CProcessProc](#)
Used by AAX_SchedulePacket()
- typedef AAX_Component< void >::CPacketAllocator [AAX_CPacketAllocator](#)
A user-defined callback that AAX calls to notify the component that an instance is being added or removed.
- typedef AAX_Component< void >::CBackgroundProc [AAX_CBackgroundProc](#)
A user-defined callback that AAX calls in the AAX Idle time.

- **typedef AAX_Component< void >::CInitPrivateDataProc AAX_CInitPrivateDataProc**
A user-defined callback to initialize a private data block.

Enumerations

- **enum AAX_CProcPtrID { kAAX_ProcPtrID_Create_EffectParameters = 0, kAAX_ProcPtrID_Create_Effect← GUI = 1, kAAX_ProcPtrID_Create_HostProcessor = 3, kAAX_ProcPtrID_Create_EffectDirectData = 5 }**

15.15.2 Typedef Documentation

15.15.2.1 typedef IACFUnknown* AAX_CALLBACK* AAXCreateObjectProc(void)

15.15.2.2 typedef AAX_Component<void>::CProcessProc AAX_CProcessProc

A user-defined callback that AAX calls to process data packets and/or audio.

iContextPtrsBegin

A vector of context pointers. Each element points to the context for one instance of this component. *iContextPtrsEnd* gives the upper bound of the vector and (*iContextPtrsEnd* - *iContextPtrsBegin*) gives the count.

iContextPtrsEnd

The upper bound of the vector at *iContextPtrsBegin*. (*iContextPtrsEnd* - *iContextPtrsBegin*) gives the count of this vector.

The instance vector was originally NULL-terminated in earlier versions of this API. However, the STL-style begin/end pattern was suggested as a more general representation that could, for instance, allow a vector to be split for parallel processing.

15.15.2.3 typedef AAX_Component<void>::CPacketAllocator AAX_CPacketAllocator

Used by **AAX_SchedulePacket()**

Deprecated

A **AAX_CProcessProc** that calls **AAX_SchedulePacket()** must include a **AAX_CPacketAllocator** field in its context and register that field with **AAX**. AAX will then populate that field with a **AAX_CPacketAllocator** to pass to **AAX_SchedulePacket()**.

See also

AAX_SchedulePacket()

15.15.2.4 typedef AAX_Component<void>::CInstanceInitProc AAX_CInstanceInitProc

A user-defined callback that AAX calls to notify the component that an instance is being added or removed.

This optional callback allows the component to keep per-instance data. It's called before the instance appears in the list supplied to **CProcessProc**, and then after the instance is removed from the list.

iInstanceContextPtr

A pointer to the context data structure of the instance being added or removed from the processing list.

iAction

Indicates the action that triggered the init callback, e.g. whether the instance is being added or removed.

Return values

<i>Should</i>	return 0 on success, anything else on failure. Failure will prevent the instance from being created.
---------------	--

15.15.2.5 `typedef AAX_Component<void>::CBackgroundProc AAX_CBackgroundProc`

A user-defined callback that AAX calls in the AAX Idle time.

This optional callback allows the component to do background processing in whatever manner the plug-in developer desires

Return values

<i>Should</i>	return 0 on success, anything else on failure. Failure will cause the AAX host to signal an error up the callchain.
---------------	---

15.15.2.6 `typedef AAX_Component<void>::CInitPrivateDataProc AAX_CInitPrivateDataProc`

A user-defined callback to initialize a private data block.

Deprecated

A component that requires private data supplies [AAX_CInitPrivateDataProc](#) callbacks to set its private data to the state it should be in at the start of audio. The component first declares one or more pointers to private data in its context. It then registers each such field with AAX along with its data size, various other attributes, and a [AAX_CInitPrivateDataProc](#). The [AAX_CInitPrivateDataProc](#) always runs on the host system, not the DSP. AAX allocates storage for each private data block and calls its associated [AAX_CInitPrivateDataProc](#) to initialize it. If the component's [AAX_CProcessProc](#) runs on external hardware, AAX initializes private data blocks on the host system and copies them to the remote system.

See also

[alg_pd_init](#)

inFieldIndex

The port ID of the block to be initialized, as generated by [AAX_FIELD_INDEX\(\)](#). A component can register a separate [AAX_CInitPrivateDataProc](#) for each of its private data blocks, or it can use fewer functions that switch on `inFieldIndex`.

inNewBlock

A pointer to the block to be initialized. If the component runs externally, AAX will copy this block to the remote system after it is initialized.

inSize

The size of the block to be initialized. If a component has multiple private blocks that only need to be zeroed out, say, it can use a single [AAX_CInitPrivateDataProc](#) for all of these blocks that zeroes them out according to `inSize`.

inController

A pointer to the current Effect instance's [AAX_IController](#).

Note

Do not directly reference data from this interface when populating `iNewBlock`. The data in this block must be fully self-contained to ensure portability to a new device or memory space.

Deprecated

15.15.3 Enumeration Type Documentation

15.15.3.1 enum AAX_CProcPtrID

Enumerator

- kAAX_ProcPtrID_Create_EffectParameters* [AAX_IEffectParameters](#) creation procedure
- kAAX_ProcPtrID_Create_EffectGUI* [AAX_IEffectGUI](#) creation procedure
- kAAX_ProcPtrID_Create_HostProcessor* [AAX_IHostProcessor](#) creation procedure
- kAAX_ProcPtrID_Create_EffectDirectData* [AAX_IEffectDirectData](#) creation procedure, used by PIs that want direct access to their alg memory

15.16 AAX_CAtomicQueue.h File Reference

```
#include "AAX_IPointerQueue.h" #include "AAX_Atomic.h" #include "AAX_CMutex.h" #include <cstring>
```

15.16.1 Description

Atomic, non-blocking queue.

Classes

- class [AAX_CAtomicQueue< T, S >](#)

15.17 AAC_AutoreleasePool.h File Reference

15.17.1 Description

Autorelease pool helper utility.

Classes

- class [AAC_AutoreleasePool](#)

Macros

- `#define _AAC_AUTORELEASEPOOL_H_`

15.17.2 Macro Definition Documentation

15.17.2.1 `#define _AAC_AUTORELEASEPOOL_H_`

15.18 AAC_CBinaryDisplayDelegate.h File Reference

```
#include "AAC_IDisplayDelegate.h" #include "AAC_CString.h" #include <vector> #include <algorithm>
```

15.18.1 Description

A binary display delegate.

Classes

- class [AAX_CBinaryDisplayDelegate< T >](#)
A binary display format conforming to [AAX_IDisplayDelegate](#).

15.19 AAX_CBinaryTaperDelegate.h File Reference

```
#include "AAX_ITaperDelegate.h"
```

15.19.1 Description

A binary taper delegate.

Classes

- class [AAX_CBinaryTaperDelegate< T >](#)
A binary taper conforming to [AAX_ITaperDelegate](#).

15.20 AAX_CChunkDataParser.h File Reference

```
#include "AAX.h"#include "AAX_CString.h"#include <vector>
```

15.20.1 Description

Parser utility for plugin chunks.

Classes

- class [AAX_CChunkDataParser](#)
Parser utility for plugin chunks.
- struct [AAX_CChunkDataParser::DataValue](#)

Namespaces

- [AAX_ChunkDataParserDefs](#)
Constants used by ChunkDataParser class.

Macros

- #define [AAX_CHUNKDATAPARSER_H](#)

Variables

- const int32_t [AAX_ChunkDataParserDefs::FLOAT_TYPE](#) = 1
- const char [AAX_ChunkDataParserDefs::FLOAT_STRING_IDENTIFIER](#) [] = "f_"
- const int32_t [AAX_ChunkDataParserDefs::LONG_TYPE](#) = 2
- const char [AAX_ChunkDataParserDefs::LONG_STRING_IDENTIFIER](#) [] = "l_"
- const int32_t [AAX_ChunkDataParserDefs::DOUBLE_TYPE](#) = 3
- const char [AAX_ChunkDataParserDefs::DOUBLE_STRING_IDENTIFIER](#) [] = "d_"
- const size_t [AAX_ChunkDataParserDefs::DOUBLE_TYPE_SIZE](#) = 8
- const size_t [AAX_ChunkDataParserDefs::DOUBLE_TYPE_INCR](#) = 8
- const int32_t [AAX_ChunkDataParserDefs::SHORT_TYPE](#) = 4
- const char [AAX_ChunkDataParserDefs::SHORT_STRING_IDENTIFIER](#) [] = "s_"
- const size_t [AAX_ChunkDataParserDefs::SHORT_TYPE_SIZE](#) = 2
- const size_t [AAX_ChunkDataParserDefs::SHORT_TYPE_INCR](#) = 4
- const int32_t [AAX_ChunkDataParserDefs::STRING_TYPE](#) = 5
- const char [AAX_ChunkDataParserDefs::STRING_STRING_IDENTIFIER](#) [] = "r_"
- const size_t [AAX_ChunkDataParserDefs::MAX_STRINGDATA_LENGTH](#) = 255
- const size_t [AAX_ChunkDataParserDefs::DEFAULT32BIT_TYPE_SIZE](#) = 4
- const size_t [AAX_ChunkDataParserDefs::DEFAULT32BIT_TYPE_INCR](#) = 4
- const size_t [AAX_ChunkDataParserDefs::STRING_IDENTIFIER_SIZE](#) = 2
- const int32_t [AAX_ChunkDataParserDefs::NAME_NOT_FOUND](#) = -1
- const size_t [AAX_ChunkDataParserDefs::MAX_NAME_LENGTH](#) = 255
- const int32_t [AAX_ChunkDataParserDefs::BUILD_DATA_FAILED](#) = -333
- const int32_t [AAX_ChunkDataParserDefs::HEADER_SIZE](#) = 4
- const int32_t [AAX_ChunkDataParserDefs::VERSION_ID_1](#) = 0x01010101

15.20.2 Macro Definition Documentation

15.20.2.1 #define [AAX_CHUNKDATAPARSER_H](#)

15.21 AAX_CDecibelDisplayDelegateDecorator.h File Reference

```
#include "AAX_IDisplayDelegateDecorator.h" #include <cmath>
```

15.21.1 Description

A decibel display delegate.

Classes

- class [AAX_CDecibelDisplayDelegateDecorator< T >](#)
A percent decorator conforming to [AAX_IDisplayDelegateDecorator](#).

15.22 AAX_CEffectDirectData.h File Reference

```
#include "AAX_IEffectDirectData.h"
```

15.22.1 Description

A default implementation of the [AAX_IEffectDirectData](#) interface.

Classes

- class [AAX_CEffectDirectData](#)
Default implementation of the [AAX_IEffectDirectData](#) interface.

Macros

- `#define AAX_CFFECTDIRECTDATA_H`

15.22.2 Macro Definition Documentation

15.22.2.1 `#define AAX_CFFECTDIRECTDATA_H`

15.23 AAX_CEffectGUI.h File Reference

```
#include "AAX_IEffectGUI.h"#include "AAX_IACFEffectParameters.h"#include
<string>#include <vector>#include <map>
```

15.23.1 Description

A default implementation of the [AAX_IEffectGUI](#) interface.

Classes

- class [AAX_CEffectGUI](#)
Default implementation of the [AAX_IEffectGUI](#) interface.

15.24 AAX_CEffectParameters.h File Reference

```
#include "AAX_IEffectParameters.h"#include "AAX_IPageTable.h"#include "A-
AX_CString.h"#include "AAX_CChunkDataParser.h"#include "AAX_CParameter-
Manager.h"#include "AAX_CPacketDispatcher.h"#include <set>#include <string>#include
<vector>
```

15.24.1 Description

A default implementation of the [AAX_IeffectParameters](#) interface.

Classes

- class [AAX_CEffectParameters](#)
Default implementation of the [AAX_IEffectParameters](#) interface.

Functions

- `int32_t NormalizedToInt32 (double normalizedValue)`
- `double Int32ToNormalized (int32_t value)`
- `double BoolToNormalized (bool value)`

Variables

- [AAX_CParamID cPreviewID](#)
- [AAX_CParamID cDefaultMasterBypassID](#)

15.24.2 Function Documentation

15.24.2.1 `int32_t NormalizedToInt32 (double normalizedValue)`

15.24.2.2 `double Int32ToNormalized (int32_t value)`

15.24.2.3 `double BoolToNormalized (bool value)`

15.24.3 Variable Documentation

15.24.3.1 [AAX_CParamID cPreviewID](#)

15.24.3.2 [AAX_CParamID cDefaultMasterBypassID](#)

15.25 AAX_CHostProcessor.h File Reference

```
#include "AAX_IEffectParameters.h"#include "AAX_IHostProcessor.h"#include  
"ACFPtr.h"
```

15.25.1 Description

Concrete implementation of the [AAX_IHostProcessor](#) interface for non-real-time processing.

Classes

- class [AAX_CHostProcessor](#)

Concrete implementation of the [AAX_IHostProcessor](#) interface for non-real-time processing.

15.26 AAX_CHostServices.h File Reference

```
#include "AAX.h"#include "AAX.Enums.h"
```

15.26.1 Description

Concrete implementation of the [AAX_IHostServices](#) interface.

Classes

- class [AAX_CHostServices](#)

Method access to a singleton implementation of the [AAX_IHostServices](#) interface.

15.27 AAX_CLinearTaperDelegate.h File Reference

```
#include "AAX_ITaperDelegate.h" #include "AAX.h" #include <cmath>
```

15.27.1 Description

A linear taper delegate.

Classes

- class [AAX_CLinearTaperDelegate< T, RealPrecision >](#)

A linear taper conforming to [AAX_ITaperDelegate](#).

15.28 AAX_CLogTaperDelegate.h File Reference

```
#include "AAX_ITaperDelegate.h" #include "AAX_UtilsNative.h" #include "AAX.h" #include <cmath>
```

15.28.1 Description

A log taper delegate.

Classes

- class [AAX_CLogTaperDelegate< T, RealPrecision >](#)

A logarithmic taper conforming to [AAX_ITaperDelegate](#).

15.29 AAX_CMonolithicParameters.cpp File Reference

```
#include "AAX_CMonolithicParameters.h" #include "AAX_Exception.h"
```

15.30 AAX_CMonolithicParameters.h File Reference

```
#include "AAX_CEffectParameters.h" #include "AAX_IEffectDescriptor.h" #include "AAX_IComponentDescriptor.h" #include "AAX_IPropertyMap.h" #include "AAX_CAtomicQueue.h" #include "AAX_IParameter.h" #include "AAX_IMIDINode.h" #include "AAX_IString.h" #include <set> #include <list> #include <utility>
```

15.30.1 Description

A convenience class extending [AAX_CEffectParameters](#) for monolithic instruments.

Classes

- struct [AAX_SInstrumentSetupInfo](#)
Information used to describe the instrument.
- struct [AAX_SInstrumentPrivateData](#)
Utility struct for [AAX_CMonolithicParameters](#).
- struct [AAX_SInstrumentRenderInfo](#)
Information used to parameterize [AAX_CMonolithicParameters::RenderAudio\(\)](#)
- class [AAX_CMonolithicParameters](#)
Extension of the [AAX_CEffectParameters](#) class for monolithic VIs and effects.

Macros

- #define [kMaxAdditionalMIDINodes](#) 15
- #define [kMaxAuxOutputStems](#) 32
- #define [kSynchronizedParameterQueueSize](#) 32

15.30.2 Macro Definition Documentation

15.30.2.1 #define kMaxAdditionalMIDINodes 15

Referenced by [AAX_CMonolithicParameters::StaticDescribe\(\)](#).

15.30.2.2 #define kMaxAuxOutputStems 32

Referenced by [AAX_SInstrumentSetupInfo::AAX_SInstrumentSetupInfo\(\)](#), and [AAX_CMonolithicParameters::StaticDescribe\(\)](#).

15.30.2.3 #define kSynchronizedParameterQueueSize 32

Referenced by [AAX_CMonolithicParameters::AddSynchronizedParameter\(\)](#).

15.31 AAX_CMutex.h File Reference

15.31.1 Description

Mutex.

Classes

- class [AAX_CMutex](#)
Mutex with try lock functionality.
- class [AAX_StLock_Guard](#)
Helper class for working with mutex.

15.32 AAX_CNumberDisplayDelegate.h File Reference

```
#include "AAX_IDisplayDelegate.h" #include "AAX_CString.h"
```

15.32.1 Description

A number display delegate.

Classes

- class [AAX_CNumberDisplayDelegate< T, Precision, SpaceAfter >](#)

A numeric display format conforming to AAX_IDisplayDelegate.

15.33 AAX_CommonConversions.h File Reference

```
#include <math.h> #include "AAX.h"
```

Functions

- double [GainToDB](#) (double aGain)
Convert Gain to dB.
- double [DBToGain](#) (double dB)
Convert dB to Gain.
- double [LongToDouble](#) (int32_t aLong)
Convert Long to Double.
- int32_t [DoubleToLong](#) (double aDouble)
convert floating point equivalent back to int32_t
- int32_t [DoubleToDSPCoef](#) (double d, double max=[k56kFloatPosMax](#), double min=[k56kFloatNegMax](#))
Convert Double to DSPCoef.
- double [DSPCoefToDouble](#) (int32_t c, int32_t max=[k56kFracPosMax](#), int32_t min=[k56kFracNegMax](#))
Convert DSPCoef to Double.
- double [ThirtyTwoBitDSPCoefToDouble](#) (int32_t c)
ThirtyTwoBitDSPCoefToDouble.
- int32_t [DoubleTo32BitDSPCoefRnd](#) (double d)
DoubleTo32BitDSPCoefRnd.
- int32_t [DoubleTo32BitDSPCoef](#) (double d)
- int32_t [DoubleToDSPCoefRnd](#) (double d, double max, double min)

Variables

- const int32_t [k32BitPosMax](#) = 0x7FFFFFFF
- const int32_t [k32BitAbsMax](#) = 0x80000000
- const int32_t [k32BitNegMax](#) = 0x80000000
- const int32_t [k56kFracPosMax](#) = 0x007FFFFF
- const int32_t [k56kFracAbsMax](#) = 0x00800000
- const int32_t [k56kFracHalf](#) = 0x00400000
- const int32_t [k56kFracNegOne](#) = 0xFF800000
- const int32_t [k56kFracNegMax](#) = [k56kFracNegOne](#)
- const int32_t [k56kFracZero](#) = 0x00000000

- const double `kOneOver56kFracAbsMax` = 1.0/double(`k56kFracAbsMax`)
- const double `k56kFloatPosMax` = double(`k56kFracPosMax`)/double(`k56kFracAbsMax`)
- const double `k56kFloatNegMax` = -1.0
- const double `kNeg144DB` = -144.0
- const double `kNeg144Gain` = 6.3095734448019324943436013662234e-8

15.33.1 Function Documentation

15.33.1.1 double GainToDB (double `aGain`) [inline]

Convert Gain to dB.

Todo This should be incorporated into parameters' tapers and not called separately

References `kNeg144DB`.

15.33.1.2 double DBToGain (double `dB`) [inline]

Convert dB to Gain.

Todo This should be incorporated into parameters' tapers and not called separately

15.33.1.3 double LongToDouble (int32_t `aLong`) [inline]

Convert Long to Double.

`LongToDouble`: convert 24 bit fixed point in a `int32_t` to floating point equivalent

References `k56kFracNegMax`, `k56kFracPosMax`, and `kOneOver56kFracAbsMax`.

15.33.1.4 int32_t DoubleToLong (double `aDouble`)

convert floating point equivalent back to `int32_t`

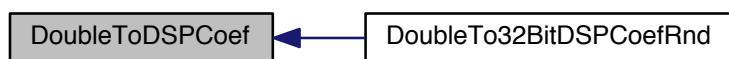
15.33.1.5 int32_t DoubleToDSPCoef (double `d`, double `max = k56kFloatPosMax`, double `min = k56kFloatNegMax`) [inline]

Convert Double to DSPCoef.

References `k56kFracAbsMax`, `k56kFracNegMax`, and `k56kFracPosMax`.

Referenced by `DoubleTo32BitDSPCoefRnd()`.

Here is the caller graph for this function:



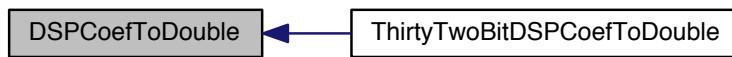
```
15.33.1.6 double DSPCoef.ToDouble ( int32_t c, int32_t max = k56kFracPosMax, int32_t min = k56kFracNegMax )  
[inline]
```

Convert DSPCoef to Double.

References k56kFracNegMax, k56kFracPosMax, and kOneOver56kFracAbsMax.

Referenced by ThirtyTwoBitDSPCoef.ToDouble().

Here is the caller graph for this function:

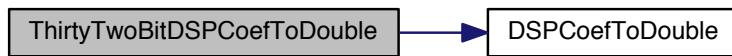


```
15.33.1.7 double ThirtyTwoBitDSPCoef.ToDouble ( int32_t c ) [inline]
```

ThirtyTwoBitDSPCoef.ToDouble.

References DSPCoef.ToDouble(), k32BitNegMax, and k32BitPosMax.

Here is the call graph for this function:



```
15.33.1.8 int32_t DoubleTo32BitDSPCoefRnd ( double d ) [inline]
```

DoubleTo32BitDSPCoefRnd.

References DoubleToDSPCoef(), k32BitNegMax, and k32BitPosMax.

Here is the call graph for this function:



15.33.1.9 `int32_t DoubleTo32BitDSPCoef (double d)`

15.33.1.10 `int32_t DoubleToDSPCoefRnd (double d, double max, double min)`

15.33.2 Variable Documentation

15.33.2.1 `const int32_t k32BitPosMax = 0x7FFFFFFF`

Referenced by `DoubleTo32BitDSPCoefRnd()`, and `ThirtyTwoBitDSPCoef.ToDouble()`.

15.33.2.2 `const int32_t k32BitAbsMax = 0x80000000`

15.33.2.3 `const int32_t k32BitNegMax = 0x80000000`

Referenced by `DoubleTo32BitDSPCoefRnd()`, and `ThirtyTwoBitDSPCoef.ToDouble()`.

15.33.2.4 `const int32_t k56kFracPosMax = 0x007FFFFF`

Referenced by `DoubleToDSPCoef()`, `DSPCoef.ToDouble()`, and `Long.ToDouble()`.

15.33.2.5 `const int32_t k56kFracAbsMax = 0x00800000`

Referenced by `DoubleToDSPCoef()`.

15.33.2.6 `const int32_t k56kFracHalf = 0x00400000`

15.33.2.7 `const int32_t k56kFracNegOne = 0xFF800000`

15.33.2.8 `const int32_t k56kFracNegMax = k56kFracNegOne`

Referenced by `DoubleToDSPCoef()`, `DSPCoef.ToDouble()`, and `Long.ToDouble()`.

15.33.2.9 `const int32_t k56kFracZero = 0x00000000`

15.33.2.10 `const double kOneOver56kFracAbsMax = 1.0/double(k56kFracAbsMax)`

Referenced by `DSPCoef.ToDouble()`, and `Long.ToDouble()`.

15.33.2.11 `const double k56kFloatPosMax = double(k56kFracPosMax)/double(k56kFracAbsMax)`

15.33.2.12 `const double k56kFloatNegMax = -1.0`

15.33.2.13 `const double kNeg144DB = -144.0`

Referenced by `GainToDB()`.

15.33.2.14 `const double kNeg144Gain = 6.3095734448019324943436013662234e-8`

15.34 AAX_CommonInterface_Algorithm.dox File Reference

15.35 AAX_CommonInterface_Communication.dox File Reference

15.36 AAX_CommonInterface_DataModel.dox File Reference

15.37 AAX_CommonInterface_Describe.dox File Reference

Functions

- [AAX_Result GetEffectDescriptions \(AAX_ICollection *inCollection\)](#)
The plug-in's static Description entrypoint.

15.38 AAX_CommonInterface_FormatSpecification.dox File Reference

15.39 AAX_CommonInterface_GUI.dox File Reference

15.40 AAX_Constants.h File Reference

```
#include <cmath>
```

15.40.1 Description

Signal processing constants.

Namespaces

- [AAX](#)

Macros

- [#define AAX_CONSTANTS_H](#)

Enumerations

- enum [AAX::ESampleRates](#) { [AAX::e44100SampleRate](#) = 44100, [AAX::e48000SampleRate](#) = 48000, [AAX::e88200SampleRate](#) = 88200, [AAX::e96000SampleRate](#) = 96000, [AAX::e176400SampleRate](#) = 176400, [AAX::e192000SampleRate](#) = 192000 }

Variables

- const int [AAX::cBigEndian](#) =0
- const int [AAX::cLittleEndian](#) =1
- const double [AAX::cPi](#) = 3.1415926535897932384626433832795
- const double [AAX::cTwoPi](#) = 6.2831853071795862319959269370884
- const double [AAX::cHalfPi](#) = 1.5707963267948965579989817342721
- const double [AAX::cQuarterPi](#) = 0.78539816339744827899949086713605
- const double [AAX::cRootTwo](#) = 1.4142135623730950488016887242097
- const double [AAX::cOneOverRootTwo](#) = 0.70710678118654752440084436210485
- const double [AAX::cPos3dB](#) = 1.4142135623730950488016887242097

- const double [AAX::cNeg3dB](#) =0.70710678118654752440084436210485
- const double [AAX::cPos6dB](#) =2.0
- const double [AAX::cNeg6dB](#) =0.5
- const double [AAX::cNormalizeLongToAmplitudeOneHalf](#) = 0.00000000023283064365386962890625
- const double [AAX::cNormalizeLongToAmplitudeOne](#) = 1.0/double(1<<31)
- const double [AAX::cMilli](#) =0.001
- const double [AAX::cMicro](#) =0.001*0.001
- const double [AAX::cNano](#) =0.001*0.001*0.001
- const double [AAX::cPico](#) =0.001*0.001*0.001*0.001
- const double [AAX::cKilo](#) =1000.0
- const double [AAX::cMega](#) =1000.0*1000.0
- const double [AAX::cGiga](#) =1000.0*1000.0*1000.0

15.40.2 Macro Definition Documentation

15.40.2.1 #define AAX_CONSTANTS_H

15.41 AAX_CPacketDispatcher.h File Reference

```
#include "AAX.h" #include "AAX_IController.h" #include "AAX_CMutex.h" #include
<string> #include <map>
```

15.41.1 Description

Helper classes related to posting AAX packets and handling parameter update events.

Classes

- class [AAX_CPacket](#)
Container for packet-related data.
- struct [AAX_IPacketHandler](#)
Callback container used by AAX_CPacketDispatcher.
- class [AAX_CPacketHandler< TWorker >](#)
Callback container used by AAX_CPacketDispatcher.
- class [AAX_CPacketDispatcher](#)
Helper class for managing AAX packet posting.

15.42 AAX_CParameter.h File Reference

```
#include "AAX_Assert.h" #include "AAX_IParameter.h" #include "AAX_ITaper<
Delegate.h" #include "AAX_IDisplayDelegate.h" #include "AAX_IAutomationDelegate.<
h" #include "AAX_CString.h" #include <cstring> #include <list> #include <map>
```

15.42.1 Description

Generic implementation of an [AAX_IParameter](#).

Classes

- class [AAX_CParameterValue< T >](#)
Concrete implementation of AAX_IParameterValue.
- class [AAX_CParameter< T >](#)
Generic implementation of an AAX_IParameter.
- class [AAX_CStatelessParameter](#)
A stateless parameter implementation.

15.43 AAX_CParameterManager.h File Reference

```
#include "AAX_CParameter.h"#include "AAX.h"#include <vector>#include <map>
```

15.43.1 Description

A container object for plug-in parameters.

Classes

- class [AAX_CParameterManager](#)
A container object for plug-in parameters.

15.44 AAX_CPercentDisplayDelegateDecorator.h File Reference

```
#include "AAX_IDisplayDelegateDecorator.h"#include <cmath>
```

15.44.1 Description

A percent display delegate decorator.

Classes

- class [AAX_CPercentDisplayDelegateDecorator< T >](#)
A percent decorator conforming to AAX_IDisplayDelegateDecorator.

Macros

- [#define AAX_CPERCENTDISPLAYDELEGATEDECORATOR_H](#)

15.44.2 Macro Definition Documentation

15.44.2.1 [#define AAX_CPERCENTDISPLAYDELEGATEDECORATOR_H](#)

15.45 AAX_CPieceWiseLinearTaperDelegate.h File Reference

```
#include "AAX_ITaperDelegate.h"
```

```
#include "AAX.h" #include <cmath>
```

15.45.1 Description

A piece-wise linear taper delegate.

Classes

- class [AAX_CPieceWiseLinearTaperDelegate< T, RealPrecision >](#)
A piece-wise linear taper conforming to [AAX_ITaperDelegate](#).

15.46 AAX_CRangeTaperDelegate.h File Reference

```
#include "AAX_ITaperDelegate.h" #include "AAX.h" #include <cmath> #include <vector>
```

15.46.1 Description

A range taper delegate decorator.

Classes

- class [AAX_CRangeTaperDelegate< T, RealPrecision >](#)
A piecewise-linear taper conforming to [AAX_ITaperDelegate](#).

15.47 AAX_CStateDisplayDelegate.h File Reference

```
#include "AAX_IDisplayDelegate.h" #include "AAX_CString.h" #include <vector>
```

15.47.1 Description

A state display delegate.

Classes

- class [AAX_CStateDisplayDelegate< T >](#)
A generic display format conforming to [AAX_IDisplayDelegate](#).

15.48 AAX_CStateTaperDelegate.h File Reference

```
#include "AAX_ITaperDelegate.h" #include "AAX.h" #include <cmath>
```

15.48.1 Description

A state taper delegate (similar to a linear taper delegate.)

Classes

- class [AAX_CStateTaperDelegate< T >](#)
A linear taper conforming to AAX_ITaperDelegate.

15.49 AAX_CString.h File Reference

```
#include "AAX_IString.h"#include <string>#include <map>
```

15.49.1 Description

A generic AAX string class with similar functionality to std::string.

Classes

- class [AAX_CString](#)
A generic AAX string class with similar functionality to std::string
- class [AAX_CStringAbbreviations](#)
Helper class to store a collection of name abbreviations.

Functions

- [AAX_CString operator+ \(AAX_CString lhs, const AAX_CString &rhs\)](#)
- [AAX_CString operator+ \(AAX_CString lhs, const char *rhs\)](#)
- [AAX_CString operator+ \(const char *lhs, const AAX_CString &rhs\)](#)

15.49.2 Function Documentation

15.49.2.1 [AAX_CString operator+ \(AAX_CString lhs, const AAX_CString & rhs \) \[inline\]](#)

15.49.2.2 [AAX_CString operator+ \(AAX_CString lhs, const char * rhs \) \[inline\]](#)

15.49.2.3 [AAX_CString operator+ \(const char * lhs, const AAX_CString & rhs \) \[inline\]](#)

15.50 AAX_CStringDisplayDelegate.h File Reference

```
#include "AAX_IDisplayDelegate.h"#include <sstream>#include <map>
```

15.50.1 Description

A string display delegate.

Classes

- class [AAX_CStringDisplayDelegate< T >](#)
A string, or list, display format conforming to AAX_IDisplayDelegate.

15.51 AAX_CUnitDisplayDelegateDecorator.h File Reference

```
#include "AAX_IDisplayDelegateDecorator.h"
```

15.51.1 Description

A unit display delegate decorator.

Classes

- class [AAX_CUnitDisplayDelegateDecorator< T >](#)
A unit type decorator conforming to [AAX_IDisplayDelegateDecorator](#).

15.52 AAX_CUnitPrefixDisplayDelegateDecorator.h File Reference

```
#include "AAX_IDisplayDelegateDecorator.h"
```

15.52.1 Description

A unit prefix display delegate decorator.

Classes

- class [AAX_CUnitPrefixDisplayDelegateDecorator< T >](#)
A unit prefix decorator conforming to [AAX_IDisplayDelegateDecorator](#).

15.53 AAX_Denormal.h File Reference

```
#include "AAX.h" #include "AAX_PlatformOptimizationConstants.h" #include <limits> #include <math.h>
```

15.53.1 Description

Signal processing utilities for denormal/subnormal floating point numbers.

Namespaces

- [AAX](#)

Macros

- `#define AAX_DENORMAL_H`
- `#define AAX_SCOPE_COMPUTE_DENORMALS() do {} while(0)`
Sets the run-time environment to handle denormal floats within the scope of this macro.
- `#define AAX_SCOPE_DENORMALS_AS_ZERO() do {} while(0)`
Sets the run-time environment to treat denormal floats as zero within the scope of this macro.

Functions

- void [AAX::DeDenormal](#) (double &iValue)
Clamps very small floating point values to zero.
- void [AAX::DeDenormal](#) (float &iValue)
Clamps very small floating point values to zero.
- void [AAX::DeDenormalFine](#) (float &iValue)
- void [AAX::FilterDenormals](#) (float *inSamples, int32_t inLength)
Round all denormal/subnormal samples in a buffer to zero.

Variables

- const double [AAX::cDenormalAvoidanceOffset](#) =3.0e-34
- const float [AAX::cFloatDenormalAvoidanceOffset](#) =3.0e-20f

15.53.2 Macro Definition Documentation

15.53.2.1 #define AAX_DENORMAL_H

15.53.2.2 #define AAX_SCOPE_COMPUTE_DENORMALS() do {} while(0)

Sets the run-time environment to handle denormal floats within the scope of this macro.

The host sets the denormal policy for all [AAX](#) threads and may use settings which treat denormal float values as zero (DAZ+FZ). This macro forces non-DAZ behavior.

15.53.2.3 #define AAX_SCOPE_DENORMALS_AS_ZERO() do {} while(0)

Sets the run-time environment to treat denormal floats as zero within the scope of this macro.

The host sets the denormal policy for all [AAX](#) threads and may already use settings which treat denormal float values as zero (DAZ+FZ). This macro forces DAZ behavior.

15.54 AAX_DigiTrace_Guide.dox File Reference

15.55 AAX_DistributingYourPlugIn.dox File Reference

15.56 AAX_DocsDirectory.dox File Reference

15.57 AAX_EndianSwap.h File Reference

```
#include <algorithm>
```

15.57.1 Description

Utility functions for byte-swapping. Used by [AAX_CChunkDataParser](#).

Macros

- #define ENDIANSWAP_H

Functions

- template<class T > void [AAX_EndianSwapInPlace](#) (T *theDataP)

Byte swap data in-place.
- template<class T > T [AAX_EndianSwap](#) (T theData)

Make a byte-swapped copy of data.
- template<class T > void [AAX_BigEndianNativeSwapInPlace](#) (T *theDataP)

Convert data in-place between Big Endian and native byte ordering.
- template<class T > T [AAX_BigEndianNativeSwap](#) (T theData)

Copy and convert data between Big Endian and native byte ordering.
- template<class T > void [AAX_LittleEndianNativeSwapInPlace](#) (T *theDataP)

Convert data in-place from the native byte ordering to Little Endian byte ordering.
- template<class T > T [AAX_LittleEndianNativeSwap](#) (T theData)

Copy and convert data from the native byte ordering to Little Endian byte ordering.
- template<class Iter > void [AAX_EndianSwapSequenceInPlace](#) (Iter beginl, Iter endl)

Byte swap a sequence of data in-place.
- template<class Iter > void [AAX_BigEndianNativeSwapSequenceInPlace](#) (Iter beginl, Iter endl)

Convert an sequence of data in-place between Big Endian and native byte ordering.
- template<class Iter > void [AAX_LittleEndianNativeSwapSequenceInPlace](#) (Iter beginl, Iter endl)

Convert an sequence of data in-place from the native byte ordering to Little Endian byte ordering.

15.57.2 Macro Definition Documentation

15.57.2.1 #define ENDIANSWAP_H

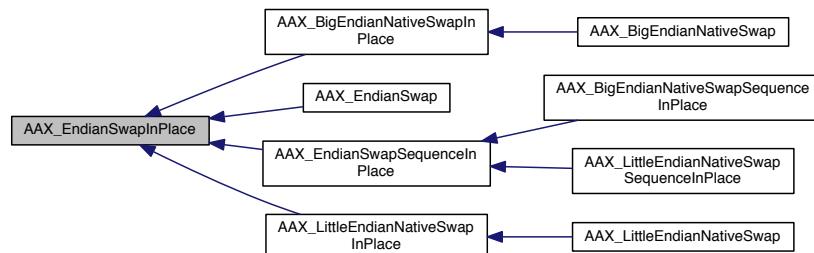
15.57.3 Function Documentation

15.57.3.1 template<class T > void [AAX_EndianSwapInPlace](#) (T * theDataP) [inline]

Byte swap data in-place.

Referenced by [AAX_BigEndianNativeSwapInPlace\(\)](#), [AAX_EndianSwap\(\)](#), [AAX_EndianSwapSequenceInPlace\(\)](#), and [AAX_LittleEndianNativeSwapInPlace\(\)](#).

Here is the caller graph for this function:

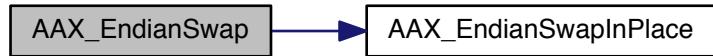


15.57.3.2 template<class T > T AAX_EndianSwap (T *theData*) [inline]

Make a byte-swapped copy of data.

References AAX_EndianSwapInPlace().

Here is the call graph for this function:

**15.57.3.3 template<class T > void AAX_BigEndianNativeSwapInPlace (T * *theDataP*) [inline]**

Convert data in-place between Big Endian and native byte ordering.

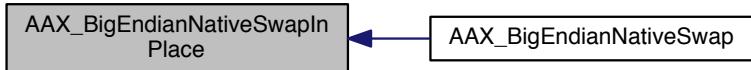
References AAX_EndianSwapInPlace().

Referenced by AAX_BigEndianNativeSwap().

Here is the call graph for this function:



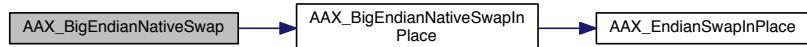
Here is the caller graph for this function:

**15.57.3.4 template<class T > T AAX_BigEndianNativeSwap (T *theData*) [inline]**

Copy and convert data between Big Endian and native byte ordering.

References AAX_BigEndianNativeSwapInPlace().

Here is the call graph for this function:



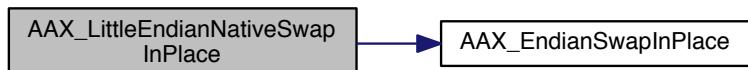
15.57.3.5 template<class T > void AAX_LittleEndianNativeSwapInPlace (T * theDataP) [inline]

Convert data in-place from the native byte ordering to Little Endian byte ordering.

References AAX_EndianSwapInPlace().

Referenced by AAX_LittleEndianNativeSwap().

Here is the call graph for this function:



Here is the caller graph for this function:



15.57.3.6 template<class T > T AAX_LittleEndianNativeSwap (T theData) [inline]

Copy and convert data from the native byte ordering to Little Endian byte ordering.

References AAX_LittleEndianNativeSwapInPlace().

Here is the call graph for this function:



15.57.3.7 template<class Iter > void AAX_EndianSwapSequenceInPlace (Iter beginl, Iter endl) [inline]

Byte swap a sequence of data in-place.

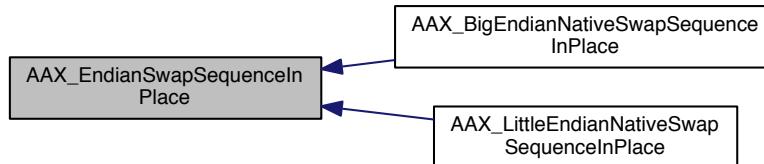
References AAX_EndianSwapInPlace().

Referenced by AAX_BigEndianNativeSwapSequenceInPlace(), and AAX_LittleEndianNativeSwapSequenceInPlace().

Here is the call graph for this function:



Here is the caller graph for this function:

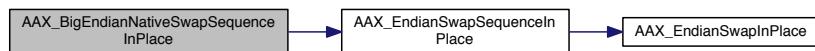


15.57.3.8 template<class Iter > void AAX_BigEndianNativeSwapSequenceInPlace (Iter beginl, Iter endl) [inline]

Convert an sequence of data in-place between Big Endian and native byte ordering.

References AAX_EndianSwapSequenceInPlace().

Here is the call graph for this function:



15.57.3.9 template<class Iter > void AAX_LittleEndianNativeSwapSequenceInPlace (Iter beginl, Iter endl) [inline]

Convert an sequence of data in-place from the native byte ordering to Little Endian byte ordering.

References AAX_EndianSwapSequenceInPlace().

Here is the call graph for this function:



15.58 AAX.Enums.h File Reference

```
#include <stdint.h>
```

15.58.1 Description

Utility functions for byte-swapping. Used by [AAX_CChunkDataParser](#).

Macros

- `#define AAX_INT32_MIN (-2147483647 - 1) /* minimum signed 32 bit value */`
- `#define AAX_INT32_MAX 2147483647 /* maximum signed 32 bit value */`
- `#define AAX_UINT32_MIN 0U /* minimum unsigned 32 bit value */`
- `#define AAX_UINT32_MAX 4294967295U /* maximum unsigned 32 bit value */`
- `#define AAX_INT16_MIN (-32767 - 1) /* minimum signed 16 bit value */`
- `#define AAX_INT16_MAX 32767 /* maximum signed 16 bit value */`
- `#define AAX_UINT16_MIN 0U /* minimum unsigned 16 bit value */`
- `#define AAX_UINT16_MAX 65535U /* maximum unsigned 16 bit value */`
- `#define AAX_ENUM_SIZE_CHECK(x) extern int __enumSizeCheck[2*(sizeof(uint32_t)==sizeof(x)) - 1]`
Macro to ensure enum type consistency across binaries.
- `#define AAX_STEM_FORMAT(alIndex, aChannelCount) (static_cast<uint32_t>((static_cast<uint16_t>(alIndex) << 16) | ((aChannelCount >= AAX_UINT16_MIN) && (aChannelCount <= 0xFFFF) ? aChannelCount & 0xFFFF : 0x0000)))`
- `#define AAX_STEM_FORMAT_CHANNEL_COUNT(aStemFormat) (static_cast<uint16_t>(aStemFormat & 0xFFFF))`
- `#define AAX_STEM_FORMAT_INDEX(aStemFormat) (static_cast<int16_t>((aStemFormat >> 16) & 0xFFFF))`

Typedefs

- `typedef enum AAX_EParameterType AAX_EParameterType`
FIC stuff that I can't include without DAE library dependence.
- `typedef int32_t AAX_EParameterOrientation`
Typedef for a bitfield of AAX_EParameterOrientationBits values.

Enumerations

- `enum AAX_EHighlightColor { AAX_eHighlightColor_Red = 0, AAX_eHighlightColor_Blue = 1, AAX_eHighlightColor_Green = 2, AAX_eHighlightColor_Yellow = 3, AAX_eHighlightColor_Num }`
Highlight color selector.

- enum `AAX_ETracePriorityHost` { `AAX_eTracePriorityHost_None` = 0, `AAX_eTracePriorityHost_High` = 0x08000000, `AAX_eTracePriorityHost_Normal` = 0x04000000, `AAX_eTracePriorityHost_Low` = 0x02000000, `AAX_eTracePriorityHost_Lowest` = 0x01000000 }
Platform-specific tracing priorities.
 - enum `AAX_ETracePriorityDSP` { `AAX_eTracePriorityDSP_None` = 0, `AAX_eTracePriorityDSP_Assert`, `AAX_eTracePriorityDSP_High`, `AAX_eTracePriorityDSP_Normal`, `AAX_eTracePriorityDSP_Low` }
Platform-specific tracing priorities.
 - enum `AAX_EModifiers` { `AAX_eModifiers_None` = 0, `AAX_eModifiers_Shift` = (1 << 0), `AAX_eModifiers_Control` = (1 << 1), `AAX_eModifiers_Option` = (1 << 2), `AAX_eModifiers_Command` = (1 << 3), `AAX_eModifiers_SecondaryButton` = (1 << 4), `AAX_eModifiers_Alt` = `AAX_eModifiers_Option`, `AAX_eModifiers_Cntl` = `AAX_eModifiers_Command`, `AAX_eModifiers_WINKEY` = `AAX_eModifiers_Control` }
Modifier key definitions used by AAX API.
 - enum `AAX_EAudioBufferLength` { `AAX_eAudioBufferLength_Undefined` = -1, `AAX_eAudioBufferLength_1` = 0, `AAX_eAudioBufferLength_2` = 1, `AAX_eAudioBufferLength_4` = 2, `AAX_eAudioBufferLength_8` = 3, `AAX_eAudioBufferLength_16` = 4, `AAX_eAudioBufferLength_32` = 5, `AAX_eAudioBufferLength_64` = 6, `AAX_eAudioBufferLength_128` = 7, `AAX_eAudioBufferLength_256` = 8, `AAX_eAudioBufferLength_512` = 9, `AAX_eAudioBufferLength_1024` = 10,
-

```
AAX_eAudioBufferLength_Max = AAX_eAudioBufferLength_1024 }
```

Generic buffer length definitions.

- enum `AAX_EAudioBufferLengthDSP` { `AAX_eAudioBufferLengthDSP_Default` = `AAX_eAudioBufferLength_4`, `AAX_eAudioBufferLengthDSP_4` = `AAX_eAudioBufferLength_4`, `AAX_eAudioBufferLengthDSP_16` = `AAX_eAudioBufferLength_16`, `AAX_eAudioBufferLengthDSP_32` = `AAX_eAudioBufferLength_32`, `AAX_eAudioBufferLengthDSP_64` = `AAX_eAudioBufferLength_64`, `AAX_eAudioBufferLengthDSP_Max` = `AAX_eAudioBufferLengthDSP_64` }

Currently supported processing buffer length definitions for AAX DSP hosts.

- enum `AAE_EAudioBufferLengthNative` { `AAX_eAudioBufferLengthNative_Min` = `AAX_eAudioBufferLength_32`, `AAX_eAudioBufferLengthNative_Max` = `AAX_eAudioBufferLength_Max` }

Processing buffer length definitions for Native AAX hosts.

- enum `AAX_EMaxAudioSuiteTracks` { `AAX_eMaxAudioSuiteTracks` = 48 }

The maximum number of tracks that an AAX host will process in a non-real-time context.

- enum `AAX_EStemFormat` { `AAX_eStemFormat_Mono` = `AAX_STEM_FORMAT(0, 1)`, `AAX_eStemFormat_Stereo` = `AAX_STEM_FORMAT(1, 2)`, `AAX_eStemFormat_LCR` = `AAX_STEM_FORMAT(2, 3)`, `AAX_eStemFormat_LCRS` = `AAX_STEM_FORMAT(3, 4)`, `AAX_eStemFormat_Quad` = `AAX_STEM_FORMAT(4, 4)`, `AAX_eStemFormat_5_0` = `AAX_STEM_FORMAT(5, 5)`, `AAX_eStemFormat_5_1` = `AAX_STEM_FORMAT(6, 6)`, `AAX_eStemFormat_6_0` = `AAX_STEM_FORMAT(7, 6)`, `AAX_eStemFormat_6_1` = `AAX_STEM_FORMAT(8, 7)`, `AAX_eStemFormat_7_0_SDDS` = `AAX_STEM_FORMAT(9, 7)`, `AAX_eStemFormat_7_1_SDDS` = `AAX_STEM_FORMAT(10, 8)`, `AAX_eStemFormat_7_0_DTS` = `AAX_STEM_FORMAT(11, 7)`, `AAX_eStemFormat_7_1_DTS` = `AAX_STEM_FORMAT(12, 8)`, `AAX_eStemFormat_7_0_2` = `AAX_STEM_FORMAT(20, 9)`, `AAX_eStemFormat_7_1_2` = `AAX_STEM_FORMAT(13, 10)`, `AAX_eStemFormat_Ambi_1_ACN` = `AAX_STEM_FORMAT(14, 4)`, `AAX_eStemFormat_Ambi_2_ACN` = `AAX_STEM_FORMAT(18, 9)`, `AAX_eStemFormat_Ambi_3_ACN` = `AAX_STEM_FORMAT(19, 16)`, `AAX_eStemFormat_Reserved_1` = `AAX_STEM_FORMAT(15, 4)`, `AAX_eStemFormat_Reserved_2` = `AAX_STEM_FORMAT(16, 9)`, `AAX_eStemFormat_Reserved_3` = `AAX_STEM_FORMAT(17, 16)`, `AAX_eStemFormatNum` = 21, `AAX_eStemFormatNone` = `AAX_STEM_FORMAT(-100, 0)`, `AAX_eStemFormatAny` = `AAX_STEM_FORMAT(-1, 0)`, `AAX_eStemFormat_INT32_MAX` = `AAX_INT32_MAX` }

Stem format definitions.

- enum `AAX_EPlugInCategory` { `AAX_ePlugInCategory_None` = 0x00000000, `AAX_ePlugInCategory_EQ` = 0x00000001, `AAX_ePlugInCategory_Dynamics` = 0x00000002, `AAX_ePlugInCategory_PitchShift` = 0x00000004, `AAX_ePlugInCategory_Reverb` = 0x00000008, `AAX_ePlugInCategory_Delay` = 0x00000010, `AAX_ePlugInCategory_Modulation` = 0x00000020, `AAX_ePlugInCategory_Harmonic` = 0x00000040, `AAX_ePlugInCategory_NoiseReduction` = 0x00000080, `AAX_ePlugInCategory_Dither` = 0x00000100, `AAX_ePlugInCategory_SoundField` = 0x00000200, `AAX_ePlugInCategory_HWGenerators` = 0x00000400, `AAX_ePlugInCategory_SWGenerators` = 0x00000800, `AAX_ePlugInCategory_WrappedPlugin` = 0x00001000, `AAX_EPlugInCategory_Effect` = 0x00002000, `AAX_ePlugInCategory_Example` = `AAX_EPlugInCategory_Effect`, `AAX_ePlugInCategory_INT32_MAX` = `AAX_INT32_MAX` }

Effect category definitions.

- enum `AAX_EPlugInStrings` { `AAX_ePlugInStrings_Analysis` = 0, `AAX_ePlugInStrings_MonoMode` = 1, `AAX_ePlugInStrings_MultiInputMode` = 2, `AAX_ePlugInStrings_RegionByRegionAnalysis` = 3, `AAX_ePlugInStrings_AllSelectedRegionsAnalysis` = 4, `AAX_ePlugInStrings_RegionName` = 5, `AAX_ePlugInStrings_ClipName` = 5, `AAX_ePlugInStrings_Progress` = 6, `AAX_ePlugInStrings_PluginFileName` = 7, `AAX_ePlugInStrings_Preview` = 8, `AAX_ePlugInStrings_Process` = 9, `AAX_ePlugInStrings_Bypass` = 10, `AAX_ePlugInStrings_ClipNameSuffix` = 11, `AAX_ePlugInStrings_INT32_MAX` = `AAX_INT32_MAX` }

Effect string identifiers.

- enum `AAX_EMeterOrientation` { `AAX_eMeterOrientation_Default` = 0, `AAX_eMeterOrientation_BottomLeft` = `AAX_eMeterOrientation_Default`, `AAX_eMeterOrientation_TopRight` = 1, `AAX_eMeterOrientation_Center` = 2, `AAX_eMeterOrientation_PhaseDot` = 3 }

Meter orientation.

- enum `AAX_EMeterBallisticType` { `AAX_eMeterBallisticType_Host` = 0, `AAX_eMeterBallisticType_NoDecay` = 1 }

Meter ballistics type.

- enum `AAX_EMeterType` { `AAX_eMeterType_Input` = 0, `AAX_eMeterType_Output` = 1, `AAX_eMeterType_CLGain` = 2, `AAX_eMeterType_EGGain` = 3, `AAX_eMeterType_Analysis` = 4, `AAX_eMeterType_Other` = 5, `AAX_eMeterTypeNone` = 31 }

Meter type.

- enum [AAECurveType](#) { [AAECurveTypeNone](#) = 0, [AAECurveTypeEQ](#) = 'AXeq', [AAECurveTypeDynamics](#) = 'AXdy', [AAECurveTypeReduction](#) = 'AXdr' }

Different Curve Types that can be queried from the Host.

- enum [AAEResourceType](#) { [AAEResourceTypeNone](#) = 0, [AAEResourceTypePageTable](#), [AAEResourceTypePageTableDir](#) }

Types of resources that can be added to an Effect's description.

- enum [AAENotificationEvent](#) { [AAENotificationEventInsertPositionChanged](#) = 'AXip', [AAENotificationEventTrackNameChanged](#) = 'AXtn', [AAENotificationEventTrackUIDChanged](#) = 'AXtu', [AAENotificationEventTrackPositionChanged](#) = 'AXtp', [AAENotificationEventAlgorithmMoved](#) = 'AXam', [AAENotificationEventGUIOpened](#) = 'AXgo', [AAENotificationEventGUIClosed](#) = 'AXgc', [AAENotificationEventASProcessingState](#) = 'AXPr', [AAENotificationEventASPreviewState](#) = 'ASPV', [AAENotificationEventSessionBeingOpened](#) = 'AXso', [AAENotificationEventPresetOpened](#) = 'AXpo', [AAENotificationEventEnteringOfflineMode](#) = 'AXof', [AAENotificationEventExitingOfflineMode](#) = 'AXox', [AAENotificationEventSessionPathChanged](#) = 'AXsp', [AAENotificationEventSignalLatencyChanged](#) = 'AXsl', [AAENotificationEventDelayCompensationState](#) = 'AXdc', [AAENotificationEventCycleCountChanged](#) = 'AXcc', [AAENotificationEventMaxViewSizeChanged](#) = 'AXws', [AAENotificationEventSideChainBeingConnected](#) = 'AXsc', [AAENotificationEventSideChainBeingDisconnected](#) = 'AXsd', [AAENotificationEventNoiseFloorChanged](#) = 'AXnf', [AAENotificationEventParameterMappingChanged](#) = 'AXpm', [AAENotificationEventHostModeChanged](#) = 'AXHm' }

Events IDs for AAX notifications.

- enum [AAEHostModeBits](#) { [AAEHostModeBitsNone](#) = 0, [AAEHostModeBitsLive](#) = (1 << 0) }
- Host mode.*
- enum [AAEHostMode](#) { [AAEHostModeShow](#) = [AAEHostModeBitsLive](#), [AAEHostModeConfig](#) = AAEX_eHostModeBitsNone }

DEPRECATED.

- enum [AAEPriDataOptions](#) { [AAEPriDataOptionsDefaultOptions](#) = 0, [AAEPriDataOptionsKeepOnReset](#) = (1 << 0), [AAEPriDataOptionsExternal](#) = (1 << 1), [AAEPriDataOptionsAlign8](#) = (1 << 2), [AAEPriDataOptionsINT32_MAX](#) = [AAINT32_MAX](#) }

Options for algorithm private data fields.

- enum [AAEConstraintLocationMask](#) { [AAEConstraintLocationMaskNone](#) = 0, [AAEConstraintLocationMaskDataModel](#) = (1 << 0), [AAEConstraintLocationMaskDLLChipAffinity](#) = (1 << 1), [AAEConstraintLocationMaskFixedLatencyDomain](#) = (1 << 2) }

Property values to describe location constraints placed on the plug-in's algorithm component (ProcessProc)

- enum [AAEConstraintTopology](#) { [AAEConstraintTopologyNone](#) = 0, [AAEConstraintTopologyMonolithic](#) = 1 }

Property values to describe the topology of the plug-in's modules (e.g. data model, GUI.)

- enum [AAEComponentInstanceInitAction](#) { [AAEComponentInstanceInitActionAddingNewInstance](#) = 0, [AAEComponentInstanceInitActionRemovingInstance](#) = 1, [AAEComponentInstanceInitActionResetInstance](#) = 2 }

Selector indicating the action that occurred to prompt a component initialization callback.

- enum [AAESampleRateMask](#) { [AAESampleRateMaskNo](#) = 0, [AAESampleRateMask44100](#) = (1 << 0), [AAESampleRateMask48000](#) = (1 << 1), [AAESampleRateMask88200](#) = (1 << 2), [AAESampleRateMask96000](#) = (1 << 3), [AAESampleRateMask176400](#) = (1 << 4), [AAESampleRateMask192000](#) = (1 << 5), [AAESampleRateMaskAll](#) = [AAINT32_MAX](#) }

Property values to describe various sample rates.

- enum [AAEParameterType](#) { [AAEParameterTypeDiscrete](#), [AAEParameterTypeContinuous](#) }

FIC stuff that I can't include without DAE library dependence.

- enum [AAEParameterOrientationBits](#) { [AAEParameterOrientationDefault](#) = 0, [AAEParameterOrientationBottomMinTopMax](#) = 0, [AAEParameterOrientationTopMinBottomMax](#) = 1, [AAEParameterOrientationLeftMinRightMax](#) = 0, [AAEParameterOrientationRightMinLeftMax](#) = 2, [AAEParameterOrientationRotarySingleDotMode](#) = 0, [AAEParameterOrientationRotaryBoostCutMode](#) = 4, [AAEParameterOrientationRotaryWrapMode](#) = 8, [AAEParameterOrientationRotarySpreadMode](#) = 12, [AAEParameterOrientationRotaryLeftMinRightMax](#) = 0, [AAEParameterOrientationRotaryRightMinLeftMax](#) = 16 }

Visual Orientation of a parameter.

- enum `AAX_EParameterValueInfoSelector` { `AAX_ePageTable_EQ_Band_Type` = 0, `AAX_ePageTable_EQ_InCircuitPolarity` = 1, `AAX_ePageTable_UseAlternateControl` = 2 }

Query type selectors for use with `AAX_IEffectParameters::GetParameterValueInfo()`

- enum `AAX_EQBandTypes` { `AAX_eEQBandType_HighPass` = 0, `AAX_eEQBandType_LowShelf` = 1, `AAX_eEQBandType_Parabolic` = 2, `AAX_eEQBandType_HighShelf` = 3, `AAX_eEQBandType_LowPass` = 4, `AAX_eEQBandType_Notch` = 5 }

Definitions of band types for EQ page table.

- enum `AAX_EQInCircuitPolarity` { `AAX_eEQInCircuitPolarity_Enabled` = 0, `AAX_eEQInCircuitPolarity_Bypassed` = 1, `AAX_eEQInCircuitPolarity_Disabled` = 2 }

Definitions for band in/out for EQ page table.

- enum `AAX_EUseAlternateControl` { `AAX_eUseAlternateControl_No` = 0, `AAX_eUseAlternateControl_Yes` = 1 }

Definitions for Use Alternate Control parameter.

- enum `AAX_EMIDINodeType` { `AAX_eMIDINodeType_LocalInput` = 0, `AAX_eMIDINodeType_LocalOutput` = 1, `AAX_eMIDINodeType_Global` = 2, `AAX_eMIDINodeType_Transport` = 3 }

MIDI node types.

- enum `AAX_EUpdateSource` { `AAX_eUpdateSource_Unspecified` = 0, `AAX_eUpdateSource_Parameter` = 1, `AAX_eUpdateSource_Chunk` = 2, `AAX_eUpdateSource_Delay` = 3 }

Source for values passed into `UpdateParameterNormalizedValue()`.

- enum `AAX_EDataInPortType` { `AAX_eDataInPortType_Unbuffered` = 0, `AAX_eDataInPortType_Buffered` = 1, `AAX_eDataInPortType_Incremental` = 2 }

Property value for whether a data in port should be buffered or not.

- enum `AAX_EFrameRate` { `AAX_eFrameRate_Undeclared` = 0, `AAX_eFrameRate_24Frame`, `AAX_eFrameRate_25Frame`, `AAX_eFrameRate_2997NonDrop`, `AAX_eFrameRate_2997DropFrame`, `AAX_eFrameRate_30NonDrop`, `AAX_eFrameRate_30DropFrame`, `AAX_eFrameRate_23976` }

FrameRate types.

- enum `AAX_EFootFramesRate` { `AAX_eFootFramesRate_23976` = 0, `AAX_eFootFramesRate_24`, `AAX_eFootFramesRate_25` }

FootFramesRate types.

- enum `AAX_EMidiGlobalNodeSelectors` { `AAX_eMIDIClick` = 1 << 0, `AAX_eMIDIMtc` = 1 << 1, `AAX_eMIDIBeatClock` = 1 << 2 }

The Global MIDI Node Selectors.

- enum `AAX_EPreviewState` { `AAX_ePreviewState_Stop` = 0, `AAX_ePreviewState_Start` = 1 }

Offline preview states for use with `AAX_eNotificationEvent_ASPreviewState`.

- enum `AAX_EProcessingState` { `AAX_eProcessingState_StopPass` = 2, `AAX_eProcessingState_StartPass` = 3, `AAX_eProcessingState_EndPassGroup` = 4, `AAX_eProcessingState_BeginPassGroup` = 5, `AAX_eProcessingState_Stop` = `AAX_eProcessingState_StopPass`, `AAX_eProcessingState_Start` = `AAX_eProcessingState_StartPass` }

Offline preview states for use with `AAX_eNotificationEvent_ASProcessingState`.

- enum `AAX_ETargetPlatform` { `kAAX_eTargetPlatform_None` = 0, `kAAX_eTargetPlatform_Native` = 1, `kAAX_eTargetPlatform_TI` = 2, `kAAX_eTargetPlatform_External` = 3, `kAAX_eTargetPlatform_Count` = 4 }

Describes what platform the component runs on.

- enum `AAX_ESupportLevel` { `AAX_eSupportLevel_Uninitialized` = 0, `AAX_eSupportLevel_Unsupported` = 1, `AAX_eSupportLevel_Supported` = 2, `AAX_eSupportLevel_Disabled` = 3, `AAX_eSupportLevel_ByProperty` = 4 }
- enum `AAX_EHostLevel` { `AAX_eHostLevel_Unknown` = 0, `AAX_eHostLevel_Standard` = 1, `AAX_eHostLevel_Entry` = 2, `AAX_eHostLevel_Intermediate` = 3 }

Host levels.

- enum `AAX_ETextEncoding` { `AAX_eTextEncoding_Undefined` = -1, `AAX_eTextEncoding_UTF8` = 0, `AAX_eTextEncoding_Num` }

Describes possible string encodings.

- enum `AAX_EAssertFlags` { `AAX_eAssertFlags_Default` = 0, `AAX_eAssertFlags_Log` = 1 << 0, `AAX_eAssertFlags_Dialog` = 1 << 1 }

Flags for use with `AAX_IHostServices::HandleAssertFailure()`

Functions

- [AAX_ENUM_SIZE_CHECK \(AAE_EHighlightColor\)](#)
- [AAX_ENUM_SIZE_CHECK \(AAE_ETracePriorityHost\)](#)
- [AAX_ENUM_SIZE_CHECK \(AAE_ETracePriorityDSP\)](#)
- [AAX_ENUM_SIZE_CHECK \(AAE_EModifiers\)](#)
- [AAX_ENUM_SIZE_CHECK \(AAE_EAudioBufferLength\)](#)
- [AAX_ENUM_SIZE_CHECK \(AAE_EAudioBufferLengthDSP\)](#)
- [AAX_ENUM_SIZE_CHECK \(AAE_EAudioBufferLengthNative\)](#)
- [AAX_ENUM_SIZE_CHECK \(AAE_EMaxAudioSuiteTracks\)](#)
- [AAX_ENUM_SIZE_CHECK \(AAE_EStemFormat\)](#)
- [AAX_ENUM_SIZE_CHECK \(AAE_EPlugInCategory\)](#)
- [AAX_ENUM_SIZE_CHECK \(AAE_EPlugInStrings\)](#)
- [AAX_ENUM_SIZE_CHECK \(AAE_EMeterOrientation\)](#)
- [AAX_ENUM_SIZE_CHECK \(AAE_EMeterBallisticType\)](#)
- [AAX_ENUM_SIZE_CHECK \(AAE_EMeterType\)](#)
- [AAX_ENUM_SIZE_CHECK \(AAE_ECurveType\)](#)
- [AAX_ENUM_SIZE_CHECK \(AAE_EResourceType\)](#)
- [AAX_ENUM_SIZE_CHECK \(AAE_ENotificationEvent\)](#)
- [AAX_ENUM_SIZE_CHECK \(AAE_EHostModeBits\)](#)
- [AAX_ENUM_SIZE_CHECK \(AAE_EHostMode\)](#)
- [AAX_ENUM_SIZE_CHECK \(AAE_EPrivateDataOptions\)](#)
- [AAX_ENUM_SIZE_CHECK \(AAE_EConstraintLocationMask\)](#)
- [AAX_ENUM_SIZE_CHECK \(AAE_EConstraintTopology\)](#)
- [AAX_ENUM_SIZE_CHECK \(AAE_EComponentInstanceInitAction\)](#)
- [AAX_ENUM_SIZE_CHECK \(AAE_ESampleRateMask\)](#)
- [AAX_ENUM_SIZE_CHECK \(AAE_EParameterType\)](#)
- [AAX_ENUM_SIZE_CHECK \(AAE_EParameterOrientationBits\)](#)
- [AAX_ENUM_SIZE_CHECK \(AAE_EParameterValueInfoSelector\)](#)
- [AAX_ENUM_SIZE_CHECK \(AAE_EQBandTypes\)](#)
- [AAX_ENUM_SIZE_CHECK \(AAE_EQInCircuitPolarity\)](#)
- [AAX_ENUM_SIZE_CHECK \(AAE_EUseAlternateControl\)](#)
- [AAX_ENUM_SIZE_CHECK \(AAE_EMIDINodeType\)](#)
- [AAX_ENUM_SIZE_CHECK \(AAE_EUpdateSource\)](#)
- [AAX_ENUM_SIZE_CHECK \(AAE_EDataInPortType\)](#)
- [AAX_ENUM_SIZE_CHECK \(AAE_EFrameRate\)](#)
- [AAX_ENUM_SIZE_CHECK \(AAE_EFeetFramesRate\)](#)
- [AAX_ENUM_SIZE_CHECK \(AAE_EMidiGlobalNodeSelectors\)](#)
- [AAX_ENUM_SIZE_CHECK \(AAE_EPreviewState\)](#)
- [AAX_ENUM_SIZE_CHECK \(AAE_EProcessingState\)](#)
- [AAX_ENUM_SIZE_CHECK \(AAE_ETargetPlatform\)](#)
- [AAX_ENUM_SIZE_CHECK \(AAE_ESupportLevel\)](#)
- [AAX_ENUM_SIZE_CHECK \(AAE_EHostLevel\)](#)
- [AAX_ENUM_SIZE_CHECK \(AAE_ETextEncoding\)](#)
- [AAX_ENUM_SIZE_CHECK \(AAE_EAssertFlags\)](#)

15.58.2 Macro Definition Documentation

```

15.58.2.1 #define AAX_INT32_MIN (-2147483647 - 1) /* minimum signed 32 bit value */

15.58.2.2 #define AAX_INT32_MAX 2147483647 /* maximum signed 32 bit value */

15.58.2.3 #define AAX_UINT32_MIN 0U /* minimum unsigned 32 bit value */

15.58.2.4 #define AAX_UINT32_MAX 4294967295U /* maximum unsigned 32 bit value */

15.58.2.5 #define AAX_INT16_MIN (-32767 - 1) /* minimum signed 16 bit value */

15.58.2.6 #define AAX_INT16_MAX 32767 /* maximum signed 16 bit value */

15.58.2.7 #define AAX_UINT16_MIN 0U /* minimum unsigned 16 bit value */

15.58.2.8 #define AAX_UINT16_MAX 65535U /* maximum unsigned 16 bit value */

15.58.2.9 #define AAX_ENUM_SIZE_CHECK( x ) extern int __enumSizeCheck[ 2*(sizeof(uint32_t)==sizeof(x)) - 1 ]

```

Macro to ensure enum type consistency across binaries.

```

15.58.2.10 #define AAX_STEM_FORMAT( aIndex, aChannelCount ) ( static_cast<uint32_t>( ( static_cast<uint16_t>(aIndex)
    << 16 ) | ( (aChannelCount >= AAX_UINT16_MIN) && (aChannelCount <= 0xFFFF) ? aChannelCount & 0xFFFF
    : 0x0000 ) ) )

15.58.2.11 #define AAX_STEM_FORMAT_CHANNEL_COUNT( aStemFormat ) ( static_cast<uint16_t>( aStemFormat &
    0xFFFF ) )

15.58.2.12 #define AAX_STEM_FORMAT_INDEX( aStemFormat ) ( static_cast<int16_t>( ( aStemFormat >> 16 ) & 0xFFFF ) )

```

15.58.3 Typedef Documentation

```
15.58.3.1 typedef enum AAX_EParameterType AAX_EParameterType
```

FIC stuff that I can't include without DAE library dependence.

Legacy Porting Notes Values must match unnamed type enum in FicTDMControl.h

Todo FLAGGED FOR REMOVAL

```
15.58.3.2 typedef int32_t AAX_EParameterOrientation
```

Typedef for a bitfield of [AAX_EParameterOrientationBits](#) values.

15.58.4 Enumeration Type Documentation

```
15.58.4.1 enum AAX_EHighlightColor
```

Highlight color selector.

See also

[AAX_IEffectGUI::SetControlHighlightInfo\(\)](#)

Enumerator

AAX_eHighlightColor_Red
AAX_eHighlightColor_Blue
AAX_eHighlightColor_Green
AAX_eHighlightColor_Yellow
AAX_eHighlightColor_Num

15.58.4.2 enum AAX_ETracePriorityHost

Platform-specific tracing priorities.

Use the generic EAAX_Trace_Priority in plug-ins for cross-platform tracing (see [AAX_Assert.h](#))

Enumerator

AAX_eTracePriorityHost_None
AAX_eTracePriorityHost_High
AAX_eTracePriorityHost_Normal
AAX_eTracePriorityHost_Low
AAX_eTracePriorityHost_Lowest

15.58.4.3 enum AAX_ETracePriorityDSP

Platform-specific tracing priorities.

Use the generic EAAX_Trace_Priority in plug-ins for cross-platform tracing (see [AAX_Assert.h](#))

Enumerator

AAX_eTracePriorityDSP_None
AAX_eTracePriorityDSP_Assert
AAX_eTracePriorityDSP_High
AAX_eTracePriorityDSP_Normal
AAX_eTracePriorityDSP_Low

15.58.4.4 enum AAX_EModifiers

Modifier key definitions used by AAX API.

Enumerator

AAX_eModifiers_None
AAX_eModifiers_Shift Shift.
AAX_eModifiers_Control Control on Mac, Winkey/Start on PC.
AAX_eModifiers_Option Option on Mac, Alt on PC.
AAX_eModifiers_Command Command on Mac, Ctrl on PC.
AAX_eModifiers_SecondaryButton Secondary mouse button.
AAX_eModifiers_Alt Option on Mac, Alt on PC.
AAX_eModifiers_Cntl Command on Mac, Cntl on PC.
AAX_eModifiers_WINKEY Control on Mac, WINKEY on PC.

15.58.4.5 enum AAX_EAudioBufferLength

Generic buffer length definitions.

These enum values can be used to calculate literal values as powers of two:

```
1 (1 << AAX_eAudioBufferLength_16) == 16;
```

See also

[AAX_EAudioBufferLengthDSP](#)
[AAE_EAudioBufferLengthNative](#)

Enumerator

AAX_eAudioBufferLength_Undefined
AAX_eAudioBufferLength_1
AAX_eAudioBufferLength_2
AAX_eAudioBufferLength_4
AAX_eAudioBufferLength_8
AAX_eAudioBufferLength_16
AAX_eAudioBufferLength_32
AAX_eAudioBufferLength_64
AAX_eAudioBufferLength_128
AAX_eAudioBufferLength_256
AAX_eAudioBufferLength_512
AAX_eAudioBufferLength_1024

AAX_eAudioBufferLength_Max Maximum buffer length for ProcessProc processing buffers. Audio buffers for other methods, such as the high-latency render callback for [AAX Hybrid](#) or the offline render callback for [Host Processor](#) effects, may contain more samples than AAX_eAudioBufferLength_Max.

15.58.4.6 enum AAX_EAudioBufferLengthDSP

Currently supported processing buffer length definitions for AAX DSP hosts.

AAX DSP decks must support at least these buffer lengths. All AAX DSP algorithm ProcessProcs must support exactly one of these buffer lengths.

See also

[AAX_eProperty_DSP_AudioBufferLength](#)

Enumerator

AAX_eAudioBufferLengthDSP_Default
AAX_eAudioBufferLengthDSP_4
AAX_eAudioBufferLengthDSP_16
AAX_eAudioBufferLengthDSP_32
AAX_eAudioBufferLengthDSP_64
AAX_eAudioBufferLengthDSP_Max

15.58.4.7 enum AAE_EAudioBufferLengthNative

Processing buffer length definitions for Native AAX hosts.

All AAX Native plug-ins must support variable buffer lengths. The buffer lengths that a host will use are constrained by the values in this enum. All Native buffer lengths will be powers of two, as per [AAX_EAudioBufferLength](#)

See also

[AAX_eProperty_DSP_AudioBufferLength](#)

Enumerator

AAX_eAudioBufferLengthNative_Min Minimum Native buffer length.

AAX_eAudioBufferLengthNative_Max Maximum Native buffer length.

15.58.4.8 enum AAX_EMaxAudioSuiteTracks

The maximum number of tracks that an AAX host will process in a non-real-time context.

See also

[AAX_eProperty_NumberOfInputs](#) and [AAX_eProperty_NumberOfOutputs](#)

Enumerator

AAX_eMaxAudioSuiteTracks

15.58.4.9 enum AAX_EStemFormat

Stem format definitions.

A stem format combines a channel count with a semantic meaning for each channel. Usually this is the speaker or speaker position associated with the data in the channel. The meanings of each channel in each stem format (i.e. channel orders) are listed below.

Not all stem formats are supported by all AAX plug-in hosts. An effect may describe support for any stem format combination which it supports and the host will ignore any configurations which it cannot support.

Note

When defining stem format support in [AAX_IHostProcessor](#) effects do not use stem format properties or values. Instead, use [AAX_eProperty_NumberOfInputs](#) and [AAX_eProperty_NumberOfOutputs](#) with integer channel count values.

See also

- [AAX_eProperty_InputStemFormat](#)
- [AAX_eProperty_OutputStemFormat](#)
- [AAX_eProperty_HybridInputStemFormat](#)
- [AAX_eProperty_HybridOutputStemFormat](#)
- [AAX_eProperty_SideChainStemFormat](#)

Enumerator

AAX_eStemFormat_Mono M.

AAX_eStemFormat_Stereo L R.

AAX_eStemFormat_LCR L C R.

AAX_eStemFormat_LCRS L C R S.

AAX_eStemFormat_Quad L R Ls Rs.

AAX_eStemFormat_5_0 L C R Ls Rs.

AAX_eStemFormat_5_1 L C R Ls Rs LFE.

AAX_eStemFormat_6_0 L C R Ls Cs Rs.

AAX_eStemFormat_6_1 L C R Ls Cs Rs LFE.

AAX_eStemFormat_7_0_SDDS L Lc C Rc R Ls Rs.

AAX_eStemFormat_7_1_SDDS L Lc C Rc R Ls Rs LFE.

AAX_eStemFormat_7_0_DTS L C R Lss Rss Lsr Rsr.

AAX_eStemFormat_7_1_DTS L C R Lss Rss Lsr Rsr LFE.

AAX_eStemFormat_7_0_2 L C R Lss Rss Lsr Rsr Lts Rts.

AAX_eStemFormat_7_1_2 L C R Lss Rss Lsr Rsr LFE Lts Rts.

AAX_eStemFormat_Ambi_1_ACN Reserved for Ambisonics: first-order with ACN channel order and SN3D (AmbiX) normalization.

AAX_eStemFormat_Ambi_2_ACN Reserved for Ambisonics: second-order with ACN channel order and SN3D (AmbiX) normalization.

AAX_eStemFormat_Ambi_3_ACN Reserved for Ambisonics: third-order with ACN channel order and SN3D (AmbiX) normalization.

AAX_eStemFormat_Reserve_1 Reserved - do not use.

AAX_eStemFormat_Reserve_2 Reserved - do not use.

AAX_eStemFormat_Reserve_3 Reserved - do not use.

AAX_eStemFormatNum

AAX_eStemFormat_None

AAX_eStemFormat_Any

AAX_eStemFormat_INT32_MAX

15.58.4.10 enum AAX_EPlugInCategory

Effect category definitions.

Used with [AAX_IEffectDescriptor::AddCategory\(\)](#) to categorize an Effect.

These values are bitwise-exclusive and may be used in a bitmask to define multiple categories:

```
1 myCategory = AAX_ePlugInCategory_EQ | AAX_ePlugInCategory_Dynamics;
```

Note

The host may handle plug-ins with different categories in different manners, e.g. replacing "analyze" with "reverse" for offline processing of delays and reverbs.

Enumerator

AAX_ePlugInCategory_None

AAX_ePlugInCategory_EQ Equalization.

AAX_ePlugInCategory_Dynamics Compressor, expander, limiter, etc.

AAX_ePlugInCategory_PitchShift Pitch processing.

AAX_ePlugInCategory_Reverb Reverberation and room/space simulation.

AAX_ePlugInCategory_Delay Delay and echo.

AAX_ePlugInCategory_Modulation Phasing, flanging, chorus, etc.

AAX_ePlugInCategory_Harmonic Distortion, saturation, and harmonic enhancement.

AAX_ePlugInCategory_NoiseReduction Noise reduction.

AAX_ePlugInCategory_Dither Dither, noise shaping, etc.

AAX_ePlugInCategory_SoundField Pan, auto-pan, upmix and downmix, and surround handling.

AAX_ePlugInCategory_HWGenerators Fixed hardware audio sources such as SampleCell.

AAX_ePlugInCategory_SWGenerators Virtual instruments, metronomes, and other software audio sources.

AAX_ePlugInCategory_WrappedPlugin All plug-ins wrapped by a third party wrapper (i.e. VST to RTAS wrapper), except for VI plug-ins which should be mapped to AAX_PlugInCategory_SWGenerators.

AAX_EPlugInCategory_Effect Special effects.

AAX_ePlugInCategory_Example

AAX_ePlugInCategory_INT32_MAX

15.58.4.11 enum AAX_EPlugInStrings

Effect string identifiers.

The AAX host may associate certain plug-in display strings with these identifiers.

See also

[AAX_IEffectGUI::GetCustomLabel\(\)](#)

Enumerator

AAX_ePlugInStrings_Analysis "Analyze" button label (AudioSuite)
Legacy Porting Notes Was pluginStrings_Analysis in the RTAS/TDM SDK

AAX_ePlugInStrings_MonoMode "Mono Mode" selector label (AudioSuite)
Legacy Porting Notes Was pluginStrings_MonoMode in the RTAS/TDM SDK

AAX_ePlugInStrings_MultiInputMode "Multi-Input Mode" selector label (AudioSuite)
Legacy Porting Notes Was pluginStrings_MultiInputMode in the RTAS/TDM SDK

AAX_ePlugInStrings_RegionByRegionAnalysis "Clip-by-Clip Analysis" selector label (AudioSuite)
Legacy Porting Notes Was pluginStrings_RegionByRegionAnalysis in the RTAS/TDM SDK

AAX_ePlugInStrings_AllSelectedRegionsAnalysis "Whole File Analysis" selector label (AudioSuite)
Legacy Porting Notes Was pluginStrings_AllSelectedRegionsAnalysis in the RTAS/TDM SDK

AAX_ePlugInStrings_RegionName **Deprecated**

AAX_ePlugInStrings_ClipName Clip name label (AudioSuite). This value will replace the clip's name.

See also

[AAX_ePlugInStrings_ClipNameSuffix](#)

Legacy Porting Notes Was pluginStrings_RegionName in the RTAS/TDM SDK

AAX_ePlugInStrings_Progress Progress bar label (AudioSuite)
Host Compatibility Notes Not currently supported by Pro Tools
Legacy Porting Notes Was pluginStrings_Progress in the RTAS/TDM SDK

AAX_ePlugInStrings_PluginFileName **Deprecated**

AAX_ePlugInStrings_Preview **Deprecated**

AAX_ePlugInStrings_Process "Render" button label (AudioSuite)

Legacy Porting Notes Was pluginStrings_Process in the RTAS/TDM SDK

AAX_ePlugInStrings_Bypass "Bypass" button label (AudioSuite)

Legacy Porting Notes Was pluginStrings_Bypass in the RTAS/TDM SDK

AAX_ePlugInStrings_ClipNameSuffix Clip name label suffix (AudioSuite). This value will be appended to the clip's name, vs [AAX_ePlugInStrings_ClipName](#) which will replace the clip's name completely.

AAX_ePlugInStrings_INT32_MAX

15.58.4.12 enum AAX_EMeterOrientation

Meter orientation.

Use this enum in conjunction with the [AAX_eProperty_Meter_Orientation](#) property

For more information about meters in [AAX](#), see [Plug-in meters](#)

Enumerator

AAX_eMeterOrientation_Default

AAX_eMeterOrientation_BottomLeft the default orientation

AAX_eMeterOrientation_TopRight Some dynamics plug-in orient their gain reduction like so.

AAX_eMeterOrientation_Center A plug-in that does gain increase and decrease may want this. meter values less than 0x40000000 would display downward from the mid-point. meter values greater than 0x40000000 would display upward from the mid-point.

AAX_eMeterOrientation_PhaseDot linear scale, displays 2 dots around the value (currently D-Control only)

15.58.4.13 enum AAX_EMeterBallisticType

Meter ballistics type.

Use this enum in conjunction with the [AAX_eProperty_Meter_Ballistics](#) property

For more information about meters in [AAX](#), see [Plug-in meters](#)

Enumerator

AAX_eMeterBallisticType_Host The ballistics follow the host settings.

AAX_eMeterBallisticType_NoDecay No decay ballistics.

15.58.4.14 enum AAX_EMeterType

Meter type.

Use this enum in conjunction with the [AAX_eProperty_Meter_Type](#) property

For more information about meters in [AAX](#), see [Plug-in meters](#)

Enumerator

AAX_eMeterType_Input e.g. Your typical input meter (possibly after an input gain stage)

AAX_eMeterType_Output e.g. Your typical output meter (possibly after an output gain stage)

AAX_eMeterType_CLGain e.g. Compressor/Limiter gain reduction

AAX_eMeterType_EGGain e.g. Expander/Gate gain reduction

AAX_eMeterType_Analysis e.g. multi-band amplitude from a Spectrum analyzer

AAX_eMeterType_Other e.g. a meter that does not fit in any of the above categories

AAX_eMeterType_None For internal host use only.

15.58.4.15 enum AAX_EResourceType

Types of resources that can be added to an Effect's description.

See also

[AAX_IEffectDescriptor::AddResourceInfo\(\)](#)

Enumerator

AAX_eResourceType_None

AAX_eResourceType_PageTable The file name of the page table xml file

AAX_eResourceType_PageTableDir The absolute path to the directory containing the plug-in's page table xml file(s)

Defaults to *.aaxplugin/Contents/Resources

15.58.4.16 enum AAX_ENotificationEvent

Events IDs for AAX notifications.

- Notifications listed with *Sent by: Host* are dispatched by the AAX host and may be received in [AAX_IEffectParameters::NotificationReceived\(\)](#) and/or [AAX_IEffectGUI::NotificationReceived\(\)](#). Note that only one of these two components may be registered to receive a notification.
- Notifications listed with *Sent by: Plug-in* are dispatched by the AAX plug-in to notify the host that some event has occurred.

Note

All 'AX__' four-char IDs are reserved for the AAX specification

Enumerator

AAX_eNotificationEvent_InsertPositionChanged (not currently sent) The zero-indexed insert position of this plug-in instance within its track *Data: int32_t*
Sent by: Host

AAX_eNotificationEvent_TrackNameChanged (const [AAX_IString](#)) The current name of this plug-in instance's track

Host Compatibility Notes Supported in Pro Tools 11.2 and higher
 Not supported by Media Composer

Data: const AAX_IString

Sent by: Host

AAX_eNotificationEvent_TrackUIDChanged (not currently sent) The current UID of this plug-in instance's track *Data: const uint8_t[16]*
Sent by: Host

AAX_eNotificationEvent_TrackPositionChanged (not currently sent) The current position index of this plug-in instance's track *Data: int32_t*
Sent by: Host

AAX_eNotificationEvent_AlgorithmMoved Not currently sent. *Data: none*
Sent by: Host

AAX_eNotificationEvent_GUIOpened Not currently sent. *Data: none*
Sent by: Host

AAX_eNotificationEvent_GUIClosed Not currently sent. *Data: none*
Sent by: Host

AAX_eNotificationEvent_ASProcessingState AudioSuite processing state change notification. One of [AAX_EProcessingState](#).

Host Compatibility Notes Supported in Pro Tools 11 and higher
 Not supported by Media Composer

Data: `int32_t`
Sent by: Host

AAX_eNotificationEvent_ASPreviewState AudioSuite preview state change notification. One of [AAX_EPreviewState](#).

Legacy Porting Notes Replacement for `SetPreviewState()`

Host Compatibility Notes Supported in Pro Tools 11 and higher
 Not supported by Media Composer

Data: `int32_t`
Sent by: Host

AAX_eNotificationEvent_SessionBeingOpened Tell the plug-in that chunk data is coming from a PTX.

Host Compatibility Notes Supported in Pro Tools 11 and higher
 Not supported by Media Composer

Data: `none`
Sent by: Host

AAX_eNotificationEvent_PresetOpened Tell the plug-in that chunk data is coming from a TFX.

Host Compatibility Notes Supported in Pro Tools 11 and higher

Data: `none`
Sent by: Host

AAX_eNotificationEvent_EnteringOfflineMode Entering offline processing mode (i.e. offline bounce)

Host Compatibility Notes Supported in Pro Tools 11 and higher

Data: `none`
Sent by: Host

AAX_eNotificationEvent_ExitingOfflineMode Exiting offline processing mode (i.e. offline bounce)

Host Compatibility Notes Supported in Pro Tools 11 and higher

Data: `none`
Sent by: Host

AAX_eNotificationEvent_SessionPathChanged A string representing the path of the current session.

Host Compatibility Notes Supported in Pro Tools 11.1 and higher

Data: `const AAX_IString`
Sent by: Host

AAX_eNotificationEvent_SignalLatencyChanged The host has changed its latency compensation for this plug-in instance.

Note

This notification may be sent redundantly just after plug-in instantiation when the [AAX_eProperty_LatencyContribution](#) property is described.

Host Compatibility Notes Supported in Pro Tools 11.1 and higher

Data: `none`
Sent by: Host

AAX_eNotificationEvent_DelayCompensationState The host's delay compensation state has changed. This notification refers to the host's delay compensation feature as a whole, rather than the specific delay compensation state for the plug-in.

Possible values: 0 (disabled), 1 (enabled)

Plug-ins may need to monitor the host's delay compensation state because, while delay compensation is disabled, the host will never change the plug-in's accounted latency and, therefore, will never dispatch [AAX_eNotificationEvent_SignalLatencyChanged](#) to the plug-in following a call to [AAX_IController::SetSignalLatency\(\)](#).

Host Compatibility Notes Supported in Pro Tools 12.6 and higher

Data: `int32_t`
Sent by: Host

AAX_eNotificationEvent_CycleCountChanged (not currently sent) The host has changed its DSP cycle allocation for this plug-in instance *Data: none*
Sent by: Host

AAX_eNotificationEvent_MaxViewSizeChanged Tell the plug-in the maximum allowed GUI dimensions.

Host Compatibility Notes Supported in Pro Tools 11.1 and higher

Data: const AAX_Point

Sent by: Host

AAX_eNotificationEvent_SideChainBeingConnected Tell the plug-in about connection of the sidechain input.

Host Compatibility Notes Supported in Pro Tools 11.1 and higher

Data: none

Sent by: Host

AAX_eNotificationEvent_SideChainBeingDisconnected Tell the plug-in about disconnection of the sidechain input.

Host Compatibility Notes Supported in Pro Tools 11.1 and higher

Data: none

Sent by: Host

AAX_eNotificationEvent_NoiseFloorChanged The plug-in's noise floor level. The notification data is the new absolute noise floor level generated by the plug-in, as amplitude. For example, a plug-in generating a noise floor at -80 dB (amplitude) would provide 0.0001 in the notification data.

Signal below the level of the plug-in's noise floor may be ignored by host features such as Dynamic Plug-In Processing, which detect whether or not there is any signal being generated by the plug-in

Data: double

Sent by: Plug-in

AAX_eNotificationEvent_ParameterMappingChanged Notify the host that some aspect of the parameters' mapping has changed. To respond to this notification, the host will call [AAX_IEffectParameters::UpdatePageTable\(\)](#) to update its cached page tables.

Data: none

Sent by: Plug-in

AAX_eNotificationEvent_HostModeChanged Notify the plug-in about Host mode changing.

Host Compatibility Notes Supported in Venue 5.6 and higher

Data: AAX_EHostModeBits

Sent by: Host

15.58.4.17 enum AAX_EHostModeBits

Host mode.

Host Compatibility Notes Supported in Venue 5.6 and higher

Enumerator

AAX_eHostModeBits_None No special host mode, e.g. Pro Tools normal operation, Venue Config mode.

AAX_eHostModeBits_Live The host is in a live playback mode, e.g. Venue Show mode - inserts are live and must not allow state changes which interrupt audio processing.

15.58.4.18 enum AAX_EHostMode

DEPRECATED.

Use [AAX_EHostModeBits](#)

Warning

The values of these modes have changed as of AAX SDK 2.3.1 from the definitions originally published in AAX SDK 2.3.0

Deprecated This enum is deprecated and will be removed in a future release.

Enumerator

AAX_eHostMode_Show **Deprecated** Use [AAX_eHostModeBits_Live](#)

AAX_eHostMode_Config **Deprecated** Use [AAX_eHostModeBits_None](#)

15.58.4.19 enum AAX_EPrivateDataOptions

Options for algorithm private data fields.

Enumerator

AAX_ePrivateDataOptions_DefaultOptions

AAX_ePrivateDataOptions_KeepOnReset Retain data upon plug-in reset.

Warning

Not currently implemented. If this functionality is desired, the recommended workaround is to cache the desired private data to be set during [AAX_IEffectParameters::ResetFieldData\(\)](#).

AAX_ePrivateDataOptions_External Place the block in external memory (internal by default)

AAX_ePrivateDataOptions_Align8 Place the block in mem aligned by 64 bits.

AAX_ePrivateDataOptions_INT32_MAX

15.58.4.20 enum AAX_EConstraintLocationMask

Property values to describe location constraints placed on the plug-in's algorithm component ([ProcessProc](#))

See also

[AAX_eProperty_Constraint_Location](#)

Enumerator

AAX_eConstraintLocationMask_None No constraint placed on component's location.

AAX_eConstraintLocationMask_DataModel This [ProcessProc](#) must be co-located with the plug-in's data model object.

AAX_eConstraintLocationMask_DLLChipAffinity This [ProcessProc](#) should be instantiated on the same chip as other effects that use the same DLL.

- This constraint is only applicable to DSP algorithms

AAX_eConstraintLocationMask_FixedLatencyDomain This [ProcessProc](#) may not be "moved" between different latency domains; it must always receive audio buffers of the same size over its lifetime.

- Different instances of the [ProcessProc](#) may receive buffers at different fixed sizes, but the buffers received by any particular [ProcessProc](#) will not change.
- This constraint only applies to Native algorithms. DSP processing always occurs at a fixed size.

Host Compatibility Notes This constraint is not currently supported by any AAX host

15.58.4.21 enum AAX_EConstraintTopology

Property values to describe the topology of the plug-in's modules (e.g. data model, GUI.)

See also

[AAX_eProperty_Constraint_Topo](#)

Enumerator

AAX_eConstraintTopology_None No constraint placed on plug-in's topology.

AAX_eConstraintTopology_Monolithic All plug-in modules (e.g. data model, GUI) must be co-located and non-relocatable.

15.58.4.22 enum AAX_EComponentInstanceInitAction

Selector indicating the action that occurred to prompt a component initialization callback.

See also

[AAX_CInstanceInitProc](#)

Enumerator

AAX_eComponentInstanceInitAction_AddingNewInstance

AAX_eComponentInstanceInitAction_RemovingInstance

AAX_eComponentInstanceInitAction_ResetInstance

15.58.4.23 enum AAX_ESampleRateMask

Property values to describe various sample rates.

These values may be used as a bitmask, so e.g. a particular Effect may declare compatibility with `AAX_eSampleRateMask_44100 | AAX_eSampleRateMask_48000`

See also

[AAX_eProperty_SampleRate](#)

Enumerator

AAX_eSampleRateMask_No

AAX_eSampleRateMask_44100

AAX_eSampleRateMask_48000

AAX_eSampleRateMask_88200

AAX_eSampleRateMask_96000

AAX_eSampleRateMask_176400

AAX_eSampleRateMask_192000

AAX_eSampleRateMask_All

15.58.4.24 enum AAX_EParameterType

FIC stuff that I can't include without DAE library dependence.

Legacy Porting Notes Values must match unnamed type enum in FicTDMControl.h

Todo FLAGGED FOR REMOVAL

Enumerator

AAX_eParameterType_Discrete [Legacy Porting Notes](#) Matches kDAE_DiscreteValues
AAX_eParameterType_Continuous [Legacy Porting Notes](#) Matches kDAE_ContinuousValues

15.58.4.25 enum AAX_EParameterOrientationBits

Visual Orientation of a parameter.

Todo FLAGGED FOR REVISION

Enumerator

AAX_eParameterOrientation_Default
AAX_eParameterOrientation_BottomMinTopMax
AAX_eParameterOrientation_TopMinBottomMax
AAX_eParameterOrientation_LeftMinRightMax
AAX_eParameterOrientation_RightMinLeftMax
AAX_eParameterOrientation_RotarySingleDotMode
AAX_eParameterOrientation_RotaryBoostCutMode
AAX_eParameterOrientation_RotaryWrapMode
AAX_eParameterOrientation_RotarySpreadMode
AAX_eParameterOrientation_RotaryLeftMinRightMax
AAX_eParameterOrientation_RotaryRightMinLeftMax

15.58.4.26 enum AAX_EParameterValueInfoSelector

Query type selectors for use with [AAX_IEffectParameters::GetParameterValueInfo\(\)](#)

See also

[AAX_EEQBandTypes](#)
[AAX_EEQInCircuitPolarity](#)
[AAX_EUseAlternateControl](#)

Legacy Porting Notes converted from EControlValueInfo in the legacy SDK

Enumerator

AAX_ePageTable_EQ_Band_Type EQ filter band type. Possible response values are listed in [AAX_EEQBandTypes](#)
Legacy Porting Notes converted from eDigi_PageTable_EQ_Band_Type in the legacy SDK
AAX_ePageTable_EQ_InCircuitPolarity Description of whether a particular EQ band is active. Possible response values are listed in [AAX_EEQInCircuitPolarity](#)

Legacy Porting Notes converted from `eDigi_PageTable_EQ_InCircuitPolarity` in the legacy SDK

AAX_ePageTable_UseAlternateControl Description of whether an alternate parameter should be used for a given slot. For example, some control surfaces support Q/Slope encoders. Using an alternate control mechanism, plug-ins mapped to these devices can assign a different slope control to the alternate slot and have it coexist with a Q control for each band. This is only applicable when mapping separate parameters to the same encoder; if the Q and Slope controls are implemented as the same parameter object in the plug-in then customization is not needed.

Possible response values are listed in [AAX_EUseAlternateControl](#)

Legacy Porting Notes converted from `eDigi_PageTable_UseAlternateControl` in the legacy SDK

15.58.4.27 enum AAX_EEQBandTypes

Definitions of band types for EQ page table.

For the [AAX_ePageTable_EQ_Band_Type](#) parameter value info selector

Enumerator

AAX_eEQBandType_HighPass Freq, Slope

AAX_eEQBandType_LowShelf Freq, Gain, Slope

AAX_eEQBandType_Parametric Freq, Gain, Q

AAX_eEQBandType_HighShelf Freq, Gain, Slope

AAX_eEQBandType_LowPass Freq, Slope

AAX_eEQBandType_Notch Freq, Q

15.58.4.28 enum AAX_EEQInCircuitPolarity

Definitions for band in/out for EQ page table.

For the [AAX_ePageTable_EQ_InCircuitPolarity](#) parameter value selector

Enumerator

AAX_eEQInCircuitPolarity_Enabled EQ band is in the signal path and enabled

AAX_eEQInCircuitPolarity_Bypassed EQ band is in the signal path but bypassed/off

AAX_eEQInCircuitPolarity_Disabled EQ band is completely removed from signal path

15.58.4.29 enum AAX_EUseAlternateControl

Definitions for Use Alternate Control parameter.

For the [AAX_ePageTable_UseAlternateControl](#) parameter value info selector

Enumerator

AAX_eUseAlternateControl_No

AAX_eUseAlternateControl_Yes

15.58.4.30 enum AAX_EMIDINodeType

MIDI node types.

See also

[AAX_IComponentDescriptor::AddMIDINode\(\)](#)

Enumerator

AAX_eMIDINodeType_LocalInput Local MIDI input. Local MIDI input nodes receive MIDI by accessing [AAX_CMidiStream](#) buffers filled with MIDI messages. These buffers of MIDI data are available within the algorithm context with data corresponding to the current audio buffer being computed. The Effect can step through this buffer like a "script" to respond to MIDI events within the audio callback.

Legacy Porting Notes Corresponds to RTAS Buffered MIDI input nodes in the legacy SDK

AAX_eMIDINodeType_LocalOutput Local MIDI output. Local MIDI output nodes send MIDI by filling buffers with MIDI messages. Messages posted to MIDI output nodes will be available in the host as MIDI streams, routable to MIDI track inputs and elsewhere.

Data posted to a MIDI output buffer will be timed to correspond with the current audio buffer being processed. MIDI outputs support custom timestamping relative to the first sample of the audio buffer.

The delivery of variable length SysEx messages is also supported. There are no buffer size limitations for output of SysEx messages.

To post a MIDI output buffer, an Effect must construct a series of [AAX_CMidiPacket](#) objects and place them in the output buffer provided in the port's [AAX_CMidiStream](#)

Legacy Porting Notes Corresponds to RTAS Buffered MIDI output nodes in the legacy SDK

AAX_eMIDINodeType_Global Global MIDI node. Global MIDI nodes allow an Effect to receive streaming global MIDI data like MIDI Time Code, MIDI Beat Clock, and host-specific message formats such as the Click messages used in Pro Tools.

The specific kind of data that will be received by a Global MIDI node is specified using a mask of [AAX_EMidiGlobalNodeSelectors](#) values.

Global MIDI nodes are like local MIDI nodes, except they do not show up as assignable outputs in the host. Instead the MIDI data is automatically routed to the plug-in, without the user making any connections.

The buffer of data provided via a Global MIDI node may be shared between all currently active Effect instances, and this node may include both explicitly requested data and data not requested by the current Effect. For example, if one plug-in requests MTC and another plug-in requests Click, all plug-ins connected to this global node will get both MTC and Click messages in the shared buffer.

Legacy Porting Notes Corresponds to RTAS Shared Buffer global nodes in the legacy SDK

AAX_eMIDINodeType_Transport Transport node. Call [AAX_IMIDINode::GetTransport\(\)](#) on this node to access the [AAX_ITransport](#) interface.

15.58.4.31 enum AAX_EUpdateSource

Source for values passed into [UpdateParameterNormalizedValue\(\)](#).

Enumerator

AAX_eUpdateSource_Unspecified Parameter updates of unknown / unspecified origin, currently including all updates from control surfaces, GUI edit events, and edits originating in the plug-in outside of the context of [UpdateParameterNormalizedValue\(\)](#) or [SetChunk\(\)](#).

AAX_eUpdateSource_Parameter Parameter updates originating (via [AAX_IAutomationDelegate::PostSetValueRequest\(\)](#)) within the scope of [UpdateParameterNormalizedValue\(\)](#).

AAX_eUpdateSource_Chunk Parameter updates originating (via [AAX_IAutomationDelegate::PostSetValueRequest\(\)](#)) within the scope of [SetChunk\(\)](#).

AAX_eUpdateSource_Delay :Not Used by AAX Plug-Ins

15.58.4.32 enum AAX_EDataInPortType

Property value for whether a data in port should be buffered or not.

See also

[AAX_IComponentDescriptor::AddDataInPort\(\)](#)

Enumerator

AAX_eDataInPortType_Unbuffered Data port is unbuffered; the most recently posted packet is always delivered to the alg proc

AAX_eDataInPortType_Buffered Data port is buffered both on the host and DSP and packets are updated to the current timestamp with every alg proc call

Data delivered to alg proc always reflects the latest posted packet that has a timestamp at or before the current processing buffer

AAX_eDataInPortType_Incremental Data port is buffered both on the host and DSP and packets are updated only once per alg proc call

Since only one packet is delivered at a time, all packets will be delivered to the alg proc unless an internal buffer overflow occurs

Note

If multiple packets are posted to this port *before* the initial call to the alg proc, only the latest packet will be delivered to the first call to the alg proc. Thereafter, all packets will be delivered incrementally.

Host Compatibility Notes Supported in Pro Tools 12.5 and higher; when [AAX_eDataInPortType_Incremental](#) is not supported the port will be treated as [AAX_eDataInPort_Type_Unbuffered](#)

15.58.4.33 enum AAX_EFrameRate

FrameRate types.

See also

[AAX_ITransport::GetTimeCodeInfo\(\)](#)

Enumerator

AAX_eFrameRate_Undclared

AAX_eFrameRate_24Frame

AAX_eFrameRate_25Frame

AAX_eFrameRate_2997NonDrop

AAX_eFrameRate_2997DropFrame

AAX_eFrameRate_30NonDrop

AAX_eFrameRate_30DropFrame

AAX_eFrameRate_23976

15.58.4.34 enum AAX_EFeetFramesRate

FeetFramesRate types.

See also

[AAX_ITransport::GetFeetFramesInfo\(\)](#)

Enumerator

AAX_eFootFramesRate_23976

AAX_eFootFramesRate_24

AAX_eFootFramesRate_25

15.58.4.35 enum AAX_EMidiGlobalNodeSelectors

The Global MIDI Node Selectors.

These selectors are used in the *channelMask* argument of [AAX_IComponentDescriptor::AddMIDISelector\(\)](#) and [AAX_IEffectDescriptor::AddControlMIDISelector\(\)](#) to request one or more kinds of global data.

Enumerator

AAX_eMIDIClick Selector to request click messages. The click messages are special 2-byte messages encoded as follows:

- Accented click: Note on pitch 0 (0x90 0x00)
 - Unaccented click: Note on pitch 1 (0x90 0x01)
- Note**

No *Note Off* messages are ever sent. This isn't up-to-spec MIDI data, just a way of encoding click events.

AAX_eMIDIMtc Selector to request MIDI Time Code (MTC) data. The Standard MIDI Time Code format.

AAX_eMIDIBeatClock Selector to request MIDI Beat Clock (MBC) messages. This includes Song Position Pointer, Start/Stop/Continue, and Midi Clock (F8).

15.58.4.36 enum AAX_EPreviewState

Offline preview states for use with [AAX_eNotificationEvent_ASPreviewState](#).

Note

Do not perform any non-trivial processing within the notification handler. Instead, use the processing state notification to inform the processing that is performed in methods such as [PreRender\(\)](#).

Enumerator

AAX_ePreviewState_Stop Offline preview has ended. For [Host Processor](#) plug-ins, this notification is sent just before the final call to [PostRender\(\)](#), or after analysis is complete for plug-ins with analysis-only preview.

AAX_ePreviewState_Start Offline preview is beginning. For [Host Processor](#) plug-ins, this notification is sent before any calls to [PreAnalyze\(\)](#) or to [PreRender\(\)](#).

15.58.4.37 enum AAX_EProcessingState

Offline preview states for use with [AAX_eNotificationEvent_ASProcessingState](#).

Note

Do not perform any non-trivial processing within the notification handler. Instead, use the processing state notification to inform the processing that is performed in methods such as [PreRender\(\)](#).

Enumerator

AAX_eProcessingState_StopPass A single offline processing pass has ended. A single offline processing pass is an analysis and/or render applied to a set of channels in parallel.

For [Host Processor](#) plug-ins, this notification is sent just before the final call to [PostRender\(\)](#), or after analysis is complete for analysis-only offline plug-ins.

AAX_eProcessingState_StartPass A single offline processing pass is beginning. A single offline processing pass is an analysis and/or render applied to a set of channels in parallel.

For [Host Processor](#) plug-ins, this notification is sent before any calls to [PreAnalyze\(\)](#), [PreRender\(\)](#), or [InitOutputBounds\(\)](#) for each processing pass.

AAX_eProcessingState_EndPassGroup An offline processing pass group has completed. An offline processing pass group is a full set of analysis and/or render passes applied to the complete set of input channels.

Host Compatibility Notes AudioSuite pass group notifications are supported starting in Pro Tools 12.0

AAX_eProcessingState_BeginPassGroup An offline processing pass group is beginning. An offline processing pass group is a full set of analysis and/or render passes applied to the complete set of input channels.

Host Compatibility Notes AudioSuite pass group notifications are supported starting in Pro Tools 12.0

AAX_eProcessingState_Stop [Deprecated](#)

AAX_eProcessingState_Start [Deprecated](#)

15.58.4.38 enum AAX_ETargetPlatform

Describes what platform the component runs on.

Enumerator

kAAX_eTargetPlatform_None

kAAX_eTargetPlatform_Native

kAAX_eTargetPlatform_TI

kAAX_eTargetPlatform_External

kAAX_eTargetPlatform_Count

15.58.4.39 enum AAX_ESupportLevel

Feature support indicators

See also

[AAX_IDescriptionHost::AcquireFeatureProperties\(\)](#)

Note

: There is no value defined for unknown features. Instead, unknown features are indicated by [AcquireFeatureProperties\(\)](#) providing a null [AAI_FeatureInfo](#) in response to a request using the unknown feature UID

Enumerator

AAX_eSupportLevel_Uninitialized An uninitialized [AAX_ESupportLevel](#)

AAX_eSupportLevel_Unsupported The feature is known but explicitly not supported

AAX_eSupportLevel_Supported

AAX_eSupportLevel_Disabled

AAX_eSupportLevel_ByProperty

15.58.4.40 enum AAX_EHostLevel

Host levels.

Some AAX software hosts support different levels which are sold as separate products. For example, there may be an entry-level version of a product as well as a full version.

The level of a host may impact the user experience, workflows, or the availability of certain plug-ins. For example, some entry-level hosts are restricted to loading only specific plug-ins.

Typically an AAX plug-in should not need to query this information or change its behavior based on the level of the host.

See also

[AAXATTR_Client_Level](#)

Enumerator

AAX_eHostLevel_Unknown

AAX_eHostLevel_Standard Standard host level.

AAX_eHostLevel_Entry Entry-level host.

AAX_eHostLevel_Intermediate Intermediate-level host.

15.58.4.41 enum AAX_ETextEncoding

Describes possible string encodings.

Enumerator

AAX_eTextEncoding_Undefined

AAX_eTextEncoding_UTF8 UTF-8 string encoding.

AAX_eTextEncoding_Num

15.58.4.42 enum AAX_EAssertFlags

Flags for use with [AAI_IHostServices::HandleAssertFailure\(\)](#)

Enumerator

AAX_eAssertFlags_Default No special handler requested.

AAX_eAssertFlags_Log Logging requested.

AAX_eAssertFlags_Dialog User-visible modal alert dialog requested.

15.58.5 Function Documentation

- 15.58.5.1 `AAX_ENUM_SIZE_CHECK(AAX_EHighlightColor)`
- 15.58.5.2 `AAX_ENUM_SIZE_CHECK(AAX_ETracePriorityHost)`
- 15.58.5.3 `AAX_ENUM_SIZE_CHECK(AAX_ETracePriorityDSP)`
- 15.58.5.4 `AAX_ENUM_SIZE_CHECK(AAX_EModifiers)`
- 15.58.5.5 `AAX_ENUM_SIZE_CHECK(AAX_EAudioBufferLength)`
- 15.58.5.6 `AAX_ENUM_SIZE_CHECK(AAX_EAudioBufferLengthDSP)`
- 15.58.5.7 `AAX_ENUM_SIZE_CHECK(AAE_EAudioBufferLengthNative)`
- 15.58.5.8 `AAX_ENUM_SIZE_CHECK(AAX_EMaxAudioSuiteTracks)`
- 15.58.5.9 `AAX_ENUM_SIZE_CHECK(AAX_EStemFormat)`
- 15.58.5.10 `AAX_ENUM_SIZE_CHECK(AAX_EPlugInCategory)`
- 15.58.5.11 `AAX_ENUM_SIZE_CHECK(AAX_EPlugInStrings)`
- 15.58.5.12 `AAX_ENUM_SIZE_CHECK(AAX_EMeterOrientation)`
- 15.58.5.13 `AAX_ENUM_SIZE_CHECK(AAX_EMeterBallisticType)`
- 15.58.5.14 `AAX_ENUM_SIZE_CHECK(AAX_EMeterType)`
- 15.58.5.15 `AAX_ENUM_SIZE_CHECK(AAX_ECurveType)`
- 15.58.5.16 `AAX_ENUM_SIZE_CHECK(AAX_EResourceType)`
- 15.58.5.17 `AAX_ENUM_SIZE_CHECK(AAX_ENotificationEvent)`
- 15.58.5.18 `AAX_ENUM_SIZE_CHECK(AAX_EHostModeBits)`
- 15.58.5.19 `AAX_ENUM_SIZE_CHECK(AAX_EHostMode)`
- 15.58.5.20 `AAX_ENUM_SIZE_CHECK(AAX_EPrivateDataOptions)`
- 15.58.5.21 `AAX_ENUM_SIZE_CHECK(AAX_EConstraintLocationMask)`
- 15.58.5.22 `AAX_ENUM_SIZE_CHECK(AAX_EConstraintTopology)`
- 15.58.5.23 `AAX_ENUM_SIZE_CHECK(AAX_EComponentInstanceInitAction)`
- 15.58.5.24 `AAX_ENUM_SIZE_CHECK(AAX_ESampleRateMask)`
- 15.58.5.25 `AAX_ENUM_SIZE_CHECK(AAX_EParameterType)`
- 15.58.5.26 `AAX_ENUM_SIZE_CHECK(AAX_EParameterOrientationBits)`
- 15.58.5.27 `AAX_ENUM_SIZE_CHECK(AAX_EParameterValueInfoSelector)`

- 15.58.5.28 AAX_ENUM_SIZE_CHECK(`AAX_EEQBandTypes`)
- 15.58.5.29 AAX_ENUM_SIZE_CHECK(`AAX_EEQInCircuitPolarity`)
- 15.58.5.30 AAX_ENUM_SIZE_CHECK(`AAX_EUseAlternateControl`)
- 15.58.5.31 AAX_ENUM_SIZE_CHECK(`AAX_EMIDINodeType`)
- 15.58.5.32 AAX_ENUM_SIZE_CHECK(`AAX_EUpdateSource`)
- 15.58.5.33 AAX_ENUM_SIZE_CHECK(`AAX_EDataInPortType`)
- 15.58.5.34 AAX_ENUM_SIZE_CHECK(`AAX_EFrameRate`)
- 15.58.5.35 AAX_ENUM_SIZE_CHECK(`AAX_EFootFramesRate`)
- 15.58.5.36 AAX_ENUM_SIZE_CHECK(`AAX_EMidiGlobalNodeSelectors`)
- 15.58.5.37 AAX_ENUM_SIZE_CHECK(`AAX_EPreviewState`)
- 15.58.5.38 AAX_ENUM_SIZE_CHECK(`AAX_EProcessingState`)
- 15.58.5.39 AAX_ENUM_SIZE_CHECK(`AAX_ETargetPlatform`)
- 15.58.5.40 AAX_ENUM_SIZE_CHECK(`AAX_ESupportLevel`)
- 15.58.5.41 AAX_ENUM_SIZE_CHECK(`AAX_EHostLevel`)
- 15.58.5.42 AAX_ENUM_SIZE_CHECK(`AAX_ETextEncoding`)
- 15.58.5.43 AAX_ENUM_SIZE_CHECK(`AAX_EAssertFlags`)

15.59 AAX_Errors.h File Reference

```
#include "AAX.Enums.h"
```

15.59.1 Description

Definitions of error codes used by AAX plug-ins.

Enumerations

- enum `AAX_EError` { `AAX_SUCCESS` = 0, `AAX_ERROR_INVALID_PARAMETER_ID` = -20001, `AAX_ERROR_INVALID_STRING_CONVERSION` = -20002, `AAX_ERROR_INVALID_METER_INDEX` = -20003,

```
AAX_ERROR_NULL_OBJECT = -20004, AAX_ERROR_OLDER_VERSION = -20005, AAX_ERROR_INVALID_CHUNK_INDEX = -20006, AAX_ERROR_INVALID_CHUNK_ID = -20007, AAX_ERROR_INCORRECT_CHUNK_SIZE = -20008, AAX_ERROR_UNIMPLEMENTED = -20009, AAX_ERROR_INVALID_PARAMETER_INDEX = -20010, AAX_ERROR_NOT_INITIALIZED = -20011, AAX_ERROR_ACF_ERROR = -20012, AAX_ERROR_INVALID_METER_TYPE = -20013, AAX_ERROR_CONTEXT_ALREADY_HAS_METERS = -20014, AAX_ERROR_NULL_COMPONENT = -20015, AAX_ERROR_PORT_ID_OUT_OF_RANGE = -20016, AAX_ERROR_FIELD_TYPE_DOES_NOT_SUPPORT_DIRECT_ACCESS = -20017, AAX_ERROR_DIRECT_ACCESS_OUT_OF_BOUNDS = -20018, AAX_ERROR_FIFO_FULL = -20019, AAX_ERROR_INITIALIZING_PACKET_STREAM_THREAD = -20020, AAX_ERROR_POST_PACKET_FAILED = -20021, AAX_RESULT_PACKET_STREAM_NOT_EMPTY = -20022, AAX_RESULT_ADD_FIELD_UNSUPPORTED_FIELD_TYPE = -20023, AAX_ERROR_MIXER_THREADS_FALLING_BEHIND = -20024, AAX_ERROR_INVALID_FIELD_INDEX = -20025, AAX_ERROR_MALFORMED_CHUNK = -20026, AAX_ERROR_TOD_BEHIND = -20027, AAX_RESULT_NEW_PACKET_POSTED = -20028, AAX_ERROR_PLUGIN_NOTAUTHORIZED = -20029, AAX_ERROR_PLUGIN_NULL_PARAMETER = -20030, AAX_ERROR_NOTIFICATION FAILED = -20031, AAX_ERROR_INVALID_VIEW_SIZE = -20032, AAX_ERROR_SIGNED_INT_OVERFLOW = -20033, AAX_ERROR_NO_COMPONENTS = -20034, AAX_ERROR_DUPLICATE_EFFECT_ID = -20035, AAX_ERROR_DUPLICATE_TYPE_ID = -20036, AAX_ERROR_EMPTY_EFFECT_NAME = -20037, AAX_ERROR_UNKNOWN_PLUGIN = -20038, AAX_ERROR_PROPERTY_UNDEFINED = -20039, AAX_ERROR_INVALID_PATH = -20040, AAX_ERROR_UNKNOWN_ID = -20041, AAX_ERROR_UNKNOWN_EXCEPTION = -20042, AAX_ERROR_INVALID_ARGUMENT = -20043, AAX_ERROR_NULL_ARGUMENT = -20044, AAX_ERROR_INVALID_INTERNAL_DATA = -20045, AAX_ERROR_ARGUMENT_BUFFER_OVERFLOW = -20046, AAX_ERROR_UNSUPPORTED_ENCODING = -20047, AAX_ERROR_UNEXPECTED_EFFECT_ID = -20048, AAX_ERROR_NO_ABBREVIATED_PARAMETER_NAME = -20049, AAX_ERROR_ARGUMENT_OUT_OF_RANGE = -20050, AAX_ERROR_PLUGIN_BEGIN = -20600, AAX_ERROR_PLUGIN_END = -21000 }
```

Functions

- [AAX_ENUM_SIZE_CHECK \(AAX_EError\)](#)

15.59.2 Enumeration Type Documentation

15.59.2.1 enum AAX_EError

AAX result codes

Enumerator

```
AAX_SUCCESS  
AAX_ERROR_INVALID_PARAMETER_ID  
AAX_ERROR_INVALID_STRING_CONVERSION  
AAX_ERROR_INVALID_METER_INDEX  
AAX_ERROR_NULL_OBJECT  
AAX_ERROR_OLDER_VERSION  
AAX_ERROR_INVALID_CHUNK_INDEX  
AAX_ERROR_INVALID_CHUNK_ID  
AAX_ERROR_INCORRECT_CHUNK_SIZE  
AAX_ERROR_UNIMPLEMENTED  
AAX_ERROR_INVALID_PARAMETER_INDEX  
AAX_ERROR_NOT_INITIALIZED  
AAX_ERROR_ACF_ERROR  
AAX_ERROR_INVALID_METER_TYPE  
AAX_ERROR_CONTEXT_ALREADY_HAS_METERS  
AAX_ERROR_NULL_COMPONENT
```

AAX_ERROR_PORT_ID_OUT_OF_RANGE

AAX_ERROR_FIELD_TYPE_DOES_NOT_SUPPORT_DIRECT_ACCESS

AAX_ERROR_DIRECT_ACCESS_OUT_OF_BOUNDS

AAX_ERROR_FIFO_FULL

AAX_ERROR_INITIALIZING_PACKET_STREAM_THREAD

AAX_ERROR_POST_PACKET_FAILED

AAX_RESULT_PACKET_STREAM_NOT_EMPTY

AAX_RESULT_ADD_FIELD_UNSUPPORTED_FIELD_TYPE

AAX_ERROR_MIXER_THREAD_FALLING_BEHIND

AAX_ERROR_INVALID_FIELD_INDEX

AAX_ERROR_MALFORMED_CHUNK

AAX_ERROR_TOD_BEHIND

AAX_RESULT_NEW_PACKET_POSTED

AAX_ERROR_PLUGIN_NOT_AUTHORIZED

AAX_ERROR_PLUGIN_NULL_PARAMETER

AAX_ERROR_NOTIFICATION FAILED

AAX_ERROR_INVALID_VIEW_SIZE

AAX_ERROR_SIGNED_INT_OVERFLOW

AAX_ERROR_NO_COMPONENTS

AAX_ERROR_DUPLICATE_EFFECT_ID

AAX_ERROR_DUPLICATE_TYPE_ID

AAX_ERROR_EMPTY_EFFECT_NAME

AAX_ERROR_UNKNOWN_PLUGIN

AAX_ERROR_PROPERTY_UNDEFINED

AAX_ERROR_INVALID_PATH

AAX_ERROR_UNKNOWN_ID

AAX_ERROR_UNKNOWN_EXCEPTION An AAX plug-in should return this to the host if an unknown exception is caught. Exceptions should never be passed to the host.

AAX_ERROR_INVALID_ARGUMENT One or more input parameters are invalid; all output parameters are left unchanged.

AAX_ERROR_NULL_ARGUMENT One or more required pointer arguments are null.

AAX_ERROR_INVALID_INTERNAL_DATA Some part of the internal data required by the method is invalid.

See also

[AAX_ERROR_NOT_INITIALIZED](#)

AAX_ERROR_ARGUMENT_BUFFER_OVERFLOW A buffer argument was not large enough to hold the data which must be placed within it.

AAX_ERROR_UNSUPPORTED_ENCODING Unsupported input argument text encoding.

AAX_ERROR_UNEXPECTED_EFFECT_ID Encountered an effect ID with a different value from what was expected.

AAX_ERROR_NO_ABBREVIATED_PARAMETER_NAME No parameter name abbreviation with the requested properties has been defined.

AAX_ERROR_ARGUMENT_OUT_OF_RANGE One or more input parameters are out of the expected range, e.g. an index argument that is negative or exceeds the number of elements.

AAX_ERROR_PLUGIN_BEGIN Custom plug-in error codes may be placed in the range [[AAX_ERROR_PLUGIN_BEGIN](#), [AAX_ERROR_PLUGIN_END](#))

AAX_ERROR_PLUGIN_END Custom plug-in error codes may be placed in the range [[AAX_ERROR_PLUGIN_BEGIN](#), [AAX_ERROR_PLUGIN_END](#))

15.59.3 Function Documentation

15.59.3.1 `AAX_ENUM_SIZE_CHECK(AAX_EError)`

15.60 AAX_Exception.h File Reference

```
#include "AAX_Assert.h" #include "AAX_StringUtilities.h" #include "AAX.h" #include <exception> #include <string> #include <set>
```

15.60.1 Description

AAX SDK exception classes and utilities

Classes

- class [AAX::Exception::Any](#)
- class [AAX::Exception::ResultError](#)
- class [AAX_CheckedResult](#)
- class [AAX_AggregateResult](#)

Namespaces

- [AAX](#)
- [AAX::Exception](#)

AAX exception classes

Macros

- `#define AAX_SWALLOW(...)`
Executes X in a try/catch block that catches [AAX_CheckedResult](#) exceptions.
- `#define AAX_SWALLOW_MULT(...)`
Executes X in a try/catch block that catches [AAX_CheckedResult](#) exceptions.
- `#define AAX_CAPTURE(X, ...)`
Executes Y in a try/catch block that catches [AAX::Exception::ResultError](#) exceptions and captures the result.
- `#define AAX_CAPTURE_MULT(X, ...)`
Executes Y in a try/catch block that catches [AAX::Exception::ResultError](#) exceptions and captures the result.

Functions

- `std::string AAX::AsString (const char *inStr)`
- `const std::string & AAX::AsString (const std::string &inStr)`
- `const std::string & AAX::AsString (const Exception::Any &inStr)`

15.60.2 Macro Definition Documentation

15.60.2.1 `#define AAX_SWALLOW(...)`

Value:

```

try { if(true) { ( __VA_ARGS__ ); } } \
catch (const AAX_CheckedResult::Exception&
      AAX_PREPROCESSOR_CONCAT(ex,__LINE__)) { \
AAX_TRACE_RELEASE(kAAX_Trace_Priority_High, "%s line %d (%s)\n"
                  exception caught: %s (swallowed)", __FILE__, __LINE__, __FUNCTION__, \
                  AAX_PREPROCESSOR_CONCAT(ex,__LINE__).What().c_str()); \
} do {} while (false)

```

Executes *X* in a try/catch block that catches [AAX_CheckedResult](#) exceptions.

Catches exceptions thrown from [AAX_CheckedResult](#) only - other exceptions require an explicit catch.

```

1 AAX_CheckedResult cr;
2 cr = NecessaryFunc1();
3 AAX_SWALLOW(cr = FailableFunc());
4 cr = NecessaryFunc2();

```

15.60.2.2 #define AAX_SWALLOW_MULT(...)

Value:

```

try { if(true) { __VA_ARGS__ } } \
catch (const AAX_CheckedResult::Exception&
      AAX_PREPROCESSOR_CONCAT(ex,__LINE__)) { \
AAX_TRACE_RELEASE(kAAX_Trace_Priority_High, "%s line %d (%s)\n"
                  exception caught: %s (swallowed)", __FILE__, __LINE__, __FUNCTION__, \
                  AAX_PREPROCESSOR_CONCAT(ex,__LINE__).What().c_str()); \
} do {} while (false)

```

Executes *X* in a try/catch block that catches [AAX_CheckedResult](#) exceptions.

Version of [AAX_SWALLOW](#) for multi-line input.

Catches exceptions thrown from [AAX_CheckedResult](#) only - other exceptions require an explicit catch.

```

1 AAX_CheckedResult cr;
2 cr = NecessaryFunc();
3 AAX_SWALLOW_MULT(
4     cr = FailableFunc1();
5     cr = FailableFunc2(); // may not execute
6     cr = FailableFunc3(); // may not execute
7 );
8 cr = NecessaryFunc2();

```

15.60.2.3 #define AAX_CAPTURE(X, ...)

Value:

```

try { if(true) { ( __VA_ARGS__ ); } } \
catch (const AAX::Exception::ResultError&
      AAX_PREPROCESSOR_CONCAT(ex,__LINE__)) { \
AAX_TRACE_RELEASE(kAAX_Trace_Priority_High, "%s line %d (%s)\n"
                  exception caught: %s (captured)", __FILE__, __LINE__, __FUNCTION__, \
                  AAX_PREPROCESSOR_CONCAT(ex,__LINE__).What().c_str()); \
(X) = AAX_PREPROCESSOR_CONCAT(ex,__LINE__).Result(); \
} do {} while (false)

```

Executes *Y* in a try/catch block that catches [AAX::Exception::ResultError](#) exceptions and captures the result.

Catches exceptions thrown from [AAX_CheckedResult](#) and other [AAX::Exception::ResultError](#) exceptions.

X must be an [AAX_Result](#)

```

1 AAX_Result result = AAX_SUCCESS;
2 AAX_CAPTURE(result, ResultErrorThrowingFunc());
3 // result now holds the error code thrown by ThrowingFunc()
4
5 AAX_CheckedResult cr;
6 AAX_CAPTURE(result, cr = FailableFunc());

```

15.60.2.4 #define AAX_CAPTURE_MULT(X, ...)

Value:

```
try { if(true) { __VA_ARGS__ } } \
catch (const AAX_CheckedResult::Exception&
       AAX_PREPROCESSOR_CONCAT(ex,__LINE__)) { \
    AAX_TRACE_RELEASE(kAAX_Trace_Priority_High, "%s line %d (%s)\n"
                      "exception caught: %s (captured)", __FILE__, __LINE__, __FUNCTION__,
                      AAX_PREPROCESSOR_CONCAT(ex,__LINE__).What().c_str()); \
    (X) = AAX_PREPROCESSOR_CONCAT(ex,__LINE__).Result(); \
} do {} while (false)
```

Executes Y in a try/catch block that catches [AAX::Exception::ResultError](#) exceptions and captures the result.

Version of [AAX_CAPTURE](#) for multi-line input.

Catches exceptions thrown from [AAX_CheckedResult](#) and other [AAX::Exception::ResultError](#) exceptions.

X must be an [AAX_Result](#) or an implicitly convertible type

```
1 AAX_Result result = AAX_SUCCESS;
2 AAX_CAPTURE_MULT(result,
3     MaybeThrowingFunc1();
4     MaybeThrowingFunc2();
5
6     // can use AAX_CheckedResult within AAX_CAPTURE_MULT
7     AAX_CheckedResult cr;
8     cr = FailableFunc1();
9     cr = FailableFunc2();
10    cr = FailableFunc3();
11 );
12
13 // result now holds the value of the last thrown error
14 return result;
```

15.61 AAX_Exports.cpp File Reference

```
#include "AAX_Init.h" #include "AAX.h" #include "acfunknown.h" #include "acfresult.h"
```

Macros

- #define [AAX_EXPORT](#) extern "C" __declspec(dllexport) ACFRESULT __stdcall

Functions

- [AAX_EXPORT ACFRegisterPlugin](#) (IACFUnknown *pUnkHostVoid, IACFPluginDefinition **ppPluginDefinitionVoid)

The main plug-in registration method.
- [AAX_EXPORT ACFRegisterComponent](#) (IACFUnknown *pUnkHost, acfUInt32 index, IACFComponentDefinition **ppComponentDefinition)

Registers a specific component in the DLL.
- [AAX_EXPORT ACGetClassFactory](#) (IACFUnknown *pUnkHost, const acfCLSID &clsid, const acfIID &iid, void **ppOut)

Gets the factory for a given class ID.
- [AAX_EXPORT ACFCanUnloadNow](#) (IACFUnknown *pUnkHost)

Determines whether or not the host may unload the DLL.
- [AAX_EXPORT ACFStartup](#) (IACFUnknown *pUnkHost)

DLL initialization routine.
- [AAX_EXPORT ACFShutdown](#) (IACFUnknown *pUnkHost)

DLL shutdown routine.

- [AAX_EXPORT ACFGetSDKVersion \(acfUInt64 *oSDKVersion\)](#)

Returns the DLL's SDK version.

15.61.1 Macro Definition Documentation

15.61.1.1 `#define AAX_EXPORT extern "C" __declspec(dllexport) ACFRESULT __stdcall`

15.61.2 Function Documentation

15.61.2.1 `ACFAPI ACFRegisterPlugin (IACFUnknown * pUnkHost, IACFPluginDefinition ** ppPluginDefinition)`

The main plug-in registration method.

References [AAXRegisterPlugin\(\)](#).

Here is the call graph for this function:



15.61.2.2 `ACFAPI ACFRegisterComponent (IACFUnknown * pUnkHost, acfUInt32 index, IACFComponentDefinition ** ppComponentDefinition)`

Registers a specific component in the DLL.

References [AAXRegisterComponent\(\)](#).

Here is the call graph for this function:



15.61.2.3 `ACFAPI ACFGetClassFactory (IACFUnknown * pUnkHost, const acfCLSID & clsid, const acfIID & iid, void ** ppOut)`

Gets the factory for a given class ID.

References [AAXGetClassFactory\(\)](#).

Here is the call graph for this function:



15.61.2.4 ACFAPI ACFCANUnloadNow (IACFUnknown * pUnkHost)

Determines whether or not the host may unload the DLL.

References AAXCanUnloadNow().

Here is the call graph for this function:



15.61.2.5 ACFAPI ACFStartup (IACFUnknown * pUnkHost)

DLL initialization routine.

References AAXStartup().

Here is the call graph for this function:



15.61.2.6 ACFAPI ACFShutdown (IACFUnknown * pUnkHost)

DLL shutdown routine.

References AAXShutdown().

Here is the call graph for this function:



15.61.2.7 ACFAPI ACFGetSDKVersion (*acfUInt64 * oSDKVersion*)

Returns the DLL's SDK version.

References AAXGetSDKVersion().

Here is the call graph for this function:



15.62 AAX_FastInterpolatedTableLookup.h File Reference

```
#include "AAX_Quantize.h"
#include <AAX_ALIGN_FILE_ALG>
#include <AAX_ALIGNFILE_RESET>
#include "AAX_FastInterpolatedTableLookup.hpp"
```

15.62.1 Description

A set of functions that provide lookup table functionality. Not necessarily optimized for TI, but used internally.

Classes

- class [AAX_FastInterpolatedTableLookup< TFLOAT, DFLOAT >](#)

Macros

- [#define AAX_FASTINTERPOLATEDTABLELOOKUP_H](#)

15.62.2 Macro Definition Documentation

15.62.2.1 [#define AAX_FASTINTERPOLATEDTABLELOOKUP_H](#)

15.63 AAX_FastInterpolatedTableLookup.hpp File Reference

```
#include "AAX_Quantize.h"
```

15.64 AAX_FastPow.h File Reference

```
#include <cmath>#include "AAX.h"
```

15.64.1 Description

Set of functions to optimize pow.

To use:

```
const int kPowTableExtent = 9;           // Lower values are less precise. 9 is the maximum
float powTableH[kPowTableSize];
float powTableL[kPowTableSize];
int radix = 2;                         // This should be whatever radix you want. Ie: radix ^ exp
PowFastSetTable( powTableH, kPowExtent, false ); // Set the high table
PowFastSetTable( powTableL, kPowExtent*2, kPowExtent, true );// Set the low table
..
result = powFastLookup(exp, log(2) / log(radix), powTableH, powTableL);
```

Namespaces

- [AAX](#)

Macros

- [#define _AAX_FASTPOW_H_](#)

Variables

- [const unsigned int AAX::kPowExtent = 9](#)
- [const unsigned int AAX::kPowTableSize = 1 << kPowExtent](#)

15.64.2 Macro Definition Documentation

[15.64.2.1 #define _AAX_FASTPOW_H_](#)

15.65 AAX_Getting_Start_Guide.dox File Reference

15.66 AAX_GUITypes.h File Reference

```
#include "AAX.h"#include "AAX_Push2ByteStructAlignment.h"#include "AAX_PopStructAlignment.h"
```

15.66.1 Description

Constants and other definitions used by AAX plug-in GUIs.

Classes

- struct [AAX_Point](#)
Data structure representing a two-dimensional coordinate point.
- struct [AAX_Rect](#)
Data structure representing a rectangle in a two-dimensional coordinate plane.

Typedefs

- typedef struct [AAX_Point](#) [AAX_Point](#)
Data structure representing a two-dimensional coordinate point.
- typedef struct [AAX_Rect](#) [AAX_Rect](#)
Data structure representing a rectangle in a two-dimensional coordinate plane.
- typedef enum [AAX_EViewContainer_Type](#) [AAX_EViewContainer_Type](#)
Type of view container.

Enumerations

- enum [AAX_EViewContainer_Type](#) { [AAX_eViewContainer_Type_NULL](#) = 0, [AAX_eViewContainer_Type_NSView](#) = 1, [AAX_eViewContainer_Type_UIView](#) = 2, [AAX_eViewContainer_Type_HWND](#) = 3 }
- Type of view container.*

Functions

- bool [operator==](#) (const [AAX_Point](#) &p1, const [AAX_Point](#) &p2)
- bool [operator!=](#) (const [AAX_Point](#) &p1, const [AAX_Point](#) &p2)
- bool [operator<](#) (const [AAX_Point](#) &p1, const [AAX_Point](#) &p2)
- bool [operator<=](#) (const [AAX_Point](#) &p1, const [AAX_Point](#) &p2)
- bool [operator>](#) (const [AAX_Point](#) &p1, const [AAX_Point](#) &p2)
- bool [operator>=](#) (const [AAX_Point](#) &p1, const [AAX_Point](#) &p2)
- bool [operator==](#) (const [AAX_Rect](#) &r1, const [AAX_Rect](#) &r2)
- bool [operator!=](#) (const [AAX_Rect](#) &r1, const [AAX_Rect](#) &r2)
- [AAX_ENUM_SIZE_CHECK](#) ([AAX_EViewContainer_Type](#))

15.66.2 Typedef Documentation

15.66.2.1 [typedef struct AAX_Point](#) [AAX_Point](#)

Data structure representing a two-dimensional coordinate point.

Comparison operators give preference to `vert`

15.66.2.2 [typedef struct AAX_Rect](#) [AAX_Rect](#)

Data structure representing a rectangle in a two-dimensional coordinate plane.

15.66.2.3 [typedef enum AAX_EViewContainer_Type](#) [AAX_EViewContainer_Type](#)

Type of `view container`.

See also

[AAX_IViewContainer::GetType\(\)](#)

15.66.3 Enumeration Type Documentation

15.66.3.1 enum AAX_EViewContainer_Type

Type of [view container](#).

See also

[AAX_IViewContainer::GetType\(\)](#)

Enumerator

AAX_eViewContainer_Type_NULL

AAX_eViewContainer_Type_NSView

AAX_eViewContainer_Type_UIView

AAX_eViewContainer_Type_HWND

15.66.4 Function Documentation

15.66.4.1 bool operator== (const AAX_Point & p1, const AAX_Point & p2) [inline]

References AAX_Point::horz, and AAX_Point::vert.

15.66.4.2 bool operator!= (const AAX_Point & p1, const AAX_Point & p2) [inline]

15.66.4.3 bool operator< (const AAX_Point & p1, const AAX_Point & p2) [inline]

References AAX_Point::horz, and AAX_Point::vert.

15.66.4.4 bool operator<= (const AAX_Point & p1, const AAX_Point & p2) [inline]

References AAX_Point::horz, and AAX_Point::vert.

15.66.4.5 bool operator> (const AAX_Point & p1, const AAX_Point & p2) [inline]

15.66.4.6 bool operator>= (const AAX_Point & p1, const AAX_Point & p2) [inline]

15.66.4.7 bool operator== (const AAX_Rect & r1, const AAX_Rect & r2) [inline]

References AAX_Rect::height, AAX_Rect::left, AAX_Rect::top, and AAX_Rect::width.

15.66.4.8 bool operator!= (const AAX_Rect & r1, const AAX_Rect & r2) [inline]

15.66.4.9 AAX_ENUM_SIZE_CHECK (AAX_EViewContainer_Type)

15.67 AAX_HostSupport.dox File Reference

15.68 AAX_IACFAutomationDelegate.h File Reference

```
#include "AAX.h" #include "acfunknow.h"
```

15.68.1 Description

Versioned interface allowing an AAX plug-in to interact with the host's automation system.

Classes

- class [AAX_IACFAutomationDelegate](#)

Versioned interface allowing an AAX plug-in to interact with the host's automation system.

15.69 AAX_IACFCollection.h File Reference

```
#include "acfbaseapi.h"
```

15.69.1 Description

Versioned interface to represent a plug-in binary's static description.

Classes

- class [AAX_IACFCollection](#)

Versioned interface to represent a plug-in binary's static description.

15.70 AAX_IACFComponentDescriptor.h File Reference

```
#include "AAX.h"#include "AAX_Callbacks.h"#include "AAX_IDma.h"#include  
"acfunknown.h"
```

15.70.1 Description

Versioned description interface for an AAX plug-in algorithm callback.

Classes

- class [AAX_IACFComponentDescriptor](#)

Versioned description interface for an AAX plug-in algorithm callback.

- class [AAX_IACFComponentDescriptor_V2](#)

Versioned description interface for an AAX plug-in algorithm callback.

- class [AAX_IACFComponentDescriptor_V3](#)

Versioned description interface for an AAX plug-in algorithm callback.

15.71 AAX_IACFController.h File Reference

```
#include "AAX.h"#include "acfunknown.h"
```

15.71.1 Description

Interface for the AAX host's view of a single instance of an effect. Used by both clients of the AAXHost and by effect components.

Classes

- class [AAX_IACFController](#)
Interface for the AAX host's view of a single instance of an effect. Used by both clients of the AAXHost and by effect components.
- class [AAX_IACFController_V2](#)
Interface for the AAX host's view of a single instance of an effect. Used by both clients of the AAXHost and by effect components.
- class [AAX_IACFController_V3](#)
Interface for the AAX host's view of a single instance of an effect. Used by both clients of the AAXHost and by effect components.

15.72 AAX_IACFDescriptionHost.h File Reference

```
#include "AAX.h" #include "acfbaseapi.h" #include "acfunknown.h"
```

Classes

- class [AAX_IACFDescriptionHost](#)

15.73 AAX_IACFEffectorDescriptor.h File Reference

```
#include "AAX.h" #include "AAX_Callbacks.h" #include "acfunknown.h"
```

15.73.1 Description

Versioned interface for an [AAX_IEffectDescriptor](#).

Classes

- class [AAX_IACFEffectorDescriptor](#)
Versioned interface for an [AAX_IEffectDescriptor](#).
- class [AAX_IACFEffectorDescriptor_V2](#)
Versioned interface for an [AAX_IEffectDescriptor](#).

15.74 AAX_IACFEffectorDirectData.h File Reference

```
#include "AAX.h"
```

```
#include "acfunknow.h"
```

15.74.1 Description

The direct data access interface that gets exposed to the host application.

Classes

- class [AAX_IACFEffecDirectData](#)
Optional interface for direct access to a plug-in's alg memory.

15.75 AAX_IACFEffecGUI.h File Reference

```
#include "AAX.h"#include "AAX_GUITypes.h"#include "AAX_IString.h"#include
"acfunknow.h"
```

15.75.1 Description

The GUI interface that gets exposed to the host application.

Classes

- class [AAX_IACFEffecGUI](#)
The interface for a AAX Plug-in's GUI (graphical user interface).

15.76 AAX_IACFEffecParameters.h File Reference

```
#include "AAX.h"#include "acfunknow.h"
```

15.76.1 Description

The data model interface that is exposed to the host application.

Classes

- class [AAX_IACFEffecParameters](#)
The interface for an AAX Plug-in's data model.
- struct [AAX_SHybridRenderInfo](#)
Hybrid render processing context.
- class [AAX_IACFEffecParameters_V2](#)
Supplemental interface for an AAX Plug-in's data model.
- class [AAX_IACFEffecParameters_V3](#)
Supplemental interface for an AAX Plug-in's data model.
- class [AAX_IACFEffecParameters_V4](#)
Supplemental interface for an AAX Plug-in's data model.

15.77 AAX_IACFFeatureInfo.h File Reference

```
#include "AAX.h" #include "acfunknown.h"
```

Classes

- class [AAX_IACFFeatureInfo](#)

15.78 AAX_IACFHostProcessor.h File Reference

```
#include "AAX.h" #include "acfunknown.h"
```

15.78.1 Description

The host processor interface that is exposed to the host application.

Classes

- class [AAX_IACFHostProcessor](#)
Versioned interface for an AAX host processing component.
- class [AAX_IACFHostProcessor_V2](#)
Supplemental interface for an AAX host processing component.

15.79 AAX_IACFHostProcessorDelegate.h File Reference

```
#include "AAX.h" #include "acfunknown.h"
```

Classes

- class [AAX_IACFHostProcessorDelegate](#)
Versioned interface for host methods specific to offline processing.
- class [AAX_IACFHostProcessorDelegate_V2](#)
Versioned interface for host methods specific to offline processing.
- class [AAX_IACFHostProcessorDelegate_V3](#)
Versioned interface for host methods specific to offline processing.

15.80 AAX_IACFHostServices.h File Reference

```
#include "AAX.h" #include "acfunknown.h"
```

Classes

- class [AAX_IACFHostServices](#)
Versioned interface to diagnostic and debugging services provided by the AAX host.
- class [AAX_IACFHostServices_V2](#)
V2 of versioned interface to diagnostic and debugging services provided by the AAX host.
- class [AAX_IACFHostServices_V3](#)
V3 of versioned interface to diagnostic and debugging services provided by the AAX host.

15.81 AAX_IACFPageTable.h File Reference

```
#include "AAX.h" #include "acfunknown.h"
```

Classes

- class [AAX_IACFPageTable](#)
Versioned interface to the host's representation of a plug-in instance's page table.
- class [AAX_IACFPageTable_V2](#)
Versioned interface to the host's representation of a plug-in instance's page table.

15.82 AAX_IACFPageTableController.h File Reference

```
#include "AAX.h" #include "acfunknown.h"
```

Classes

- class [AAX_IACFPageTableController](#)
Interface for host operations related to the page tables for this plug-in.
- class [AAX_IACFPageTableController_V2](#)
Interface for host operations related to the page tables for this plug-in.

15.83 AAX_IACFPrivateDataAccess.h File Reference

```
#include "AAX.h" #include "acfunknown.h"
```

15.83.1 Description

Interface for the AAX host's data access functionality.

Classes

- class [AAX_IACFPrivateDataAccess](#)
Interface for the AAX host's data access functionality.

15.84 AAX_IACFPropertyMap.h File Reference

```
#include "AAX.h" #include "acfunknown.h"
```

15.84.1 Description

Versioned interface for an [AAX_IPropertyMap](#).

Classes

- class [AAX_IACFPropertyMap](#)
Versioned interface for an [AAX_IPropertyMap](#).
- class [AAX_IACFPropertyMap_V2](#)
Versioned interface for an [AAX_IPropertyMap](#).
- class [AAX_IACFPropertyMap_V3](#)
Versioned interface for an [AAX_IPropertyMap](#).

15.85 AAX_IACFTransport.h File Reference

```
#include "AAX.h" #include "acfunknown.h"
```

15.85.1 Description

Interface for the AAX Transport data access functionality.

Classes

- class [AAX_IACFTransport](#)
Versioned interface to information about the host's transport state.
- class [AAX_IACFTransport_V2](#)
Versioned interface to information about the host's transport state.

15.86 AAX_IACFViewContainer.h File Reference

```
#include "AAX_GUITypes.h" #include "AAX.h" #include "acfunknown.h"
```

15.86.1 Description

Interface for the AAX host's view of a single instance of an effect. Used by both clients of the AAXHost and by effect components.

Classes

- class [AAX_IACFViewContainer](#)
Interface for the AAX host's view of a single instance of an effect. Used by both clients of the host app and by effect components.
- class [AAX_IACFViewContainer_V2](#)
Supplemental interface for the AAX host's view of a single instance of an effect. Used by both clients of the host app and by effect components.

15.87 AAX_IAutomationDelegate.h File Reference

```
#include "AAX.h"
```

15.87.1 Description

Interface allowing an AAX plug-in to interact with the host's automation system.

Classes

- class [AAX_IAutomationDelegate](#)

Interface allowing an AAX plug-in to interact with the host's event system.

15.88 AAX_ICollection.h File Reference

```
#include "AAX.h"
```

15.88.1 Description

Interface to represent a plug-in binary's static description.

Classes

- class [AAX_ICollection](#)

Interface to represent a plug-in binary's static description.

15.89 AAX_IComponentDescriptor.h File Reference

```
#include "AAX.h" #include "AAX_IDma.h" #include "AAX_Callbacks.h"
```

15.89.1 Description

Description interface for an AAX plug-in algorithm.

Classes

- class [AAX_IComponentDescriptor](#)

Description interface for an AAX plug-in component.

15.90 AAX.IContainer.h File Reference

15.90.1 Description

Abstract container interface.

Classes

- class [AAX.IContainer](#)

15.91 AAX_IController.h File Reference

```
#include "AAX_Properties.h" #include "AAX_IString.h" #include "AAX.h"
```

15.91.1 Description

Interface for the AAX host's view of a single instance of an effect.

Classes

- class [AAX_IController](#)

Interface for the AAX host's view of a single instance of an effect. Used by both clients of the AAX host and by effect components.

15.92 AAX_IDescriptionHost.h File Reference

```
#include "AAX.h"
```

Classes

- class [AAX_IDescriptionHost](#)

15.93 AAX_IDisplayDelegate.h File Reference

```
#include "AAX.h"
```

15.93.1 Description

Defines the display behavior for a parameter.

Classes

- class [AAX_IDisplayDelegateBase](#)
Defines the display behavior for a parameter.
- class [AAX_IDisplayDelegate< T >](#)
Classes for parameter value string conversion.

15.94 AAX_IDisplayDelegateDecorator.h File Reference

```
#include "AAX_IDisplayDelegate.h"
```

15.94.1 Description

The base class for all concrete display delegate decorators.

Classes

- class [AAX_IDisplayDelegateDecorator< T >](#)

The base class for all concrete display delegate decorators.

15.95 AAX_IDma.h File Reference

```
#include "AAX.h"
```

15.95.1 Description

Cross-platform interface for access to the host's direct memory access (DMA) facilities.

Classes

- class [AAX_IDma](#)

Cross-platform interface for access to the host's direct memory access (DMA) facilities.

Macros

- #define [AAX_IDMA_H](#)
- #define [AAX_DMA_API](#)

15.95.2 Macro Definition Documentation

15.95.2.1 #define [AAX_IDMA_H](#)

15.95.2.2 #define [AAX_DMA_API](#)

15.96 AAX_IEffectDescriptor.h File Reference

```
#include "AAX.h" #include "AAX_Callbacks.h"
```

15.96.1 Description

Description interface for an effect's (plug-in type's) components.

Classes

- class [AAX_IEffectDescriptor](#)

Description interface for an effect's (plug-in type's) components.

15.97 AAX_IEffectDirectData.h File Reference

```
#include "AAX_IACFEffectDirectData.h" #include "AAX.h" #include "CACFUnknown.h"
```

15.97.1 Description

Optional interface for direct access to alg memory.

Classes

- class [AAX_IEffectDirectData](#)
The interface for a AAX Plug-in's direct data interface.

15.98 AAX_IEffectGUI.h File Reference

```
#include "AACFUnknown.h"
```

15.98.1 Description

The interface for a AAX Plug-in's user interface.

Classes

- class [AAX_IEffectGUI](#)

The interface for a AAX Plug-in's user interface.

15.99 AAX_IEffectParameters.h File Reference

```
#include "AAX_IACFEffectParameters.h"#include "AAX.h"#include "CACFUnknown.h"
```

15.99.1 Description

The interface for an AAX Plug-in's data model.

Classes

- class [AAX_IEffectParameters](#)

The interface for an AAX Plug-in's data model.

15.100 AAX_IFeatureInfo.h File Reference

```
#include "AAX.h"
```

Classes

- class [AAX_IFeatureInfo](#)

15.101 AAX_IHostProcessor.h File Reference

```
#include "AAX_IACFHostProcessor.h"#include "AAX.h"#include "CACFUnknown.h"
```

15.101.1 Description

Base class for the host processor interface which is extended by plugin code.

Classes

- class [AAX_IHostProcessor](#)

Base class for the host processor interface.

15.102 AAX_IHostProcessorDelegate.h File Reference

```
#include "AAX.h"
```

15.102.1 Description

Interface allowing plug-in's HostProcessor to interact with the host's side.

Classes

- class [AAX_IHostProcessorDelegate](#)

Versioned interface for host methods specific to offline processing.

15.103 AAX_IHostServices.h File Reference

```
#include "AAX.h"
```

15.103.1 Description

Various host services.

Classes

- class [AAX_IHostServices](#)

Interface to diagnostic and debugging services provided by the AAX host.

15.104 AAX_IMIDINode.h File Reference

```
#include "AAX.h" #include "AAX_ITransport.h"
```

15.104.1 Description

Declaration of the base MIDI Node interface.

Author

by Andriy Goshko

Classes

- class [AAX_IMIDINode](#)

Interface for accessing information in a MIDI node.

15.105 AAX_Init.h File Reference

```
#include "AAX.h" #include "acfbasetypes.h"
```

15.105.1 Description

AAX library implementations of required plug-in initialization, registration, and tear-down methods.

Functions

- [AAX_Result AAXRegisterPlugin \(IACFUnknown *pUnkHost, IACFPluginDefinition **ppPluginDefinition\)](#)
The main plug-in registration method.
- [AAX_Result AAXRegisterComponent \(IACFUnknown *pUnkHost, acfUInt32 index, IACFComponentDefinition **ppComponentDefinition\)](#)
Registers a specific component in the DLL.
- [AAX_Result AAXGetClassFactory \(IACFUnknown *pUnkHost, const acfCLSID &clsid, const acfIID &iid, void **ppOut\)](#)
Gets the factory for a given class ID.
- [AAX_Result AAXCanUnloadNow \(IACFUnknown *pUnkHost\)](#)
Determines whether or not the host may unload the DLL.
- [AAX_Result AAXStartup \(IACFUnknown *pUnkHost\)](#)
DLL initialization routine.
- [AAX_Result AAXShutdown \(IACFUnknown *pUnkHost\)](#)
DLL shutdown routine.
- [AAX_Result AAXGetSDKVersion \(acfUInt64 *oSDKVersion\)](#)
Returns the DLL's SDK version.

15.105.2 Function Documentation

15.105.2.1 AAX_Result AAXRegisterComponent (IACFUnknown * pUnkHost, acfUInt32 index, IACFComponentDefinition ** ppComponentDefinition)

Registers a specific component in the DLL.

The implementation of this method in the AAX library simply sets *ppComponentDefinition to NULL and returns [AAX_SUCCESS](#).

Wrapped by [ACFRegisterComponent\(\)](#)

Referenced by [ACFRegisterComponent\(\)](#).

Here is the caller graph for this function:



15.105.2.2 AAX_Result AAXGetClassFactory (IACFUnknown * pUnkHost, const acfCLSID & clsid, const acfIID & iid, void ** ppOut)

Gets the factory for a given class ID.

This method is required by ACF but is not supported by AAX. Therefore the implementation of this method in the AAX library simply sets *ppOut to NULL and returns [AAX_ERROR_UNIMPLEMENTED](#).

Wrapped by [ACFGetClassFactory\(\)](#)

Referenced by ACFGetClassFactory().

Here is the caller graph for this function:



15.105.2.3 AAX_Result AAXCanUnloadNow (IACFUnknown * pUnkHost)

Determines whether or not the host may unload the DLL.

The implementation of this method in the AAX library returns the result of GetActiveObjectCount () as an [AAX_Result](#), with zero active objects interpreted as [AAX_SUCCESS](#) (see CACFUnknown.h)

Wrapped by [ACFCanUnloadNow\(\)](#)

Referenced by ACFCanUnloadNow().

Here is the caller graph for this function:



15.105.2.4 AAX_Result AAXStartup (IACFUnknown * pUnkHost)

DLL initialization routine.

Called once at init time. The implementation of this method in the AAX library uses pUnkHost as an [IACFComponentFactory](#) to initialize global services (see acfbaseapi.h)

Wrapped by [ACFStartup\(\)](#)

Referenced by ACFStartup().

Here is the caller graph for this function:



15.105.2.5 `AAX_Result AAXShutdown(IACFUnknown * pUnkHost)`

DLL shutdown routine.

Called once before unloading the DLL. The implementation of this method in the AAX library tears down any globally initialized state and releases any globally retained resources.

Wrapped by [ACFShutdown\(\)](#)

Referenced by `ACFShutdown()`.

Here is the caller graph for this function:



15.105.2.6 `AAX_Result AAXGetSDKVersion(acfUInt64 * oSDKVersion)`

Returns the DLL's SDK version.

The implementation of this method in the AAX library provides a 64-bit value in which the upper 32 bits represent the SDK version and the lower 32 bits represent the revision number of the SDK. See [AAX_Version.h](#)

Wrapped by [ACFGetSDKVersion\(\)](#)

Referenced by `ACFGetSDKVersion()`.

Here is the caller graph for this function:



15.106 AAX_InstrumentParameters.dox File Reference

15.107 AAX_InterfaceList.dox File Reference

15.108 AAX_IPageTable.h File Reference

```
#include "AAX.h" #include "AAX_IString.h"
```

Classes

- class [AAX_IPageTable](#)
Interface to the host's representation of a plug-in instance's page table.

15.109 AAX_IParameter.h File Reference

```
#include "AAX.h"
```

15.109.1 Description

The base interface for all normalizable plug-in parameters.

Classes

- class [AAX_IParameterValue](#)
An abstract interface representing a parameter value of arbitrary type.
- class [AAX_IParameter](#)
The base interface for all normalizable plug-in parameters.

15.110 AAX_IPointerQueue.h File Reference

```
#include "AAX.IContainer.h"
```

15.110.1 Description

Abstract interface for a basic FIFO queue of pointers-to-objects.

Classes

- class [AAX_IPointerQueue< T >](#)

15.111 AAX_IPrivateDataAccess.h File Reference

```
#include "AAX.h"
```

15.111.1 Description

Interface to data access provided by host to plug-in.

Classes

- class [AAX_IPrivateDataAccess](#)

Interface to data access provided by host to plug-in.

15.112 AAX_IPropertyMap.h File Reference

```
#include "AAX_Properties.h" #include "AAX.h"
```

15.112.1 Description

Generic plug-in description property map.

Classes

- class [AAX_IPropertyMap](#)

Generic plug-in description property map.

15.113 AAX_IString.h File Reference

```
#include "AAX.h"
```

15.113.1 Description

An AAX string interface.

Classes

- class [AAX_IString](#)

A simple string container that can be passed across a binary boundary. This class, for simplicity, is not versioned and thus can never change.

15.114 AAX_ITaperDelegate.h File Reference

15.114.1 Description

Defines the taper conversion behavior for a parameter.

Classes

- class [AAX_ITaperDelegateBase](#)

Defines the taper conversion behavior for a parameter.

- class [AAX_ITaperDelegate< T >](#)

Classes for conversion to and from normalized parameter values.

15.115 AAX_ITransport.h File Reference

```
#include "AAX.h"
```

15.115.1 Description

The interface for query ProTools transport information.

Note

To use this interface plug-in must describe AAX_eProperty_UsesMIDI property

Classes

- class [AAX_ITransport](#)

Interface to information about the host's transport state.

15.116 AAX_IViewContainer.h File Reference

```
#include "AAX_GUITypes.h" #include "AAX.h"
```

15.116.1 Description

Interface for the AAX host's view of a single instance of an effect.

Classes

- class [AAX_IViewContainer](#)

Interface for the AAX host's view of a single instance of an effect. Used both by clients of the AAX host and by effect components.

15.117 AAX_LinkedParameters.dox File Reference

15.118 AAX_Map.h File Reference

```
#include "AAX.h" #include <AAX_ALIGN_FILE_ALG> #include <AAX_ALIGN_FILE_RES←  
ET>
```

15.118.1 Description

Author

Mykola Kryvonos

Classes

- class [AAX_Map](#)

Macros

- `#define AAX_MAP_H`

15.118.2 Macro Definition Documentation

15.118.2.1 `#define AAX_MAP_H`

15.119 AAX_Media_Composer_Guide.dox File Reference

15.120 AAX_MIDILogging.cpp File Reference

```
#include "AAX_MIDILogging.h" #include "AAX_CString.h" #include "AAX_Assert.h" #include <map> #include <vector> #include <algorithm>
```

Classes

- struct `SAutoArray< T >`
- class `AAX_IMIDIMessageInfoDelegate`

15.121 AAX_MIDILogging.h File Reference

```
#include "AAX.h"
```

15.121.1 Description

Utilities for logging MIDI data.

Namespaces

- `AAX`

Functions

MIDI logging utilities

- void `AAX::AsStringMIDIStream_Debug` (const `AAX_CMidiStream` &inStream, char *outBuffer, int32_t bufferSize)

15.122 AAX_MIDIUtilities.h File Reference

```
#include "AAX.h"
```

15.122.1 Description

Utilities for managing MIDI data.

Namespaces

- [AAX](#)

Enumerations

- enum [AAX::EStatusNibble](#) { [AAX::eStatusNibble_NoteOff](#) = 0x80, [AAX::eStatusNibble_NoteOn](#) = 0x90, [AAX::eStatusNibble_KeyPressure](#) = 0xA0, [AAX::eStatusNibble_ControlChange](#) = 0xB0, [AAX::eStatusNibble_ChannelMode](#) = 0xB0, [AAX::eStatusNibble_ProgramChange](#) = 0xC0, [AAX::eStatusNibble_ChannelPressure](#) = 0xD0, [AAX::eStatusNibble_PitchBend](#) = 0xE0, [AAX::eStatusNibble_SystemCommon](#) = 0xF0, [AAX::eStatusNibble_SystemRealTime](#) = 0xF0 }

Values for the status nibble in a MIDI packet.

- enum [AAX::EStatusByte](#) { [AAX::eStatusByte_SysExBegin](#) = 0xF0, [AAX::eStatusByte_MTCQuarterFrame](#) = 0xF1, [AAX::eStatusByte_SongPosition](#) = 0xF2, [AAX::eStatusByte_SongSelect](#) = 0xF3, [AAX::eStatusByte_TuneRequest](#) = 0xF6, [AAX::eStatusByte_SysExEnd](#) = 0xF7, [AAX::eStatusByte_TimingClock](#) = 0xF8, [AAX::eStatusByte_Start](#) = 0xFA, [AAX::eStatusByte_Continue](#) = 0xFB, [AAX::eStatusByte_Stop](#) = 0xFC, [AAX::eStatusByte_ActiveSensing](#) = 0xFE, [AAX::eStatusByte_Reset](#) = 0xFF }

Values for the status byte in a MIDI packet.

- enum [AAX::EChannelModeData](#) { [AAX::eChannelModeData_AllSoundOff](#) = 120, [AAX::eChannelModeData_ResetControllers](#) = 121, [AAX::eChannelModeData_LocalControl](#) = 122, [AAX::eChannelModeData_AllNotesOff](#) = 123, [AAX::eChannelModeData_OmniOff](#) = 124, [AAX::eChannelModeData_OmniOn](#) = 125, [AAX::eChannelModeData_PolyOff](#) = 126, [AAX::eChannelModeData_PolyOn](#) = 127 }

Values for the first data byte in a Channel Mode Message MIDI packet.

- enum [AAX::ESpecialData](#) { [AAX::eSpecialData_AccentedClick](#) = 0x00, [AAX::eSpecialData_UnaccentedClick](#) = 0x01 }

Special message data for the first data byte in a message.

Functions

- bool [AAX::IsNoteOn](#) (const [AAX_CMidiPacket](#) *inPacket)
Returns true if inPacket is a Note On message.
- bool [AAX::IsNoteOff](#) (const [AAX_CMidiPacket](#) *inPacket)
Returns true if inPacket is a Note Off message, or a Note On message with velocity zero.
- bool [AAX::IsAllNotesOff](#) (const [AAX_CMidiPacket](#) *inPacket)
Returns true if inPacket is an All Sound Off or All Notes Off message.
- bool [AAX::IsAccentedClick](#) (const [AAX_CMidiPacket](#) *inPacket)
Returns true if inPacket is a special Pro Tools accented click message.
- bool [AAX::IsUnaccentedClick](#) (const [AAX_CMidiPacket](#) *inPacket)
Returns true if inPacket is a special Pro Tools unaccented click message.
- bool [AAX::IsClick](#) (const [AAX_CMidiPacket](#) *inPacket)
Returns true if inPacket is a special Pro Tools click message.

15.123 AAX_MiscUtils.h File Reference

```
#include "AAX_PlatformOptimizationConstants.h"
#include "AAX_Constants.h"
#include "AAX_Denormal.h"
```

15.123.1 Description

Miscellaneous signal processing utilities.

Namespaces

- [AAX](#)

Macros

- `#define AAX_MISCUTILS_H`
- `#define AAX_ALIGNMENT_HINT(a, b)`

Currently only functional on TI, these word alignments will provide better performance on TI.

- `#define AAX_WORD_ALIGNED_HINT(a)`
- `#define AAX_DWORD_ALIGNED_HINT(a)`
- `#define AAX_LO(x) x`

These macros are used on TI to convert 2 single words accesses to one double word access to provide additional optimization.

- `#define AAX_HI(x) *((const_cast<float*>(&x))+1)`
- `#define AAX_INT_LO(x) x`
- `#define AAX_INT_HI(x) *((const_cast<int32_t*>(reinterpret_cast<const int32_t*>(&x)))+1)`

Functions

- template<class GFLOAT > GFLOAT [AAX::ClampToZero](#) (GFLOAT iValue, GFLOAT iClampThreshold)
 - void [AAX::ZeroMemory](#) (void *iPointer, int iNumBytes)
 - void [AAX::ZeroMemoryDW](#) (void *iPointer, int iNumBytes)
 - template<typename T , int N>
-

```

void AAX::Fill (T *iArray, const T *iVal)
• template<typename T , int M, int N> void AAX::Fill (T *iArray, const T *iVal)
• template<typename T , int L, int M, int N> void AAX::Fill (T *iArray, const T *iVal)
• double AAX::fabs (double iVal)
• float AAX::fabs (float iVal)
• float AAX::fabsf (float iVal)
• template<class T > T AAX::AbsMax (const T &iValue, const T &iMax)
• template<class T > T AAX::MinMax (const T &iValue, const T &iMin, const T &iMax)
• template<class T > T AAX::Max (const T &iValue1, const T &iValue2)
• template<class T > T AAX::Min (const T &iValue1, const T &iValue2)
• template<class T > T AAX::Sign (const T &iValue)
• double AAX::PolyEval (double x, const double *coefs, int numCoefs)
• double AAX::CeilLog2 (double iValue)
• void AAX::SinCosMix (float aLinearMix, float &aSinMix, float &aCosMix)

```

15.123.2 Macro Definition Documentation

15.123.2.1 #define AAX_MISCUTILS_H

15.123.2.2 #define AAX_ALIGNMENT_HINT(a, b)

Currently only functional on TI, these word alignments will provide better performance on TI.

15.123.2.3 #define AAX_WORD_ALIGNED_HINT(a)

15.123.2.4 #define AAX_DWORD_ALIGNED_HINT(a)

15.123.2.5 #define AAX_LO(x) x

These macros are used on TI to convert 2 single words accesses to one double word access to provide additional optimization.

15.123.2.6 #define AAX_HI(x) *((const_cast<float*>(&x))+1)

15.123.2.7 #define AAX_INT_LO(x) x

15.123.2.8 #define AAX_INT_HI(x) *((const_cast<int32_t*>(reinterpret_cast<const int32_t*>(&x)))+1)

15.124 AAX_OtherExtensions.dox File Reference

15.125 AAX_Page_Table_Guide.dox File Reference

15.126 AAX_PageTableUtilities.h File Reference

```
#include "AAX_CString.h" #include "AAX.h"
```

Namespaces

- **AAX**

Functions

- template<class T1 , class T2 > bool [AAX::PageTableParameterMappingsAreEqual](#) (const T1 &inL, const T2 &inR)
- template<class T1 , class T2 > bool [AAX::PageTableParameterNameVariationsAreEqual](#) (const T1 &inL, const T2 &inR)
- template<class T1 , class T2 > bool [AAX::PageTablesAreEqual](#) (const T1 &inL, const T2 &inR)
- template<class T > void [AAX::CopyPageTable](#) (T &to, const T &from)
- template<class T > std::vector< std::pair< int32_t, int32_t > > [AAX::FindParameterMappingsInPageTable](#) (const T &inTable, [AAX_CParamID](#) inParameterID)
- template<class T > void [AAX::ClearMappedParameterByID](#) (T &ioTable, [AAX_CParamID](#) inParameterID)

15.127 AAX_ParameterAutomation.dox File Reference

15.128 AAX_ParameterManager.dox File Reference

15.129 AAX_ParameterUpdateProtocol.dox File Reference

15.130 AAX_ParameterUpdateTiming.dox File Reference

15.131 AAX_PlatformOptimizationConstants.h File Reference

15.131.1 Description

Constants descriptor...

Macros

- #define [AAX_PLATFORMOPTIMIZATIONCONSTANTS_H](#)

15.131.2 Macro Definition Documentation

15.131.2.1 #define AAX_PLATFORMOPTIMIZATIONCONSTANTS_H

15.132 AAX_PluginBundleLocation.h File Reference

15.132.1 Description

Utilities for interacting with the host OS.

Namespaces

- [AAX](#)

Functions

Filesystem utilities

- bool [AAX::GetPathToPluginBundle](#) (const char *iBundleName, int iMaxLength, char *oModuleName)
Retrieve the file path of the .aaxplugin bundle.

15.133 AAX_PopStructAlignment.h File Reference

15.133.1 Description

Resets (pops) the struct alignment to its previous value.

See also

[AAX_ALIGN_HOST](#)
[AAX_ALIGN_ALG](#)
[AAX_ALIGN_RESET](#)

Note

Inclusion of this file is mandatory after any 'push' inclusion.

Some compilers do not properly "pop" alignment, so nesting push/pop inclusions is not allowed.

See also

[AAX_Push2ByteStructAlignment.h](#)
[AAX_Push4ByteStructAlignment.h](#)
[AAX_Push8ByteStructAlignment.h](#)

15.134 AAX_Pro_Tools_Guide.dox File Reference

15.135 AAX_Properties.h File Reference

```
#include "AAX.h"
```

15.135.1 Description

Contains IDs for properties that can be added to an [AAX_IPropertyMap](#).

Enumerations

- enum [AAX_EProperty](#) { [AAX_eProperty_NoID](#) = 0, [AAX_eProperty_MinProp](#) = 10, [AAX_eProperty_PlugInSpecPropsBase](#) = 10, [AAX_eProperty_ManufacturerID](#) = 11, [AAX_eProperty_ProductID](#) = 12, [AAX_eProperty_PluginID_Native](#) = 13, [AAX_eProperty_PluginID_RTAS](#) = [AAX_eProperty_PluginID_Native](#), [AAX_eProperty_PluginID_AudioSuite](#) = 14, [AAX_eProperty_PluginID_TI](#) = 15, [AAX_eProperty_PluginID_Deprecated](#) = 18, [AAX_eProperty_Deprecated_Plugin_List](#) = 21, [AAX_eProperty_Related_DSP_Plugin_List](#) = 22, [AAX_eProperty_Related_Native_Plugin_List](#) = 23, [AAX_eProperty_Deprecated_DSP_Plugin_List](#) = 24, [AAX_eProperty_Deprecated_Native_Plugin_List](#) = [AAX_eProperty_Deprecated_Plugin_List](#), [AAX_eProperty_PluginID_ExternalProcessor](#) = 25, [AAX_eProperty_ExternalProcessorTypeID](#) = 26, [AAX_eProperty_ProcessProcPropsBase](#) = 35, [AAX_eProperty_NativeProcessProc](#) = 36, [AAX_eProperty_NativeInstanceInitProc](#) = 37, [AAX_eProperty_NativeBackgroundProc](#) = 38, [AAX_eProperty_TIDLLFileName](#) = 39, [AAX_eProperty_TIProcessProc](#) = 40, [AAX_eProperty_TIInstanceInitProc](#) = 41, [AAX_eProperty_TIBackgroundProc](#) = 42, [AAX_eProperty_GeneralPropsBase](#) = 50, [AAX_eProperty_InputStemFormat](#) = 51, [AAX_eProperty_OutputStemFormat](#) = 52, [AAX_eProperty_DSP_AudioBufferLength](#) = 54, [AAX_eProperty_AudioBufferLength](#) = [AAX_eProperty_DSP_AudioBufferLength](#), [AAX_eProperty_LatencyContribution](#) = 56, [AAX_eProperty_SampleRate](#) = 58, [AAX_eProperty_CanBypass](#) = 60, [AAX_eProperty_SideChainStemFormat](#) = 61, [AAX_eProperty_TI_SharedCycleCount](#) = 62, [AAX_eProperty_TI_InstanceCycleCount](#) = 63, [AAX_eProperty_TI_MaxInstancesPerChip](#) = 64, [AAX_eProperty_TI_ForceAllowChipSharing](#) = 65, [AAX_eProperty_HybridOutputStemFormat](#) = 90, [AAX_eProperty_HybridInputStemFormat](#) = 91, [AAX_eProperty_AudiosuitePropsBase](#) = 100, [AAX_eProperty_UsesRandomAccess](#) = 101, [AAX_eProperty_RequiresAnalysis](#) = 102, [AAX_eProperty_OptionalAnalysis](#) = 103, [AAX_eProperty_AllowPreviewWithoutAnalysis](#)

```
= 104, AAX_eProperty_DestinationTrack = 105, AAX_eProperty_RequestsAllTrackData = 106, AAX_eProperty_ContinuousOnly = 107, AAX_eProperty_MultiInputModeOnly = 108, AAX_eProperty_DisablePreview = 110, AAX_eProperty_SupportsProgressDialog = 111, AAX_eProperty_DoesntIncrOutputSample = 112, AAX_eProperty_NumberOfInputs = 113, AAX_eProperty_NumberOfOutputs = 114, AAX_eProperty_DisableHandles = 115, AAX_eProperty_SupportsSideChainInput = 116, AAX_eProperty_NeedsOutputDithered = 117, AAX_eProperty_DisableAudiosuiteReverse = 118, AAX_eProperty_MaxASProp, AAX_eProperty_GUIBase = 150, AAX_eProperty_UsesClientGUI = 151, AAX_eProperty_MaxGUIProp, AAX_eProperty_MeterBase = 199, AAX_eProperty_Meter_Type = 200, AAX_eProperty_Meter_Orientation = 201, AAX_eProperty_Meter_Ballistics = 202, AAX_eProperty_MaxMeterProp, AAX_eProperty_ConstraintBase = 299, AAX_eProperty_Constraint_Location = 300, AAX_eProperty_Constraint_Topology = 301, AAX_eProperty_Constraint_NeverUnload = 302, AAX_eProperty_Constraint_NeverCache = 303, AAX_eProperty_Constraint_MultiMonoSupport = 304, AAX_eProperty_MaxConstraintProp, AAX_eProperty_FeaturesBase = 305, AAX_eProperty_SupportsSaveRestore = 305, AAX_eProperty_UsesTransport = 306, AAX_eProperty_StoreXMLPageTablesByEffect = 307, AAX_eProperty_StoreXMLPageTablesByType = AAX_eProperty_StoreXMLPageTablesByEffect, AAX_eProperty_RequiresChunkCallsOnMainThread = 308, AAX_eProperty_MaxFeaturesProp, AAX_eProperty_ConstraintBase_2 = 350, AAX_eProperty_Constraint_AlwaysProcess = 351, AAX_eProperty_MaxConstraintProp_2, AAX_eProperty_DebugPropertiesBase = 400, AAX_eProperty_EnableHostDebugLogs = 401, AAX_eProperty_MaxProp, AAX_eProperty_MaxCap = 10000 }
```

The list of properties that can be added to an [AAX_IPropertyMap](#).

Functions

- [AAX_ENUM_SIZE_CHECK\(AAX_EProperty\)](#)

15.135.2 Enumeration Type Documentation

15.135.2.1 enum AAX_EProperty

The list of properties that can be added to an [AAX_IPropertyMap](#).

See [AAX_IPropertyMap::AddProperty\(\)](#) for more information

Sections

- [Plug-In spec properties](#)
- [ProcessProc properties](#)
- [General properties](#)
- [TI-specific properties](#)
- [Offline \(AudioSuite\) properties](#)
- [GUI properties](#)
- [Meter properties](#)
- [Plug-in management constraints](#)

Legacy Porting Notes These property IDs are somewhat analogous to the pluginGestalt system in the legacy SDK, and several [AAX_EProperty](#) values correlate directly with a corresponding legacy plug-in gestalt.

Legacy Porting Notes To ensure session interchange compatibility, make sure the 4 character IDs for [AAX_eProperty_ManufacturerID](#), [AAX_eProperty_ProductID](#), [AAX_eProperty_PluginID_Native](#), and [AAX_eProperty_PluginID_AudioSuite](#) are identical to the legacy SDK's counterpart.

Enumerator

AAX_eProperty_NoID

AAX_eProperty_MinProp

AAX_eProperty_PluginSpecPropsBase

AAX_eProperty_ManufacturerID Four-character osid-style manufacturer identifier. Should be registered with Avid, and must be identical for all plug-ins from the same manufacturer.

- Apply this property at the **ProcessProc** level

Legacy Porting Notes For legacy plug-in session compatibility, this ID should match the Manufacturer ID used in the corresponding legacy plug-ins.

AAX_eProperty_ProductID Four-character osid-style Collection identifier. Must be identical for all ProcessProcs within a single **Effect**.

- Apply this property at the **ProcessProc** level

Legacy Porting Notes For legacy plug-in session compatibility, this ID should match the Product ID used in the corresponding legacy plug-in.

AAX_eProperty_PluginID_Native Four-character osid-style Effect identifier for real-time native audio Effects. All registered plug-in IDs ([AAX_eProperty_PluginID_Native](#), [AAX_eProperty_PluginID_AudioSuite](#), and [AAX_eProperty_PluginID_TI](#)) must be unique across all ProcessProcs registered within a single **Effect**.

Warning

As with all plug-in ID properties, this value must remain constant across all releases of the plug-in which support this Effect configuration. The value of this property should be stored in a constant rather than being calculated at run-time in order to avoid unresolvable compatibility issues with saved sessions which can occur if an ID value is accidentally changed between two plug-in version releases.

- Apply this property at the **ProcessProc** level

Legacy Porting Notes For legacy plug-in session compatibility, this ID should match the Type ID used in the corresponding legacy RTAS plug-in Types.

AAX_eProperty_PluginID_RTAS **Deprecated** Use [AAX_eProperty_PluginID_Native](#)

AAX_eProperty_PluginID_AudioSuite Four-character osid-style Effect identifier for offline native audio Effects. All registered plug-in IDs ([AAX_eProperty_PluginID_Native](#), [AAX_eProperty_PluginID_AudioSuite](#), and [AAX_eProperty_PluginID_TI](#)) must be unique across all ProcessProcs registered within a single **Effect**.

- Apply this property at the **ProcessProc** level

Legacy Porting Notes For legacy plug-in session compatibility, this ID should match the Type ID used in the corresponding legacy AudioSuite plug-in Types.

AAX_eProperty_PluginID_TI Four-character osid-style Effect identifier for real-time TI-accelerated audio Effects. All registered product IDs ([AAX_eProperty_PluginID_Native](#), [AAX_eProperty_PluginID_AudioSuite](#), and [AAX_eProperty_PluginID_TI](#)) must be unique across all ProcessProcs registered within a single **Effect**.

Warning

As with all plug-in ID properties, this value must remain constant across all releases of the plug-in which support this Effect configuration. The value of this property should be stored in a constant rather than being calculated at run-time in order to avoid unresolvable compatibility issues with saved sessions which can occur if an ID value is accidentally changed between two plug-in version releases.

- Apply this property at the **ProcessProc** level

Legacy Porting Notes For legacy plug-in session compatibility, this ID should match the Type ID used in the corresponding legacy TDM plug-in Types.

AAX_eProperty_PluginID_Deprecated Four-character osid-style Effect identifier for a corresponding deprecated Effect. Only one deprecated effect ID may correspond to each valid (non-deprecated) effect ID. To associate a plug-in type with more than one deprecated type, use the following properties instead:

- [AAX_eProperty_Deprecated_DSP_Plugin_List](#)
- [AAX_eProperty_Deprecated_Native_Plugin_List](#)
- Apply this property at the **ProcessProc** level

AAX_eProperty_Deprecated_Plugin_List Deprecated Use [AAX_eProperty_Deprecated_Native_Plugin>List](#) and [AAX_eProperty_Deprecated_DSP_Plugin_List](#) See [AAX_eProperty_PluginID](#) RTAS for an example.

AAX_eProperty_Related_DSP_Plugin_List Specify a list of DSP plug-ins that are related to a plug-in type.

- For example, use this property inside a Native process to tell the host that this plug-in can be used in place of a DSP version.
- This property must be applied at the ProcessProc level and used with the [AAX_IPropertyMap::AddPropertyWithIDArray](#) method, which takes a list of full plug-in identifier specification triads (ManufacturerID, ProductID, PluginID)

AAX_eProperty_Related_Native_Plugin_List Specify a list of Native plug-ins that are related to a plug-in type.

- This property must be applied at the ProcessProc level and used with the [AAX_IPropertyMap::AddPropertyWithIDArray](#) method, which takes a list of full plug-in identifier specification triads (ManufacturerID, ProductID, PluginID)

AAX_eProperty_Deprecated_DSP_Plugin_List Specify a list of DSP plug-ins that are deprecated by a new plug-in type.

- This property must be applied at the ProcessProc level and used with the AddPropertyWithIDArray, which is a list of full plug-in specs (ManufacturerID, ProductID, PluginID)

AAX_eProperty_Deprecated_Native_Plugin_List Specify a list of Native plug-ins that are deprecated by a new plug-in type.

- This property must be applied at the ProcessProc level and used with the AddPropertyWithIDArray, which is a list of full plug-in specs (ManufacturerID, ProductID, PluginID)

AAX_eProperty_PluginID_ExternalProcessor Four-character osid-style Effect identifier for audio effects rendered on external hardware.

Note

This property is not currently used by any AAX plug-in host software

All registered plug-in IDs must be unique across all ProcessProcs registered within a single [Effect](#).

Warning

As with all plug-in ID properties, this value must remain constant across all releases of the plug-in which support this Effect configuration. The value of this property should be stored in a constant rather than being calculated at run-time in order to avoid unresolvable compatibility issues with saved sessions which can occur if an ID value is accidentally changed between two plug-in version releases.

- Apply this property at the **ProcessProc** level

AAX_eProperty_ExternalProcessorTypeID Identifier for the type of the external processor hardware.

See also

[AAX_eProperty_PluginID_ExternalProcessor](#)

The value of this property will be specific to the external processor hardware. Currently there are no public external processor hardware type IDs.

- Apply this property at the **ProcessProc** level

AAX_eProperty_ProcessProcPropsBase
AAX_eProperty_NativeProcessProc Address of a native effect's ProcessProc callback

Data type: [AAX_CProcessProc](#)

For use with [AAX_IComponentDescriptor::AddProcessProc\(\)](#)

AAX_eProperty_NativeInstanceInitProc Address of a native effect's instance initialization callback

Data type: [AAX_CInstanceInitProc](#)

For use with [AAX_IComponentDescriptor::AddProcessProc\(\)](#)

AAX_eProperty_NativeBackgroundProc Address of a native effect's background callback

Data type: [AAX_CBackgroundProc](#)

For use with [AAX_IComponentDescriptor::AddProcessProc\(\)](#)

AAX_eProperty_TIDLLFileName Name of the DLL for a TI effect

Data type: UTF-8 C-string

For use with [AAX_IComponentDescriptor::AddProcessProc\(\)](#)

AAX_eProperty_TIProcessProc Name of a TI effect's ProcessProc callback

Data type: C-string

For use with [AAX_IComponentDescriptor::AddProcessProc\(\)](#)

AAX_eProperty_TIInstanceInitProc Name of a TI effect's instance initialization callback

Data type: C-string

For use with [AAX_IComponentDescriptor::AddProcessProc\(\)](#)

AAX_eProperty_TIBackgroundProc Name of a TI effect's background callback

Data type: C-string

For use with [AAX_IComponentDescriptor::AddProcessProc\(\)](#)

AAX_eProperty_GeneralPropsBase

AAX_eProperty_InputStemFormat Input stem format. One of [AAX_EStemFormat](#).

- Apply this property at the **ProcessProc** level

For offline processing, use [AAX_eProperty_NumberOfInputs](#)

AAX_eProperty_OutputStemFormat Output stem format. One of [AAX_EStemFormat](#).

- Apply this property at the **ProcessProc** level

For offline processing, use [AAX_eProperty_NumberOfOutputs](#)

AAX_eProperty_DSP_AudioBufferLength Audio buffer length for DSP processing callbacks. One of [AA↔X_EAudioBufferLengthDSP](#).

- Apply this property at the **ProcessProc** level
- This property is only applicable to DSP algorithms

AAX_eProperty_AudioBufferLength **Deprecated** Use [AAX_eProperty_DSP_AudioBufferLength](#)

AAX_eProperty_LatencyContribution Default latency contribution of a given processing callback, in samples.

- Apply this property at the **ProcessProc** level

Unlike most properties, an Effect's latency contribution may also be changed dynamically at runtime. This is done via [AAX_IController::SetSignalLatency\(\)](#). Dynamic latency reporting may not be recognized by the host application in all circumstances, however, so Effects should always define any nonzero initial latency value using [AAX_eProperty_LatencyContribution](#)

Host Compatibility Notes Maximum delay compensation limits will vary from host to host. If your plug-in exceeds the delay compensation sample limit for a given AAX host then you should note this limitation in your user documentation. Example limits:

- Pro Tools 9 and higher: 16,383 samples at 44.1/48 kHz, 32,767 samples at 88.2/96 kHz, or 65,534 samples at 176.4/192 kHz
- Media Composer 8.1 and higher: 16,383 samples at 44.1/48 kHz, 32,767 samples at 88.2/96 kHz

AAX_eProperty_SampleRate Specifies which sample rates the Effect supports. A mask of [AAX_ESampleRateMask](#).

- Apply this property at the **ProcessProc** level

See also

[AAX_IComponentDescriptor::AddSampleRate\(\)](#)

AAX_eProperty_CanBypass The plug-in supports a Master Bypass control.

- Apply this property at the **ProcessProc** level

Legacy Porting Notes Was pluginGestalt_CanBypass.

Todo This property should always be enabled for AAX plug-ins

AAX_eProperty_SideChainStemFormat Side chain stem format. One of [AAX_EStemFormat](#).

Host Compatibility Notes Currently Pro Tools supports only [AAX_eStemFormat_Mono](#) side chain inputs

- Apply this property at the **ProcessProc** level

Host Compatibility Notes [AAX_eProperty_SideChainStemFormat](#) is not currently implemented in DAE or AAE

AAX_eProperty_TI_SharedCycleCount Shared cycle count (outer, per clump, loop overhead)

- Apply this property at the **ProcessProc** level
- This property is only applicable to DSP algorithms

AAX_eProperty_TI_InstanceCycleCount Instance cycle count (inner, per instance, loop overhead)

- Apply this property at the **ProcessProc** level
- This property is only applicable to DSP algorithms

AAX_eProperty_TI_MaxInstancesPerChip Maximum number of instances of this plug-in that can be loaded on a chip. This property is only used for DMA and background thread-enabled plug-ins.

- Apply this property at the **ProcessProc** level
- This property is only applicable to DSP algorithms

AAX_eProperty_TI_ForceAllowChipSharing Allow different plug-in types to share the same DSP even if [AAX_eProperty_TI_MaxInstancesPerChip](#) is declared. In general, this is not desired behavior. However, this can be useful if your plug-in instance counts are bound by a system constraint other than CPU usage and you require chip-sharing between instances of different types of the plug-in.

Note

In addition to defining this property, the types which will share allocations on the same DSP chip must be compiled into the same ELF DLL file.

- Apply this property at the **ProcessProc** level
- This property is only applicable to DSP algorithms

AAX_eProperty_HybridOutputStemFormat Hybrid Output stem format. One of [AAX_EStemFormat](#). This property represents the stem format for the audio channels that are sent from the ProcessProc callback to the [AAX_IEffectParameters::RenderAudio_Hybrid\(\)](#) method

- Apply this property at the **ProcessProc** level
- Normally plugins will set this to the same thing as [AAX_eProperty_InputStemFormat](#)

AAX_eProperty_HybridInputStemFormat Hybrid Input stem format. One of [AAX_EStemFormat](#). This property represents the stem format for the audio channels that are sent from the [AAX_IEffectParameters::RenderAudio_Hybrid\(\)](#) method to the ProcessProc callback

- Apply this property at the **ProcessProc** level
- Normally plugins will set this to the same thing as [AAX_eProperty_OutputStemFormat](#)

AAX_eProperty_AudiosuitePropsBase**AAX_eProperty_UsesRandomAccess** The Effect requires random access to audio data.

- Apply this property at the [AAX_IEffectDescriptor](#) level
- This property is only applicable to [Host Processor](#) algorithms

Legacy Porting Notes Was pluginGestalt_UsesRandomAccess

AAX_eProperty_RequiresAnalysis The Effect requires an analysis pass.

- Apply this property at the [AAX_IEffectDescriptor](#) level
- This property is only applicable to offline processing

Legacy Porting Notes Was pluginGestalt_RequiresAnalysis

AAX_eProperty_OptionalAnalysis The Effect supports an analysis pass, but does not require it.

Host Compatibility Notes In Media Composer, optional analysis will also be performed automatically before each channel is rendered. See [MCDEV-2904](#)

- Apply this property at the [AAX_IEffectDescriptor](#) level
- This property is only applicable to offline processing

Legacy Porting Notes Was pluginGestalt_OptionalAnalysis

AAX_eProperty_AllowPreviewWithoutAnalysis The Effect requires analysis, but is also allowed to preview.

- Apply this property at the [AAX_IEffectDescriptor](#) level
- This property is only applicable to offline processing

Legacy Porting Notes Was pluginGestalt_AnalyzeOnTheFly

AAX_eProperty_DestinationTrack Informs the host application to reassign output to a different track.

- Apply this property at the [AAX_IEffectDescriptor](#) level
- This property is only applicable to offline processing

Host Compatibility Notes This property is not supported on Media Composer

Legacy Porting Notes Was pluginGestalt_DestinationTrack

AAX_eProperty_RequestsAllTrackData The host should make all of the processed track's data available to the Effect.

- Apply this property at the [AAX_IEffectDescriptor](#) level
- This property is only applicable to [Host Processor](#) algorithms

Legacy Porting Notes Was pluginGestalt_RequestsAllTrackData

AAX_eProperty_ContinuousOnly The Effect only processes on continuous data and does not support 'clip by clip' rendering.

- Apply this property at the [AAX_IEffectDescriptor](#) level
- This property is only applicable to offline processing

Legacy Porting Notes Was pluginGestalt_ContinuousOnly

AAX_eProperty_MultiInputModeOnly The Effect wants multi-input mode only (no mono mode option)

- Apply this property at the [AAX_IEffectDescriptor](#) level
- This property is only applicable to offline processing

Legacy Porting Notes Was pluginGestalt_MultiInputModeOnly

AAX_eProperty_DisablePreview The Effect does not support preview.

- Apply this property at the [AAX_IEffectDescriptor](#) level
- This property is only applicable to offline processing

Legacy Porting Notes Was pluginGestalt_DisablePreview

AAX_eProperty_SupportsProgressDialog The Effect controls its own progress dialog. If unsupported, the host manages this dialog.

- Apply this property at the [AAX_IEffectDescriptor](#) level
- This property is only applicable to offline processing

Legacy Porting Notes Was pluginGestalt_SupportsProgressDialog

AAX_eProperty_DoesntIncrOutputSample The Effect may not increment its output sample during some rendering calls.

- Apply this property at the [AAX_IEffectDescriptor](#) level

- This property is only applicable to [Host Processor](#) algorithms

Legacy Porting Notes Was pluginGestalt_DoesntIncrOutputSample

AAX_eProperty_NumberOfInputs The number of input channels that the plug-in supports.

- Apply this property at the [AAX_IEffectDescriptor](#) level
- This property is only applicable to [Host Processor](#) algorithms

For real-time processing, use [AAX_eProperty_InputStemFormat](#)

AAX_eProperty_NumberOfOutputs The number of output channels that the plug-in supports.

- Apply this property at the [AAX_IEffectDescriptor](#) level
- This property is only applicable to [Host Processor](#) algorithms

For real-time processing, use [AAX_eProperty_OutputStemFormat](#)

AAX_eProperty_DisableHandles Prevents the application of rendered region handles by the host.

- Apply this property at the [AAX_IEffectDescriptor](#) level
- This property is only applicable to offline processing

AAX_eProperty_SupportsSideChainInput Tells the host that the plug-in supports side chain inputs.

- Apply this property at the [AAX_IEffectDescriptor](#) level
- This property is only applicable to offline processing

AAX_eProperty_NeedsOutputDithered Requests that the host apply dithering to the Effect's output.

- Apply this property at the [AAX_IEffectDescriptor](#) level
- This property is only applicable to offline processing

Legacy Porting Notes Was pluginGestalt_NeedsOutputDithered

AAX_eProperty_DisableAudiosuiteReverse The plug-in supports audiosuite reverse. By default, all reverb and delay plug-ins support this feature. If a plug-in needs to opt out of this feature, they can set this property to true.

- Apply this property at the [AAX_IEffectDescriptor](#) level
- This property is only applicable to offline processing

AAX_eProperty_MaxASProp

AAX_eProperty_GUIBase

AAX_eProperty_UsesClientGUI Requests a host-generated GUI based on the Effect's parameters. Use this property while your plug-in is in development to test the plug-in's data model and algorithm before its GUI has been created, or when troubleshooting problems to isolate the data model and algorithm operation from the plug-in's GUI.

- Apply this property at the [ProcessProc](#) level

Host Compatibility Notes Currently supported by Pro Tools only

Note

See [PTSW-189725 / PT-218397](#)

AAX_eProperty_MaxGUIProp

AAX_eProperty_MeterBase

AAX_eProperty_Meter_Type Indicates meter type as one of [AAX_EMeterType](#).

- Apply this property at the [AAX_IEffectDescriptor::AddMeterDescription\(\)](#) level

AAX_eProperty_Meter_Orientation Indicates meter orientation as one of [AAX_EMeterOrientation](#).

- Apply this property at the [AAX_IEffectDescriptor::AddMeterDescription\(\)](#) level

AAX_eProperty_Meter_Ballistics Indicates meter ballistics preference as one of [AAX_EMeterBallisticType](#).

- Apply this property at the [AAX_IEffectDescriptor::AddMeterDescription\(\)](#) level

AAX_eProperty_MaxMeterProp

AAX_eProperty_ConstraintBase

AAX_eProperty_Constraint_Location Constraint on the algorithm's location, as a mask of [AAX_EConstraintLocationMask](#).

- Apply this property at the **ProcessProc** level

AAX_eProperty_Constraint_Topo Constraint on the topology of the Effect's modules, as one of [AAX_EConstraintTopology](#).

- Apply this property at the [AAX_IEffectDescriptor](#) level

AAX_eProperty_Constraint_NeverUnload Tells the host that it should never unload the plug-in binary.

- Apply this property at the [AAX_IEffectDescriptor](#) level

Host Compatibility Notes [AAX_eProperty_Constraint_NeverUnload](#) is not currently implemented in DAE or AAE

AAX_eProperty_Constraint_NeverCache Tells the host that it should never cache the plug-in binary. Only use this if required as there is a performance penalty on launch to not use the Cache. Set this property to 1, if you really need to not cache. Default is 0. This property should be applied at the collection level as it affects the entire bundle.

- Apply this property at the [AAX_IEffectDescriptor](#) level

AAX_eProperty_Constraint_MultiMonoSupport Indicates whether or not the plug-in supports multi-mono configurations (true/false)

Note

Multi-mono mode may not work as expected for VIs and other plug-ins which rely on non-global MIDI input. Depending on the host, multi-mono instances may not all be automatically connected to the same MIDI port upon instantiation. Therefore it is recommended to set this property to 0 for any plug-ins if this lack of automatic connection may confuse users.

- Apply this property at the **ProcessProc** level

AAX_eProperty_MaxConstraintProp

AAX_eProperty_FeaturesBase

AAX_eProperty_SupportsSaveRestore Indicates whether or not the plug-in supports Save/Restore features. (true/false)

- Apply this property to show or hide the Settings section in the plug-in window.
- This property value is true by default.

Legacy Porting Notes Was pluginGestalt_SupportsSaveRestore

AAX_eProperty_UsesTransport Indicates whether or not the plug-in uses transport requests. (true/false)

- Apply this property if your plug-in uses [AAX_ITransport](#) class.
- Apply this property at the [AAX_IEffectDescriptor](#) level

AAX_eProperty_StoreXMLPageTablesByEffect This property specifies whether the plug-in bundle contains an XML file per plug-in type. AAX plug-ins always provide XML page table data via external files referenced by [AAX_eResourceType_PageTable](#). If [AAX_eProperty_StoreXMLPageTablesByEffect](#) is not defined or is set to 0 (the default) then the host may assume that all Effects in the collection use the same XML page table file. If this property is set to a non-zero value, the plug-in may describe a different [AAX_eResourceType_PageTable](#) for each separate Effect.

This property needs to be set at the collection level.

AAX_eProperty_StoreXMLPageTablesByType **Deprecated** Use [AAX_eProperty_StoreXMLPageTablesByEffect](#)

AAX_eProperty_RequiresChunkCallsOnMainThread Indicates whether the plug-in supports SetChunk and GetChunk calls on threads other than the main thread. It is actually important for plug-ins to support these calls on non-main threads, so that is the default. However, in response to a few companies having issues with this, we have decided to support this constraint for now. property value should be set to true if you need Chunk calls on the main thread.

Values: 0 (off, default), 1 (on)

- Apply this property at the [AAX_IEffectDescriptor](#) level

AAX_eProperty_MaxFeaturesProp***AAX_eProperty_ConstraintBase_2***

AAX_eProperty_Constraint_AlwaysProcess Indicates that the plug-in's processing should never be disabled by the host (true/false). Some hosts will disable processing for plug-in chains in certain circumstances to conserve system resources, e.g. when the chains' output drops to silence for an extended period.

Note

This property may impact performance of other plug-ins. For example, the Dynamic Plug-In Processing feature in Pro Tools operates over chains of plug-ins rather than single instances; any plug-in that defines ***AAX_eProperty_Constraint_AlwaysProcess*** will force its entire signal chain to continue processing. Therefore it is important to avoid using this property unless features such as Dynamic Plug-In Processing are actually interfering in some way with the operation of the plug-in.

- This property value is false by default.
- Apply this property at the [AAX_IEffectDescriptor](#) level

AAX_eProperty_MaxConstraintProp_2***AAX_eProperty_DebugPropertiesBase***

AAX_eProperty_EnableHostDebugLogs Enables host debug logging for this plug-in. This logging is made via DigiTrace using the DTF_AAXHOST facility, generally at DTP_LOW priority

- It is recommended to set this property to 1 for debug builds and to 0 for release builds of a plug-in
- Apply this property at the [AAX_IEffectDescriptor](#) level

AAX_eProperty_MaxProp***AAX_eProperty_MaxCap***

15.135.3 Function Documentation

15.135.3.1 AAX_ENUM_SIZE_CHECK(*AAX_EProperty*)

15.136 AAX_Push2ByteStructAlignment.h File Reference

15.136.1 Description

Set the struct alignment to 2-byte. This file will throw an error on platforms that do not support 2-byte alignment (i.e. TI DSPs)

When setting the alignment for a struct in order to match a particular environment (e.g. host/plug-in binary compatibility) the following macros are recommended:

- [AAX_ALIGN_FILE_HOST](#)
- [AAX_ALIGN_FILE_ALG](#)
- [AAX_ALIGN_FILE_RESET](#)

15.136.2 Usage notes

- Always follow an inclusion of this file with a matching inclusion of [AAX_PopStructAlignment.h](#)
- Do not place other file #include after this file. For example:

```
// HeaderFile1.h
#include AAX_Push2ByteStructAlignment.h
#include HeaderFile2.h // this file now has 2-byte alignment also!!
// HeaderFile1.h definitions...
#include AAX_PopStructAlignment.h
// end HeaderFile1.h
```

This will cause problems if HeaderFile2.h is included elsewhere without the 2-byte alignment which will manifest as hard to find run-time bugs. The proper usage is:

```
// HeaderFile1.h
#include HeaderFile2.h
#include AAX_Push2ByteStructAlignment.h
// HeaderFile1.h definitions...
#include AAX_PopStructAlignment.h
// end HeaderFile1.h
```

See also

[AAX_Push4ByteStructAlignment.h](#)
[AAX_Push8ByteStructAlignment.h](#)
[AAX_PopStructAlignment.h](#)

15.137 AAX_Push4ByteStructAlignment.h File Reference

15.137.1 Description

Set the struct alignment to 4-byte.

When setting the alignment for a struct in order to match a particular environment (e.g. host/plug-in binary compatibility) the following macros are recommended:

- [AAX_ALIGN_FILE_HOST](#)
- [AAX_ALIGN_FILE_ALG](#)
- [AAX_ALIGN_FILE_RESET](#)

15.137.2 Usage notes

- Always follow an inclusion of this file with a matching inclusion of [AAX_PopStructAlignment.h](#)
- Do not place other file #include after this file. For example:

```
// HeaderFile1.h
#include AAX_Push4ByteStructAlignment.h
#include HeaderFile2.h // this file now has 4-byte alignment also!!
// HeaderFile1.h definitions...
#include AAX_PopStructAlignment.h
// end HeaderFile1.h
```

This will cause problems if HeaderFile2.h is included elsewhere without the 4-byte alignment which will manifest as hard to find run-time bugs. The proper usage is:

```
// HeaderFile1.h
#include HeaderFile2.h
#include AAX_Push4ByteStructAlignment.h
// HeaderFile1.h definitions...
#include AAX_PopStructAlignment.h
// end HeaderFile1.h
```

See also

[AAX_Push2ByteStructAlignment.h](#)
[AAX_Push8ByteStructAlignment.h](#)
[AAX_PopStructAlignment.h](#)

15.138 AAX_Push8ByteStructAlignment.h File Reference

15.138.1 Description

Set the struct alignment to 8-byte.

When setting the alignment for a struct in order to match a particular environment (e.g. host/plug-in binary compatibility) the following macros are recommended:

- [AAX_ALIGN_FILE_HOST](#)
- [AAX_ALIGN_FILE_ALG](#)
- [AAX_ALIGN_FILE_RESET](#)

15.138.2 Usage notes

- Always follow an inclusion of this file with a matching inclusion of [AAX_PopStructAlignment.h](#)
- Do not place other file #include after this file. For example:

```
// HeaderFile1.h
#include AAX_Push8ByteStructAlignment.h
#include HeaderFile2.h // this file now has 8-byte alignment also!!
// HeaderFile1.h definitions...
#include AAX_PopStructAlignment.h
// end HeaderFile1.h
```

This will cause problems if HeaderFile2.h is included elsewhere without the 8-byte alignment which will manifest as hard to find run-time bugs. The proper usage is:

```
// HeaderFile1.h
#include HeaderFile2.h
#include AAX_Push8ByteStructAlignment.h
// HeaderFile1.h definitions...
#include AAX_PopStructAlignment.h
// end HeaderFile1.h
```

See also

[AAX_Push2ByteStructAlignment.h](#)
[AAX_Push4ByteStructAlignment.h](#)
[AAX_PopStructAlignment.h](#)

15.139 AAX_Quantize.h File Reference

```
#include "AAX.h"
#include "AAX_PlatformOptimizationConstants.h"
#include "AAX_X_Constants.h"
#include <xmmmintrin.h>
#include <pmmmintrin.h>
#include <tmmintrin.h>
```

15.139.1 Description

Quantization utilities.

Namespaces

- [AAX](#)

Macros

- `#define AAX_QUANTIZE_H`

Functions

- `int32_t AAX::FastRound2Int32 (double iVal)`
Round to Int32.
- `int32_t AAX::FastRound2Int32 (float iVal)`
Round to Int32.
- `int32_t AAX::FastRndDbl2Int32 (double iVal)`
- `int32_t AAX::FastTrunc2Int32 (double iVal)`
Float to Int conversion with truncation.
- `int32_t AAX::FastTrunc2Int32 (float iVal)`
Float to Int conversion with truncation.
- `int64_t AAX::FastRound2Int64 (double iVal)`
Round to Int64.

15.139.2 Macro Definition Documentation

15.139.2.1 `#define AAX_QUANTIZE_H`

15.140 AAX_RandomGen.h File Reference

```
#include <stdlib.h>#include <time.h>#include <stdint.h>#include "AAX_PlatformOptimizationConstants.h"
```

```
#include "AAX_Constants.h"
```

15.140.1 Description

Functions for calculating pseudo-random numbers.

Namespaces

- [AAX](#)

Macros

- `#define AAX_RANDOMGEN_H`

Functions

- `int32_t AAX::GetInt32RPDF (int32_t *iSeed)`
- `int32_t AAX::GetFastInt32RPDF (int32_t *iSeed)`
CALL: Calculate pseudo-random 32 bit number based on linear congruential method.
- `float AAX::GetRPDFWithAmplitudeOneHalf (int32_t *iSeed)`
- `float AAX::GetRPDFWithAmplitudeOne (int32_t *iSeed)`
- `float AAX::GetFastRPDFWithAmplitudeOne (int32_t *iSeed)`
- `float AAX::GetTPDFWithAmplitudeOne (int32_t *iSeed)`

Variables

- `const float AAX::cSeedDivisor = 1/127773.0f`
- `const int32_t AAX::cInitialSeedValue =0x00F54321`

15.140.2 Macro Definition Documentation

15.140.2.1 `#define AAX_RANDOMGEN_H`

15.141 AAX_RelatedTypes.dox File Reference

15.142 AAX_SampleRateUtils.h File Reference

15.142.1 Description

Description.

Enumerations

- `enum ESRUtils { eSRUtils_48kRangeCoarse = 48000, eSRUtils_96kRangeCoarse = 96000, eSRUtils_192kRangeCoarse = 192000, eSRUtils_48kRangeMin = 0, eSRUtils_48kRangeMax = 51000, eSRUtils_96kRangeMin = eSRUtils_48kRangeMax+1, eSRUtils_96kRangeMax = 102000, eSRUtils_192kRangeMin = eSRUtils_96kRangeMax+1, eSRUtils_192kRangeMax = 204000, eSRUtils_48kIndex = 0, eSRUtils_96kIndex = 1, eSRUtils_192kIndex = 2 }`

Functions

- int [CoarseSampleRate](#) (int iRate)
- int [CoarseSampleRateFactor](#) (int iRate)
- int [CoarseSampleRateIndex](#) (int iRate)

15.142.2 Enumeration Type Documentation

15.142.2.1 enum ESRUtils

Enumerator

`eSRUtils_48kRangeCoarse`
`eSRUtils_96kRangeCoarse`
`eSRUtils_192kRangeCoarse`
`eSRUtils_48kRangeMin`
`eSRUtils_48kRangeMax`
`eSRUtils_96kRangeMin`
`eSRUtils_96kRangeMax`
`eSRUtils_192kRangeMin`
`eSRUtils_192kRangeMax`
`eSRUtils_48kIndex`
`eSRUtils_96kIndex`
`eSRUtils_192kIndex`

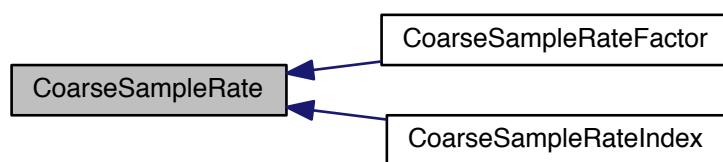
15.142.3 Function Documentation

15.142.3.1 int CoarseSampleRate (int iRate) [inline]

References `eSRUtils_192kRangeCoarse`, `eSRUtils_192kRangeMax`, `eSRUtils_192kRangeMin`, `eSRUtils_48kRangeCoarse`, `eSRUtils_48kRangeMax`, `eSRUtils_48kRangeMin`, `eSRUtils_96kRangeCoarse`, `eSRUtils_96kRangeMax`, and `eSRUtils_96kRangeMin`.

Referenced by `CoarseSampleRateFactor()`, and `CoarseSampleRateIndex()`.

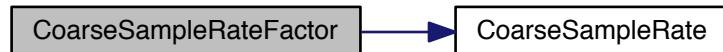
Here is the caller graph for this function:



15.142.3.2 int CoarseSampleRateFactor(int iRate) [inline]

References CoarseSampleRate(), and eSRUtils_48kRangeCoarse.

Here is the call graph for this function:



15.142.3.3 int CoarseSampleRateIndex(int iRate) [inline]

References CoarseSampleRate(), eSRUtils_192kRangeCoarse, eSRUtils_48kRangeCoarse, and eSRUtils_96kRangeCoarse.

Here is the call graph for this function:



15.143 AAX_SDK_ChangeLog.dox File Reference

15.144 AAX_SDK_ExamplePlugIns.dox File Reference

15.145 AAX_SDK_GUIExtensions.dox File Reference

15.146 AAX_SliderConversions.h File Reference

```
#include "AAX.h" #include <algorithm> #include <stdint.h>
```

15.146.1 Description

Legacy utilities for converting parameter values to and from the normalized full-scale 32-bit fixed domain that was used for RTAS/TDM plug-ins.

Legacy Porting Notes These utilities may be required in order to maintain settings chunk compatibility with plug-ins that were ported from the legacy RTAS/TDM format.

Note

AAX does not provide facilities for converting to and from extended80 data types. If you use these types in your plug-in settings then you must provide your own chunk data parsing routines.

Macros

- `#define AAX_SLIDERCONVERSIONS_H`
- `#define AAX_LIMIT(v1, firstVal, secondVal) ((secondVal > firstVal) ? (std::max)((std::min)(v1,secondVal),firstVal) : (std::min)((std::max)(v1,secondVal),firstVal))`

Functions

- `int32_t LongControlToNewRange (int32_t aValue, int32_t rangeMin, int32_t rangeMax)`
Convert from int32_t control value 0x80000000...0x7FFFFFFF to a int32_t ranging from rangeMin to rangeMax (linear)
- `int32_t LongToLongControl (int32_t aValue, int32_t rangeMin, int32_t rangeMax)`
Convert from int32_t control value 0x80000000...0x7FFFFFFF to an double ranging from firstVal to secondVal (linear)
- `double LongControlToDouble (int32_t aValue, double firstVal, double secondVal)`
Convert from int32_t control value 0x80000000...0x7FFFFFFF to an double ranging from firstVal to secondVal (linear)
- `int32_t DoubleToLongControl (double aValue, double firstVal, double secondVal)`
Convert from an double ranging from firstVal to secondVal (linear) to int32_t control value 0x80000000...0x7FFFFFFF.
- `int32_t DoubleToLongControlNonlinear (double aValue, double *minVal, double *rangePercent, int32_t numRanges)`
- `double LongControlToDoubleNonlinear (int32_t aValue, double *minVal, double *rangePercent, int32_t numRanges)`
- `double LongControlToLogDouble (int32_t aValue, double minVal, double maxVal)`
Convert from int32_t control value 0x80000000...0x7FFFFFFF to an double ranging from minVal to maxVal (logarithmic)
- `int32_t LogDoubleToLongControl (double aValue, double minVal, double maxVal)`
Convert from an double ranging from minVal to maxVal (logarithmic) to int32_t control value 0x80000000...0x7FFFFFFF.

15.146.2 Macro Definition Documentation**15.146.2.1 #define AAX_SLIDERCONVERSIONS_H****15.146.2.2 #define AAX_LIMIT(v1, firstVal, secondVal) ((secondVal > firstVal) ? (std::max)((std::min)(v1,secondVal),firstVal) : (std::min)((std::max)(v1,secondVal),firstVal))****15.146.3 Function Documentation****15.146.3.1 int32_t LongControlToNewRange (int32_t aValue, int32_t rangeMin, int32_t rangeMax)****15.146.3.2 int32_t LongToLongControl (int32_t aValue, int32_t rangeMin, int32_t rangeMax)**

Convert from int32_t control value 0x80000000...0x7FFFFFFF to a int32_t ranging from rangeMin to rangeMax (linear)

15.146.3.3 double LongControlToDouble (int32_t aValue, double firstVal, double secondVal)

Convert from int32_t control value 0x80000000...0x7FFFFFFF to an double ranging from firstVal to secondVal (linear)

15.146.3.4 int32_t DoubleToLongControl (double aValue, double firstVal, double secondVal)

Convert from an double ranging from firstVal to secondVal (linear) to int32_t control value 0x80000000...0x7FFF←FFFF.

15.146.3.5 int32_t DoubleToLongControlNonlinear (double aValue, double * minVal, double * rangePercent, int32_t numRanges)

15.146.3.6 double LongControlToDoubleNonlinear (int32_t aValue, double * minVal, double * rangePercent, int32_t numRanges)

15.146.3.7 double LongControlToLogDouble (int32_t aValue, double minVal, double maxVal)

Convert from int32_t control value 0x80000000...0x7FFFFFFF to an double ranging from minVal to maxVal (logarithmic)

Note

This is LOGARITHMIC, so minVal & maxVal have to be > zero!

15.146.3.8 int32_t LogDoubleToLongControl (double aValue, double minVal, double maxVal)

Convert from an double ranging from minVal to maxVal (logarithmic) to int32_t control value 0x80000000...0x7FFF←FFFF.

Note

This is LOGARITHMIC, so minVal & maxVal have to be > zero!

15.147 AAX_StringUtilities.h File Reference

```
#include "AAX.h"
#include "AAX.Enums.h"
#include <string>
#include "AAX←
StringUtilities.hpp"
```

15.147.1 Description

Various string utility definitions for AAX Native.

Namespaces

- [AAX](#)

Macros

- `#define AAXLibrary_AAX_StringUtilities_h`

Functions

- `void AAX::GetCStringOfLength (char *stringOut, const char *stringIn, int32_t aMaxChars)`
=====
- `int32_t AAX::Caseless_strcmp (const char *cs, const char *ct)`

- std::string [AAX::Binary2String](#) (uint32_t binaryValue, int32_t numBits)
- uint32_t [AAX::String2Binary](#) (const [AAX_IString](#) &s)
- bool [AAX::IsASCII](#) (char inChar)
- bool [AAX::IsFourCharASCII](#) (uint32_t inFourChar)
- std::string [AAX::AsStringFourChar](#) (uint32_t inFourChar)
- std::string [AAX::AsStringPropertyValue](#) ([AAX_EProperty](#) inProperty, [AAX_CPropertyValue](#) inPropertyValue)
- std::string [AAX::AsStringInt32](#) (int32_t inInt32)
- std::string [AAX::AsStringUInt32](#) (uint32_t inUInt32)
- std::string [AAX::AsStringIDTriad](#) (const [AAX_SPlugInIdentifierTriad](#) &inIDTriad)
- std::string [AAX::AsStringStemFormat](#) ([AAX_EStemFormat](#) inStemFormat, bool inAbbreviate=false)
- std::string [AAX::AsStringStemChannel](#) ([AAX_EStemFormat](#) inStemFormat, uint32_t inChannelIndex, bool inAbbreviate)
- std::string [AAX::AsStringResult](#) ([AAX_Result](#) inResult)

15.147.2 Macro Definition Documentation

15.147.2.1 #define [AAXLibrary_AAX_StringUtilities_h](#)

15.148 AAX_StringUtilities.hpp File Reference

```
#include "AAX_IString.h"#include "AAX_Errors.h"#include "AAX_Assert.h"#include
<stdlib.h>#include <algorithm>#include <vector>#include <string>#include
<cstring>
```

Macros

- #define [DEFINE_AAX_ERROR_STRING](#)(X) if (X == inResult) { return std::string(#X); }

15.148.1 Macro Definition Documentation

15.148.1.1 #define [DEFINE_AAX_ERROR_STRING\(X \)](#) if (X == inResult) { return std::string(#X); }

Referenced by [AAX::AsStringResult\(\)](#).

15.149 AAX_TI_Guide.dox File Reference

15.150 AAX_UIDs.h File Reference

```
#include "acfbasetypes.h"#include "defineacfuid.h"#include "acfuids.h"
```

15.150.1 Description

Unique identifiers for AAX/ACF interfaces.

Variables

AAX Host interface IDs

- const acfIID [AAXComplID_HostServices](#)
ACF component ID for [AAX_IHostServices](#) components.

- const acflID IID_IAAXHostServicesV1
 ACF interface ID for [AAC_IACFHostServices](#).
- const acflID IID_IAAXHostServicesV2
 ACF interface ID for [AAC_IACFHostServices_V2](#).
- const acflID IID_IAAXHostServicesV3
 ACF interface ID for [AAC_IACFHostServices_V3](#).
- const acflID AAXComplID_AAXCollection
 ACF component ID for [AAC_ICollection](#) components.
- const acflID IID_IAAXCollectionV1
 ACF interface ID for [AAC_IACFCollection](#).
- const acflID AAXComplID_AAXEffectDescriptor
 ACF component ID for [AAC_IEffectDescriptor](#) components.
- const acflID IID_IAAXEffectDescriptorV1
 ACF interface ID for [AAC_IACFEffectorDescriptor](#).
- const acflID IID_IAAXEffectDescriptorV2
 ACF interface ID for [AAC_IACFEffectorDescriptor_V2](#).
- const acflID AAXComplID_AAXComponentDescriptor
 ACF component ID for [AAC_IComponentDescriptor](#) components.
- const acflID IID_IAAXComponentDescriptorV1
 ACF interface ID for [AAC_IACFCOMPONENTDescriptor](#).
- const acflID IID_IAAXComponentDescriptorV2
 ACF interface ID for [AAC_IACFCOMPONENTDescriptor_V2](#).
- const acflID IID_IAAXComponentDescriptorV3
 ACF interface ID for [AAC_IACFCOMPONENTDescriptor_V3](#).
- const acflID AAXComplID_AAXPropertyMap
 ACF component ID for [AAC_IPROPERTYMAP](#) components.
- const acflID IID_IAAXPropertyMapView1
 ACF interface ID for [AAC_IACFPROPERTYMAP](#).
- const acflID IID_IAAXPropertyMapView2
 ACF interface ID for [AAC_IACFPROPERTYMAP_V2](#).
- const acflID IID_IAAXPropertyMapView3
 ACF interface ID for [AAC_IACFPROPERTYMAP_V3](#).
- const acflID AAXComplID_HostProcessorDelegate
 ACF component ID for [AAC_IHOSTPROCESSORDELEGATE](#) components.
- const acflID IID_IAAXHostProcessorDelegateV1
 ACF interface ID for [AAC_IACFHOSTPROCESSORDELEGATE](#).
- const acflID IID_IAAXHostProcessorDelegateV2
 ACF interface ID for [AAC_IACFHOPSTPROCESSORDELEGATE_V2](#).
- const acflID IID_IAAXHostProcessorDelegateV3
 ACF interface ID for [AAC_IACFHOPSTPROCESSORDELEGATE_V3](#).
- const acflID AAXComplID_AutomationDelegate
 ACF component ID for [AAC_IAUTOMATIONDELEGATE](#) components.
- const acflID IID_IAAXAutomationDelegateV1
 ACF interface ID for [AAC_IACFAUTOMATIONDELEGATE](#).
- const acflID AAXComplID_Controller
 ACF component ID for [AAC_ICONTROLLER](#) components.
- const acflID IID_IAAXControllerV1
 ACF interface ID for [AAC_IACFCONTROLLER](#).
- const acflID IID_IAAXControllerV2
 ACF interface ID for [AAC_IACFCONTROLLER_V2](#).
- const acflID IID_IAAXControllerV3
 ACF interface ID for [AAC_IACFCONTROLLER_V3](#).
- const acflID AAXComplID_PageTableController
 ACF component ID for AAC page table controller components.
- const acflID IID_IAAXPageTableController
 ACF interface ID for [AAC_IACFPAGETABLECONTROLLER](#).
- const acflID IID_IAAXPageTableControllerV2
 ACF interface ID for [AAC_IACFPAGETABLECONTROLLER_V2](#).
- const acflID AAXComplID_PrivateDataAccess

- const acfIID IID_IAAXPrivateDataAccessV1
ACF component ID for [AAX_IPrivateDataAccess](#) components.
- const acfIID IID_AAXComplID_ViewContainer
ACF component ID for [AAX_IViewContainer](#) components.
- const acfIID IID_IAAXViewContainerV1
ACF interface ID for [AAX_IACFViewContainer](#).
- const acfIID IID_IAAXViewContainerV2
ACF interface ID for [AAX_IACFViewContainer_V2](#).
- const acfIID IID_AAXComplID_Transport
ACF component ID for [AAX_ITransport](#) components.
- const acfIID IID_IAAXTransportV1
ACF interface ID for [AAX_IACFTransport](#).
- const acfIID IID_IAAXTransportV2
ACF interface ID for [AAX_IACFTransport_V2](#).
- const acfIID IID_IAAXPageTable
ACF component ID for [AAX_IPageTable](#) components.
- const acfIID IID_IAAXPageTableV1
ACF interface ID for [AAX_IACFPageTable](#).
- const acfIID IID_IAAXPageTableV2
ACF interface ID for [AAX_IACFPageTable_V2](#).
- const acfIID IID_AAX_CompID_DescriptionHost
ACF component ID for [AAX_IDescriptionHost](#) components.
- const acfIID IID_IAAXDescriptionHostV1
ACF interface ID for [AAX_IACFDescriptionHost](#).
- const acfIID IID_AAX_CompID_FeatureInfo
ACF component ID for [AAX_IFeatureInfo](#) components.
- const acfIID IID_IAAXFeatureInfoV1
ACF interface ID for [AAX_IACFFeatureInfo](#).

AAX plug-in interface IDs

- const acfIID IID_AAXComplID_EffectParameters
ACF component ID for [AAX_IEffectParameters](#) components.
- const acfIID IID_IAAXEffectParametersV1
ACF interface ID for [AAX_IACFEffectorParameters](#).
- const acfIID IID_IAAXEffectParametersV2
ACF interface ID for [AAX_IACFEffectorParameters_V2](#).
- const acfIID IID_IAAXEffectParametersV3
ACF interface ID for [AAX_IACFEffectorParameters_V3](#).
- const acfIID IID_IAAXEffectParametersV4
ACF interface ID for [AAX_IACFEffectorParameters_V4](#).
- const acfIID IID_AAXComplID_HostProcessor
ACF component ID for [AAX_IHostProcessor](#) components.
- const acfIID IID_IAAXHostProcessorV1
ACF interface ID for [AAX_IACFHostProcessor](#).
- const acfIID IID_IAAXHostProcessorV2
ACF interface ID for [AAX_IACFHostProcessor_V2](#).
- const acfIID IID_AAXComplID_EffectGUI
ACF component ID for [AAX_IEffectGUI](#) components.
- const acfIID IID_IAAXEffectGUIV1
ACF interface ID for [AAX_IACFEffectorGUI](#).
- const acfIID IID_AAXComplID_EffectDirectData
ACF component ID for [AAX_IEffectDirectData](#) components.
- const acfIID IID_IAAXEffectDirectDataV1
ACF interface ID for [AAX_IACFEffectorDirectData](#).

AAX host attributes

- const acfUID AAXATTR_Client_Level
Client application level.

AAX Feature UIDs

- `typedef acfUID AAX_Feature_UID`
- `const AAX_Feature_UID AAXATTR_ClientFeature_StemFormat`
Client stem format feature support.
- `const AAX_Feature_UID AAXATTR_ClientFeature_AuxOutputStem`
Client Auxiliary Output Stem feature support.
- `const AAX_Feature_UID AAXATTR_ClientFeature_SideChainInput`
- `const AAX_Feature_UID AAXATTR_ClientFeature_MIDI`
Client MIDI feature support.

15.150.2 Typedef Documentation

15.150.2.1 `typedef acfUID AAX_Feature_UID`

Identifier for AAX features

See [AAX_IDescriptionHost::AcquireFeatureProperties\(\)](#) and [AAX_IFeatureInfo](#)

15.150.3 Variable Documentation

15.150.3.1 `const acfIID IID_IAAXHostServices`

ACF component ID for [AAX_IHostServices](#) components.

15.150.3.2 `const acfIID IID_IAAXHostServicesV1`

ACF interface ID for [AAX_IACFHostServices](#).

15.150.3.3 `const acfIID IID_IAAXHostServicesV2`

ACF interface ID for [AAX_IACFHostServices_V2](#).

15.150.3.4 `const acfIID IID_IAAXHostServicesV3`

ACF interface ID for [AAX_IACFHostServices_V3](#).

15.150.3.5 `const acfIID IID_IAAXCollection`

ACF component ID for [AAX_ICollection](#) components.

15.150.3.6 `const acfIID IID_IAAXCollectionV1`

ACF interface ID for [AAX_IACFCollection](#).

15.150.3.7 `const acfIID IID_IAAXEffectDescriptor`

ACF component ID for [AAX_IEffectDescriptor](#) components.

15.150.3.8 const acfIID IID_IAAXEffectDescriptorV1

ACF interface ID for [AAX_IACFEffectDescriptor](#).

15.150.3.9 const acfIID IID_IAAXEffectDescriptorV2

ACF interface ID for [AAX_IACFEffectDescriptor_V2](#).

15.150.3.10 const acfIID AAXComplID_AAXComponentDescriptor

ACF component ID for [AAX_IComponentDescriptor](#) components.

15.150.3.11 const acfIID IID_IAAXComponentDescriptorV1

ACF interface ID for [AAX_IACFComponentDescriptor](#).

15.150.3.12 const acfIID IID_IAAXComponentDescriptorV2

ACF interface ID for [AAX_IACFComponentDescriptor_V2](#).

15.150.3.13 const acfIID IID_IAAXComponentDescriptorV3

ACF interface ID for [AAX_IACFComponentDescriptor_V3](#).

15.150.3.14 const acfIID AAXComplID_AAXPropertyMap

ACF component ID for [AAX_IPropertyMap](#) components.

15.150.3.15 const acfIID IID_IAAXPropertyMapV1

ACF interface ID for [AAX_IACFPropertyMap](#).

15.150.3.16 const acfIID IID_IAAXPropertyMapV2

ACF interface ID for [AAX_IACFPropertyMap_V2](#).

15.150.3.17 const acfIID IID_IAAXPropertyMapV3

ACF interface ID for [AAX_IACFPropertyMap_V3](#).

15.150.3.18 const acfIID AAXComplID_HostProcessorDelegate

ACF component ID for [AAX_IHostProcessorDelegate](#) components.

15.150.3.19 const acfIID IID_IAAXHostProcessorDelegateV1

ACF interface ID for [AAX_IACFHostProcessorDelegate](#).

15.150.3.20 const acfIID IID_IAAXHostProcessorDelegateV2

ACF interface ID for [AAX_IACFHostProcessorDelegate_V2](#).

15.150.3.21 const acfIID IID_IAAXHostProcessorDelegateV3

ACF interface ID for [AAX_IACFHostProcessorDelegate_V3](#).

15.150.3.22 const acfIID AAXComplID_AutomationDelegate

ACF component ID for [AAX_IAutomationDelegate](#) components.

15.150.3.23 const acfIID IID_IAAXAutomationDelegateV1

ACF interface ID for [AAX_IACFAutomationDelegate](#).

15.150.3.24 const acfIID AAXComplID_Controller

ACF component ID for [AAX_IController](#) components.

15.150.3.25 const acfIID IID_IAAXControllerV1

ACF interface ID for [AAX_IACFController](#).

15.150.3.26 const acfIID IID_IAAXControllerV2

ACF interface ID for [AAX_IACFController_V2](#).

15.150.3.27 const acfIID IID_IAAXControllerV3

ACF interface ID for [AAX_IACFController_V3](#).

15.150.3.28 const acfIID AAXComplID_PageTableController

ACF component ID for AAX page table controller components.

15.150.3.29 const acfIID IID_IAAXPageTableController

ACF interface ID for [AAX_IACFPageTableController](#).

15.150.3.30 const acfIID IID_IAAXPageTableControllerV2

ACF interface ID for [AAX_IACFPageTableController_V2](#).

15.150.3.31 const acfIID AAXComplID_PrivateDataAccess

ACF component ID for [AAX_IPrivateDataAccess](#) components.

15.150.3.32 const acfIID IID_IAAXPrivateDataAccessV1

ACF interface ID for [AAX_IACFPrivateDataAccess](#).

15.150.3.33 const acfIID AAXComplID_ViewContainer

ACF component ID for [AAX_IViewContainer](#) components.

15.150.3.34 const acfIID IID_IAAXViewContainerV1

ACF interface ID for [AAX_IACFViewContainer](#).

15.150.3.35 const acfIID IID_IAAXViewContainerV2

ACF interface ID for [AAX_IACFViewContainer_V2](#).

15.150.3.36 const acfIID AAXComplID_Transport

ACF component ID for [AAX_ITransport](#) components.

15.150.3.37 const acfIID IID_IAAXTransportV1

ACF interface ID for [AAX_IACFTransport](#).

15.150.3.38 const acfIID IID_IAAXTransportV2

ACF interface ID for [AAX_IACFTransport_V2](#).

15.150.3.39 const acfIID AAXComplID_PageTable

ACF component ID for [AAX_IPageTable](#) components.

15.150.3.40 const acfIID IID_IAAXPageTableV1

ACF interface ID for [AAX_IACFPageTable](#).

15.150.3.41 const acfIID IID_IAAXPageTableV2

ACF interface ID for [AAX_IACFPageTable_V2](#).

15.150.3.42 const acfIID AAX_ComplID_DescriptionHost

ACF component ID for [AAX_IDescriptionHost](#) components.

15.150.3.43 const acfIID IID_IAAXDescriptionHostV1

ACF interface ID for [AAX_IACFDescriptionHost](#).

15.150.3.44 const acfIID AAX_CompID_FeatureInfo

ACF component ID for [AAX_IFeatureInfo](#) components.

15.150.3.45 const acfIID IID_IAAXFeatureInfoV1

ACF interface ID for [AAX_IACFFeatureInfo](#).

15.150.3.46 const acfIID AAXCompID_EffectParameters

ACF component ID for [AAX_IEffectParameters](#) components.

15.150.3.47 const acfIID IID_IAAXEffectParametersV1

ACF interface ID for [AAX_IACFEffectorParameters](#).

15.150.3.48 const acfIID IID_IAAXEffectParametersV2

ACF interface ID for [AAX_IACFEffectorParameters_V2](#).

15.150.3.49 const acfIID IID_IAAXEffectParametersV3

ACF interface ID for [AAX_IACFEffectorParameters_V3](#).

15.150.3.50 const acfIID IID_IAAXEffectParametersV4

ACF interface ID for [AAX_IACFEffectorParameters_V4](#).

15.150.3.51 const acfIID AAXCompID_HostProcessor

ACF component ID for [AAX_IHostProcessor](#) components.

15.150.3.52 const acfIID IID_IAAXHostProcessorV1

ACF interface ID for [AAX_IACFHostProcessor](#).

15.150.3.53 const acfIID IID_IAAXHostProcessorV2

ACF interface ID for [AAX_IACFHostProcessor_V2](#).

15.150.3.54 const acfIID AAXCompID_EffectGUI

ACF component ID for [AAX_IEffectGUI](#) components.

15.150.3.55 const acfIID IID_IAAXEffectGUIV1

ACF interface ID for [AAX_IACFEffectorGUI](#).

15.150.3.56 const acfIID AAXComplID_EffectDirectData

ACF component ID for [AAX_IEffectDirectData](#) components.

15.150.3.57 const acfIID IID_IAAXEffectDirectDataV1

ACF interface ID for [AAX_IACFEfectDirectData](#).

15.150.3.58 AAXATTR_ClientFeature_StemFormat

Client stem format feature support.

To determine the client's support for specific stem formats, use the property map

Property map contents Key: [AAX_EStemFormat](#) values Value: [AAX_ESupportLevel](#) value; if undefined then no information is available

15.150.3.59 AAXATTR_ClientFeature_AuxOutputStem

Client [Auxiliary Output Stem](#) feature support.

Client [Side Chain](#) feature support.

Plug-ins must detect when a host does not support AOS in order to avoid running off the end of the output audio buffer list in the audio algorithm.

[AddAuxOutputStem\(\)](#) will return an error for hosts that do not support this feature, so typically a feature support query using this [AAX_Feature_UID](#) is not required.

15.150.3.60 const AAX_Feature_UID AAXATTR_ClientFeature_SideChainInput**15.150.3.61 AAXATTR_ClientFeature_MIDI**

Client [MIDI](#) feature support.

15.150.3.62 AAXATTR_Client_Level

Client application level.

Type: `uint32_t` (ACFTypeID_UInt32) Value: one of [AAX_EHostLevel](#)

Query using the host's [IACFDefinition](#)

15.151 AAX_UtilsNative.h File Reference

```
#include "AAX_CString.h" #include "AAX_IString.h" #include "AAX_Assert.h" #include "AAX.h" #include <cmath> #include <string.h>
```

15.151.1 Description

Various utility definitions for AAX Native.

Namespaces

- [AAX](#)

Macros

- `#define _AAX_UTILSNATIVE_H_`

Functions

- `double AAX::SafeLog (double aValue)`
Double-precision safe log function. Returns zero for input values that are <= 0.0.
- `float AAX::SafeLogf (float aValue)`
Single-precision safe log function. Returns zero for input values that are <= 0.0.
- `AAX_CBoolean AAX::IsParameterIDEqual (AAX_CParamID iParam1, AAX_CParamID iParam2)`
Helper function to check if two parameter IDs are equivalent.
- `AAX_CBoolean AAX::IsEffectIDEqual (const AAX_IString *iEffectID1, const AAX_IString *iEffectID2)`
Helper function to check if two Effect IDs are equivalent.
- `AAX_CBoolean AAX::IsAvidNotification (AAX_CTypeID inNotificationID)`
Helper function to check if a notification ID is reserved for host notifications.

15.151.2 Macro Definition Documentation

15.151.2.1 `#define _AAX_UTILSNATIVE_H_`

15.152 AAX_VAutomationDelegate.h File Reference

```
#include "AAX_IAutomationDelegate.h"#include "AAX_IACFAutomationDelegate.h" #include "ACFPtr.h"
```

15.152.1 Description

Version-managed concrete AutomationDelegate class.

Classes

- class `AAX_VAutomationDelegate`
Version-managed concrete automation delegate class.

15.153 AAX_VCollection.h File Reference

```
#include "AAX.h"#include "AAX_ICollection.h"#include "AAX_IACFCollection.h" #include "AAX_VDescriptionHost.h" #include "acfunknow.h" #include "ACFPtr.h" #include <set>
```

15.153.1 Description

Version-managed concrete Collection class.

Classes

- class `AAX_VCollection`
Version-managed concrete AAX_ICollection class.

15.154 AAX_VComponentDescriptor.h File Reference

```
#include "AAX_IComponentDescriptor.h"#include "AAX_IDma.h"#include "AAX_I←  
ACFComponentDescriptor.h"#include "acfunknow.h"#include "ACFPtr.h"#include  
<set>
```

15.154.1 Description

Version-managed concrete ComponentDescriptor class.

Classes

- class [AAX_VComponentDescriptor](#)

Version-managed concrete AAX_IComponentDescriptor class.

15.155 AAX_VController.h File Reference

```
#include "AAX_IController.h"#include "AAX_IACFController.h"#include "ACF←  
Ptr.h"
```

15.155.1 Description

Version-managed concrete Controller class.

Classes

- class [AAX_VController](#)

Version-managed concrete Controller class.

15.156 AAX_VDescriptionHost.h File Reference

```
#include "AAX_IDescriptionHost.h"
```

```
#include "ACFPtr.h"
```

Classes

- class [AAX_VDescriptionHost](#)

15.157 AAX_VEffectDescriptor.h File Reference

```
#include "AAX.h"#include "AAX_IEffectDescriptor.h"#include "AAX_IACFEffectDescriptor.h"#include "acfunknown.h"#include "ACFPtr.h"#include <set>#include <map>
```

15.157.1 Description

Version-managed concrete EffectDescriptor class.

Classes

- class [AAX_VEffectDescriptor](#)
Version-managed concrete AAX_IEffectDescriptor class.

15.158 AAX_VENUE_Guide.dox File Reference

15.159 AAX_Version.h File Reference

15.159.1 Description

Version stamp header for the AAX SDK.

This file defines a unique number that can be used to identify the version of the AAX SDK

Macros

- #define [_AAX_VERSION_H_](#)
- #define [AAX_SDK_VERSION](#) (0x0203)
The SDK's version number.
- #define [AAX_SDK_CURRENT_REVISION](#) (13200373)
An atomic revision number for the source included in this SDK.
- #define [AAX_SDK_1p0p1_REVISION](#) (3712639)
- #define [AAX_SDK_1p0p2_REVISION](#) (3780585)
- #define [AAX_SDK_1p0p3_REVISION](#) (3895859)
- #define [AAX_SDK_1p0p4_REVISION](#) (4333589)
- #define [AAX_SDK_1p0p5_REVISION](#) (4598560)
- #define [AAX_SDK_1p0p6_REVISION](#) (5051497)
- #define [AAX_SDK_1p5p0_REVISION](#) (5740047)
- #define [AAX_SDK_2p0b1_REVISION](#) (6169787)
- #define [AAX_SDK_2p0p0_REVISION](#) (6307708)
- #define [AAX_SDK_2p0p1_REVISION](#) (6361692)
- #define [AAX_SDK_2p1p0_REVISION](#) (7820991)

- #define AAX_SDK_2p1p1_REVISION (8086416)
- #define AAX_SDK_2p2p0_REVISION (9967334)
- #define AAX_SDK_2p2p1_REVISION (10693954)
- #define AAX_SDK_2p2p2_REVISION (11819832)
- #define AAX_SDK_2p3p0_REVISION (12546840)
- #define AAX_SDK_2p3p1_REVISION (13200373)

15.159.2 Macro Definition Documentation

15.159.2.1 #define _AAX_VERSION_H_

15.159.2.2 #define AAX_SDK_VERSION (0x0203)

The SDK's version number.

This version number is generally updated only when changes have been made to the AAX binary interface

- The first byte is the major version number
- The second byte is the minor version number

For example:

- SDK 1.0.5 > 0x0100
- SDK 10.2.1 > 0xA02

15.159.2.3 #define AAX_SDK_CURRENT_REVISION (13200373)

An atomic revision number for the source included in this SDK.

15.159.2.4 #define AAX_SDK_1p0p1_REVISION (3712639)

15.159.2.5 #define AAX_SDK_1p0p2_REVISION (3780585)

15.159.2.6 #define AAX_SDK_1p0p3_REVISION (3895859)

15.159.2.7 #define AAX_SDK_1p0p4_REVISION (4333589)

15.159.2.8 #define AAX_SDK_1p0p5_REVISION (4598560)

15.159.2.9 #define AAX_SDK_1p0p6_REVISION (5051497)

15.159.2.10 #define AAX_SDK_1p5p0_REVISION (5740047)

15.159.2.11 #define AAX_SDK_2p0b1_REVISION (6169787)

15.159.2.12 #define AAX_SDK_2p0p0_REVISION (6307708)

15.159.2.13 #define AAX_SDK_2p0p1_REVISION (6361692)

15.159.2.14 #define AAX_SDK_2p1p0_REVISION (7820991)

15.159.2.15 #define AAX_SDK_2p1p1_REVISION (8086416)

15.159.2.16 #define AAX_SDK_2p2p0_REVISION (9967334)

15.159.2.17 #define AAX_SDK_2p2p1_REVISION (10693954)

15.159.2.18 #define AAX_SDK_2p2p2_REVISION (11819832)

15.159.2.19 #define AAX_SDK_2p3p0_REVISION (12546840)

15.159.2.20 #define AAX_SDK_2p3p1_REVISION (13200373)

15.160 AAX_VFeatureInfo.h File Reference

```
#include "AAX_IFeatureInfo.h"#include "ACFPtr.h"#include "acfbasetypes.h"
```

Classes

- class [AAX_VFeatureInfo](#)

15.161 AAX_VHostProcessorDelegate.h File Reference

```
#include "AAX_IHostProcessorDelegate.h"#include "AAX_IACFHostProcessorDelegate.h"#include "ACFPtr.h"
```

15.161.1 Description

Version-managed concrete HostProcessorDelegate class.

Classes

- class [AAX_VHostProcessorDelegate](#)

Version-managed concrete Host Processor delegate class.

15.162 AAX_VHostServices.h File Reference

```
#include "AAX_IHostServices.h"#include "AAX.h"#include "acfunknown.h"#include "ACFPtr.h"#include "AAX_IACFHostServices.h"
```

15.162.1 Description

Version-managed concrete HostServices class.

Classes

- class [AAX_VHostServices](#)

Version-managed concrete AAX_IHostServices class.

15.163 AAX_VPageTable.h File Reference

```
#include "AAX_IPageTable.h"#include "AAX_IACFPageTable.h"#include "ACFPtr.h"
```

Classes

- class [AAX_VPageTable](#)

Version-managed concrete AAX_IPageTable class.

15.164 AAX_VPrivateDataAccess.h File Reference

```
#include "AAX_IPrivateDataAccess.h"#include "AAX_IACFPrivateDataAccess.h"#include "ACFPtr.h"
```

15.164.1 Description

Version-managed concrete PrivateDataAccess class.

Classes

- class [AAX_VPrivateDataAccess](#)

Version-managed concrete AAX_IPrivateDataAccess class.

15.165 AAX_VPropertyMap.h File Reference

```
#include "AAX_IPropertyMap.h"#include "AAX_IACFPropertyMap.h"#include "AAX.h"#include "acfunknow.h"#include "ACFPtr.h"#include <map>
```

15.165.1 Description

Version-managed concrete PropertyMap class.

Classes

- class [AAX_VPropertyMap](#)

Version-managed concrete AAX_IPropertyMap class.

15.166 AAX_VTransport.h File Reference

```
#include "AAX_ITransport.h"#include "AAX_IACFTransport.h"#include "ACFPtr.h"
```

15.166.1 Description

Version-managed concrete Transport class.

Classes

- class [AAX_VTransport](#)
Version-managed concrete AAX_ITransport class.

15.167 AAX_VViewContainer.h File Reference

```
#include "AAX_IVViewContainer.h"#include "AAX_IACFViewContainer.h"#include  
"ACFPtr.h"
```

15.167.1 Description

Version-managed concrete ViewContainer class.

Classes

- class [AAX_VViewContainer](#)
Version-managed concrete AAX_IVViewContainer class.

15.168 DSH_Guide.dox File Reference

15.169 DTT_Guide.dox File Reference

15.170 ReadMe.dox File Reference