## CONTENTS FOR VOLUME 2

### INTERFACE LIST

<table>
<thead>
<tr>
<th>Interface Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>IHXActivePropUser</td>
<td>2</td>
</tr>
<tr>
<td>IHXActivePropUserResponse</td>
<td>4</td>
</tr>
<tr>
<td>IHXActiveRegistry</td>
<td>7</td>
</tr>
<tr>
<td>IHXASMSource</td>
<td>10</td>
</tr>
<tr>
<td>IHXASMSink</td>
<td>14</td>
</tr>
<tr>
<td>IHXAsyncEnumAuthenticationDBResponse</td>
<td>15</td>
</tr>
<tr>
<td>IHXAsyncEnumAuthenticationDBResponse</td>
<td>17</td>
</tr>
<tr>
<td>IHXAsyncIOSelection</td>
<td>19</td>
</tr>
<tr>
<td>IHXAudioCrossFade</td>
<td>21</td>
</tr>
<tr>
<td>IHXAudioDevice</td>
<td>22</td>
</tr>
<tr>
<td>IHXAudioDeviceManager</td>
<td>26</td>
</tr>
<tr>
<td>IHXAudioDeviceResponse</td>
<td>28</td>
</tr>
<tr>
<td>IHXAudioHook</td>
<td>29</td>
</tr>
<tr>
<td>IHXAudioHookManager</td>
<td>31</td>
</tr>
<tr>
<td>IHXAudioPlayer</td>
<td>32</td>
</tr>
<tr>
<td>IHXAudioPlayerResponse</td>
<td>35</td>
</tr>
<tr>
<td>IHXAudioPushdown</td>
<td>36</td>
</tr>
<tr>
<td>IHXAudioStream</td>
<td>37</td>
</tr>
<tr>
<td>IHXAudioStream2</td>
<td>40</td>
</tr>
<tr>
<td>IHXAudioStreamInfoResponse</td>
<td>41</td>
</tr>
<tr>
<td>IHXAuthenticationDBAccess</td>
<td>42</td>
</tr>
<tr>
<td>IHXAuthenticationDBAccessResponse</td>
<td>44</td>
</tr>
<tr>
<td>IHXAuthenticationDBManager</td>
<td>46</td>
</tr>
<tr>
<td>IHXAuthenticationDBManagerResponse</td>
<td>48</td>
</tr>
<tr>
<td>IHXAuthenticationManager</td>
<td>50</td>
</tr>
<tr>
<td>IHXAuthenticationManager2</td>
<td>51</td>
</tr>
<tr>
<td>IHXAuthenticationManagerResponse</td>
<td>52</td>
</tr>
<tr>
<td>IHXAuthenticator</td>
<td>53</td>
</tr>
<tr>
<td>IHXAuthenticatorRequest</td>
<td>54</td>
</tr>
<tr>
<td>IHXAuthenticatorResponse</td>
<td>55</td>
</tr>
<tr>
<td>IHXAuthResponse</td>
<td>56</td>
</tr>
<tr>
<td>IHXAutoConfig</td>
<td>57</td>
</tr>
<tr>
<td>IHXAutoConfigResponse</td>
<td>58</td>
</tr>
<tr>
<td>IHXBackChannel</td>
<td>59</td>
</tr>
<tr>
<td>IHXBroadcastFormatObject</td>
<td>60</td>
</tr>
<tr>
<td>IHXBroadcastMapper</td>
<td>63</td>
</tr>
<tr>
<td>IHXBroadcastMapperResponse</td>
<td>64</td>
</tr>
<tr>
<td>IHXBuffer</td>
<td>65</td>
</tr>
<tr>
<td>IHXCallback</td>
<td>67</td>
</tr>
<tr>
<td>IHXChallenge</td>
<td>68</td>
</tr>
<tr>
<td>IHXChallengeResponse</td>
<td>69</td>
</tr>
<tr>
<td>Interface Name</td>
<td>Page Number</td>
</tr>
<tr>
<td>---------------------------------------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>IHXPacketTimeOffsetHandler</td>
<td>253</td>
</tr>
<tr>
<td>IHXPacketTimeOffsetHandlerResponse</td>
<td>255</td>
</tr>
<tr>
<td>IHXPassiveSiteWatcher</td>
<td>256</td>
</tr>
<tr>
<td>IHXPassword</td>
<td>257</td>
</tr>
<tr>
<td>IHXPendingStatus</td>
<td>260</td>
</tr>
<tr>
<td>IHXPlayer</td>
<td>262</td>
</tr>
<tr>
<td>IHXPlayer2</td>
<td>266</td>
</tr>
<tr>
<td>IHXPlayerCreationSink</td>
<td>268</td>
</tr>
<tr>
<td>IHXPlayerConnectionAdviseSink</td>
<td>269</td>
</tr>
<tr>
<td>IHXPlayerConnectionAdviseSinkManager</td>
<td>272</td>
</tr>
<tr>
<td>IHXPlayerConnectionResponse</td>
<td>273</td>
</tr>
<tr>
<td>IHXPlayerController</td>
<td>275</td>
</tr>
<tr>
<td>IHXPlayerControllerProxyRedirect</td>
<td>278</td>
</tr>
<tr>
<td>IHXPlayerNavigator</td>
<td>279</td>
</tr>
<tr>
<td>IHXPlayerSinkControl</td>
<td>281</td>
</tr>
<tr>
<td>IHXPlugin</td>
<td>282</td>
</tr>
<tr>
<td>IHXPlugin2Handler</td>
<td>284</td>
</tr>
<tr>
<td>IHXPluginEnumerator</td>
<td>290</td>
</tr>
<tr>
<td>IHXPluginFactory</td>
<td>291</td>
</tr>
<tr>
<td>IHXPluginGroupEnumerator</td>
<td>292</td>
</tr>
<tr>
<td>IHXPluginProperties</td>
<td>294</td>
</tr>
<tr>
<td>IHXPluginReloader</td>
<td>295</td>
</tr>
<tr>
<td>IHXPVPDatabase</td>
<td>296</td>
</tr>
<tr>
<td>IHXPVPDBManager</td>
<td>303</td>
</tr>
<tr>
<td>IHXPVPDBManagerResponse</td>
<td>307</td>
</tr>
<tr>
<td>IHXPreferenceEnumerator</td>
<td>309</td>
</tr>
<tr>
<td>IHXPreferences</td>
<td>311</td>
</tr>
<tr>
<td>IHXPreferences2</td>
<td>312</td>
</tr>
<tr>
<td>IHXPrefetch</td>
<td>314</td>
</tr>
<tr>
<td>IHXPrefetchSink</td>
<td>316</td>
</tr>
<tr>
<td>IHXProcess</td>
<td>318</td>
</tr>
<tr>
<td>IHXProcessEntryPoint</td>
<td>319</td>
</tr>
<tr>
<td>IHXPropWatch</td>
<td>320</td>
</tr>
<tr>
<td>IHXPropWatchResponse</td>
<td>323</td>
</tr>
<tr>
<td>IHXRawSinkObject</td>
<td>325</td>
</tr>
<tr>
<td>IHXRawSourceObject</td>
<td>327</td>
</tr>
<tr>
<td>IHXReconfigServerResponse</td>
<td>329</td>
</tr>
<tr>
<td>IHXRecordTimeline</td>
<td>330</td>
</tr>
<tr>
<td>IHXRedirectDBManager</td>
<td>331</td>
</tr>
<tr>
<td>IHXRedirectDBManagerResponse</td>
<td>333</td>
</tr>
<tr>
<td>IHXRegConfig</td>
<td>335</td>
</tr>
<tr>
<td>IHXRegistrationLogger</td>
<td>336</td>
</tr>
<tr>
<td>IHXRegistry</td>
<td>338</td>
</tr>
<tr>
<td>IHXRegistryAltStringHandling</td>
<td>348</td>
</tr>
<tr>
<td>IHXRegistryID</td>
<td>349</td>
</tr>
<tr>
<td>IHXRemoteBroadcastConfiguration</td>
<td>350</td>
</tr>
<tr>
<td>IHXRemoteBroadcastConfigurationResponse</td>
<td>355</td>
</tr>
<tr>
<td>IHXRemoteBroadcastServices</td>
<td>356</td>
</tr>
<tr>
<td>IHXRenderer</td>
<td>358</td>
</tr>
</tbody>
</table>
IHXRequest ................................................................. 363
IHXRequestHandler ...................................................... 365
IHXResolver ................................................................. 366
IHXResolverResponse .................................................... 367
IHRTPPacket ................................................................. 368
IHXScheduler ............................................................... 373
IHXServerAuthConversation ........................................... 375
IHXServerAuthResponse ................................................ 377
IHXServerControl ......................................................... 378
IHXServerControl2 ......................................................... 379
IHXServerFork .............................................................. 380
IHXServerReconfigNotification ....................................... 381
IHXSetSocketOption ....................................................... 382
IHXSite .......................................................... 383
IHXSite2 ............................................................ 387
IHXSiteEnumerator ......................................................... 390
IHXSiteFullScreen ........................................................ 392
IHXSiteManager .......................................................... 394
IHXSiteSupplier ........................................................... 395
IHXSiteUser ............................................................... 397
IHXSiteUserSupplier ...................................................... 399
IHXSiteWatcher .......................................................... 401
IHXSiteWindowed ........................................................ 403
IHXSiteWindowless ....................................................... 405
IHXSLTA ............................................................... 406
IHXStaEvent ............................................................... 409
IHXSourceFinderObject ................................................. 412
IHXSourceFinderResponse .............................................. 413
IHXStatistics .............................................................. 414
IHXStatusMessage ........................................................ 415
IHXStream ............................................................... 416
IHXSyncSource ........................................................... 420
IHXSyncFileManager ..................................................... 422
IHTCPResponse ........................................................... 425
IHTCPSocket ............................................................. 427
IHThreadSafeMethods ................................................... 431
IHTThreadSafeScheduler ............................................ 432
IHTTrack ................................................................. 434
IHTTrackSink .............................................................. 437
IHTTransportControl ..................................................... 440
IHXUDPMulticastInit ..................................................... 441
IHXUDPResponse ........................................................ 442
IHXUDPVisitor ............................................................ 443
IHXUpdateProperties .................................................... 446
IHXUpgradeCollection ................................................ 447
IHXUpgradeHandler ..................................................... 450
IHXUserContext ........................................................... 452
IHXUserImpersonation .................................................. 453
IHXUserProperties ....................................................... 454
B  STRUCTURE LIST

HXAudioData .......................................................... 497
HXAudioFormat ........................................................ 497
HXBitmapInfo ......................................................... 498
HXBitmapInfoHeader ............................................... 500
HXTimeval ............................................................. 502
HXxEvent ............................................................. 502
HXxPoint .............................................................. 503
HXxRect ................................................................ 503
HXxSize ................................................................ 503
HXxWindow ........................................................... 504
PixEffectInfo .......................................................... 505
PixImageInfo .......................................................... 508
PixInitInfo ............................................................ 508
PPVAccessLog ........................................................ 510
PPVPermission ....................................................... 512
PPVRegLog ........................................................... 513

C  FUNCTION LIST

CloseEngine .......................................................... 515
CreateEngine ......................................................... 515
CreateContext ........................................................ 515
CreateLiveText ....................................................... 515
CreateLiveRealPix ................................................... 515
CreatePassword ...................................................... 516
HXCreateInstance ................................................. 516
HXShutdown ........................................................ 516

D  RETURN VALUES

E  REALMEDIA FILE FORMAT (RMFF) REFERENCE

Header Section ....................................................... 528
The following sections document the Helix Client and Server Software Development Kit interfaces.
**IHXActivePropUser**

Purpose: Handles active properties in the registry.

Implemented by: Server plug-ins

Used by: Server core

Header file: *hxmon.h*

You can set an IHXActivePropUser interface as the active user of a property in an IHXActiveRegistry interface. This causes the IHXActivePropUser interface to be consulted every time a plug-in or the server core wants to change a property. The difference between this interface and a property watch is that this interface is asynchronous and can call a “done” method with a failure to cause the property to not be set, and this gets called instead of calling into the IHXRegistry interface. The response interface is IHXActivePropUserResponse.

The IHXActivePropUser interface contains the following methods:

- IHXActivePropUser::DeleteActiveProp
- IHXActivePropUser::SetActiveBuf
- IHXActivePropUser::SetActiveInt
- IHXActivePropUser::SetActiveStr

As with all Component Object Model (COM) interfaces, the IHXActivePropUser interface inherits the following IUnknown methods:

- IUnknown::AddRef
- IUnknown::QueryInterface
- IUnknown::Release

**IHXActivePropUser::DeleteActiveProp**

Provides an asynchronous request to delete the active property.

```cpp
STDMETHOD(DeleteActiveProp) ( 
    THIS_ 
    const char* pName, 
    IHXActivePropUserResponse* pResponse 
) PURE;
```

**pName**
Pointer to the name of the property to delete.

**pResponse**
Pointer to an IHXActivePropUserResponse interface that manages the response to this method.

**IHXActivePropUser::SetActiveBuf**

Provides an asynchronous request to set the property to the specified buffer.
STDMETHOD(SetActiveBuf) (
    THIS_,
    const char* pName,
    IHXBuffer* pBuffer,
    IHXActivePropUserResponse* pResponse
) PURE;

pName
    Pointer to the name of the property.

pBuffer
    Pointer to an IHXBuffer interface that manages the buffer with which the property is set.

pResponse
    Pointer to an IHXActivePropUserResponse interface that manages the response to this method.

IHXActivePropUser::SetActiveInt

Provides an asynchronous request to set the property to the specified integer value.

STDMETHOD(SetActiveInt) (
    THIS_,
    const char* pName,
    UINT32 ul,
    IHXActivePropUserResponse* pResponse
) PURE;

ul
    The integer value to which to set this property.

pResponse
    Pointer to an IHXActivePropUserResponse interface that manages the response to this method.

IHXActivePropUser::SetActiveStr

Provides an asynchronous request to set the property to the specified string.

STDMETHOD(SetActiveStr) (
    THIS_,
    const char* pName,
    IHXBuffer* pBuffer,
    IHXActivePropUserResponse* pResponse
) PURE;

pName
    Pointer to the name of the property.

pBuffer
    Pointer to an IHXBuffer interface that manages the string with which the property is set.

pResponse
    Pointer to an IHXActivePropUserResponse interface that manages the response to this method.
IHXActivePropUserResponse

Purpose: Responds to queries to set properties.
Implemented by: Server core
Used by: Server plug-in
Header file: hxmon.h

This method provides status and information about the active properties requested by the IHXActivePropUser interface.

The IHXActivePropUserResponse interface contains the following methods:

- IHXActivePropUserResponse::DeleteActivePropDone
- IHXActivePropUserResponse::SetActiveBufDone
- IHXActivePropUserResponse::SetActiveIntDone
- IHXActivePropUserResponse::SetActiveStrDone

As with all Component Object Model (COM) interfaces, the IHXActivePropUserResponse interface inherits the following IUnknown methods:

- IUnknown::AddRef
- IUnknown::QueryInterface
- IUnknown::Release

IHXActivePropUserResponse::DeleteActivePropDone

Reports the status of the request to delete the active property.

STDMETHOD(DeleteActivePropDone) (
    THIS_ 
    HX_RESULT res,
    const char* pName,
    IHXBuffer* pInfo[],
    UINT32 ulNumInfo
) PURE;

res
The status of the operation. A value of HXR_OK indicates that the operation was completed successfully.

pName
Pointer to the name of the property.

pInfo
Pointer to an IHXBuffer interface that manages the description string.

ulNumInfo
Indicates the number of descriptions provided in pInfo.

IHXActivePropUserResponse::SetActiveBufDone

Reports the status of the request to set the property to a specified buffer.
STDMETHOD(SetActiveBufDone) (  
    THIS_  
    HX_RESULT res,  
    const char* pName,  
    IHXBuffer* pBuffer,  
    IHXBuffer* pInfo[],  
    UINT32 ulNumInfo  
) PURE;

res  
The status of the operation. A value of HXR_OK indicates that the operation was completed successfully.

pName  
Pointer to the name of the property.

pBuffer  
Pointer to an IHXBuffer interface that manages the buffer to which the property was set.

pInfo  
Pointer to an IHXBuffer interface that manages the description string.

ulNumInfo  
Indicates the number of descriptions provided in pInfo.

IHXActivePropUserResponse::SetActiveIntDone

Report the status of the request to set the property to an integer value.

STDMETHOD(SetActiveIntDone) (  
    THIS_  
    HX_RESULT res,  
    const char* pName,  
    UINT32 ul,  
    IHXBuffer* pInfo[],  
    UINT32 ulNumInfo  
) PURE;

res  
The status of the operation. A value of HXR_OK indicates that the operation was completed successfully.

pName  
Pointer to the name of the property.

ul  
The integer value of the property that was set.

pInfo  
Pointer to an IHXBuffer interface that manages the description string.

ulNumInfo  
Indicates the number of descriptions provided in pInfo.
**IHXActivePropUserResponse::SetActiveStrDone**

Reports the status of the request to set the property to a specified string.

```c
STDMETHOD(SetActiveStrDone) ( 
    THIS_ 
    HX_RESULT res, 
    const char* pName, 
    IHXBuffer* pBuffer, 
    IHXBuffer* pInfo[], 
    UINT32 ulNumInfo 
) PURE;
```

- **res**: The status of the operation. A value of HXR_OK indicates that the operation was completed successfully.

- **pName**: Pointer to the name of the property.

- **pBuffer**: Pointer to an IHXBuffer interface that manages the string to which the property was set.

- **pInfo**: Pointer to an IHXBuffer interface that manages the description string.

- **ulNumInfo**: Indicates the number of descriptions provided in pInfo.
**IHXActiveRegistry**

Purpose: Manages requests for active properties.

Implemented by: Server core

Used by: Server plug-ins

Header file: hxmon.h

This interface manages requests for active properties, that is, properties being watched by an active property user (IHXActivePropUser).

The IHXActiveRegistry interface contains the following methods:

- IHXActiveRegistry::DeleteActiveProp
- IHXActiveRegistry::IsActive
- IHXActiveRegistry::SetActiveBuf
- IHXActiveRegistry::SetActiveInt
- IHXActiveRegistry::SetActiveStr
- IHXActiveRegistry::SetAsActive
- IHXActiveRegistry::SetAsInactive

As with all Component Object Model (COM) interfaces, the IHXActiveRegistry interface inherits the following IUnknown methods:

- IUnknown::AddRef
- IUnknown::QueryInterface
- IUnknown::Release

**IHXActiveRegistry::DeleteActiveProp**

Provides an asynchronous request to delete the active property.

```c
STDMETHOD(DeleteActiveProp) (  
    THIS_  
    const char* pName,  
    IHXActivePropUserResponse* pResponse
) PURE;
```

pName

Pointer to the name of the property to delete.

pResponse

Pointer to an IHXActivePropUserResponse interface that manages the response to this method.

**IHXActiveRegistry::IsActive**

Returns TRUE if a property has an active user that must be queried to change the value. Returns FALSE if the value can only be set.
STDMETHOD_(BOOL, IsActive) (  
    THIS_,  
    const char* pName  
) PURE;

pName  
The property to query.

**IHXActiveRegistry::SetActiveBuf**

Provides an asynchronous request to set a property to a specified buffer.

```cpp
STDMETHOD(SetActiveBuf) (  
    THIS_,  
    const char* pName,  
    IHXBuffer* pBuffer,  
    IHXActivePropUserResponse* pResponse  
) PURE;

pName  
Pointer to the name of the property.

pBuffer  
Pointer to an IHXBuffer interface that manages the buffer with which the property is set.

pResponse  
Pointer to an IHXActivePropUserResponse interface that manages the response to this method.

**IHXActiveRegistry::SetActiveInt**

Provides an asynchronous request to set a property to a specified integer value.

```cpp
STDMETHOD(SetActiveInt) (  
    THIS_,  
    const char* pName,  
    UINT32 ul,  
    IHXActivePropUserResponse* pResponse  
) PURE;

pName  
Pointer to the name of the property.

ul  
The integer value to which to set this property.

pResponse  
Pointer to an IHXActivePropUserResponse interface that manages the response to this method.

**IHXActiveRegistry::SetActiveStr**

Provides an asynchronous request to set a property to a specified string.
STDMETHOD(SetActiveStr) (  
    THIS_,  
    const char* pName,  
    IHXBuffer* pBuffer,  
    IHXActivePropUserResponse* pResponse  
) PURE;

pName  
    Pointer to the name of the property.

pBuffer  
    Pointer to an IHXBuffer interface that manages the string with which the property is set.

pResponse  
    Pointer to an IHXActivePropUserResponse interface that manages the response to this method.

IHXActiveRegistry::SetAsActive

Sets a property to active and registers the active property user.

STDMETHOD(SetAsActive) (  
    THIS_,  
    const char* pName,  
    IHXActivePropUser* pUser  
) PURE;

pName  
    Pointer to the property to set as active.

pUser  
    Pointer to an IHXActivePropUser interface that manages the active property user.

IHXActiveRegistry::SetAsInactive

Removes an active property user from property activation.

STDMETHOD(SetAsInactive) (  
    THIS_,  
    const char* pName,  
    IHXActivePropUser* pUser  
) PURE;

pName  
    Pointer to the property to set as inactive.

pUser  
    Pointer to an IHXActivePropUser interface that manages the active property user to remove.
IHXASMSource

Purpose: Informs a file format plug-in of the Helix client’s Adaptive Stream Management (ASM) rule choices.

Implemented by: File format plug-in

Used by: Helix Universal Server

Header file: hxasm.h

A file format plug-in implements this interface to receive rule subscription and unsubscription information for ASM. Helix Universal Server calls this interface’s methods to transmit the Helix client’s rule subscription choices to the file format plug-in.


The IHXASMSource interface contains the following methods:

- IHXASMSource::Subscribe
- IHXASMSource::Unsubscribe

As with all Component Object Model (COM) interfaces, the IHXASMSource interface inherits the following IUnknown methods:

- IUnknown::AddRef
- IUnknown::QueryInterface
- IUnknown::Release

IHXASMSource::Subscribe

Informs a file format plug-in that a subscription has occurred to the supplied rule number and stream number.

STDMETHOD(Subscribe) (  
    THIS_  
    UINT16 uStreamNumber,  
    UINT16 uRuleNumber  
) PURE;

uStreamNumber
    The stream number to which the subscription applies.

uRuleNumber
    The rule number to which the subscription applies.

IHXASMSource::Unsubscribe

Informs a file format plug-in that a unsubscription has occurred to the supplied rule number and stream number.
STDMETHOD(Unsubscribe) (THIS_,
    UINT16 uStreamNumber,
    UINT16 uRuleNumber
) PURE;

**uStreamNumber**
   The stream number to which the unsubscription applies.

**uRuleNumber**
   The rule number to which the unsubscription applies.
**IHXASMStream**

**Purpose:** Informs Helix Universal Server of Adaptive Stream Management (ASM) rule subscription; adds stream sink objects.

**Implemented by:** Stream object (Helix architecture)

**Used by:** Client and rendering plug-ins

**Header file:** hxasm.h

The Helix Client uses the IHXASMStream::Subscribe and IHXASMStream::Unsubscribe methods to inform Helix Universal Server of its ASM subscription choices. A rendering plug-in can use the IHXASMStream::AddStreamSink method to define itself as a stream sink object that receives notice of rule subscriptions and unsubscriptions through IHXASMStreamSink.

**For More Information:** See “Chapter 11: Adaptive Stream Management” beginning in Volume 1, on page 141.

The IHXASMStream interface contains the following methods:

- IHXASMStream::AddStreamSink
- IHXASMStream::RemoveStreamSink
- IHXASMStream::Subscribe
- IHXASMStream::Unsubscribe

As with all Component Object Model (COM) interfaces, the IHXASMStream interface inherits the following IUnknown methods:

- IUnknown::AddRef
- IUnknown::QueryInterface
- IUnknown::Release

**IHXASMStream::AddStreamSink**

Adds a stream sink to get subscribe and unsubscribe notifications.

```cpp
STDMETHOD(AddStreamSink) (  
    THIS_  
    IHXASMStreamSink* pASMStreamSink  
) PURE;
```

**pASMStreamSink**

Pointer to the IHXASMStreamSink interface that contains the rule subscriptions and unsubscriptions.

**IHXASMStream::RemoveStreamSink**

Removes an advise sink for getting subscribe and unsubscribe notifications.

```cpp
STDMETHOD(RemoveStreamSink) (  
    THIS_  
    IHXASMStreamSink* pASMStreamSink  
) PURE;
```
pASMSStreamSink
Pointer to the IHXASMSStreamSink interface to be removed.

IHXASMSStream::Subscribe
Informs the Helix Universal Server core to subscribe to a particular rule number for this stream.

STDMETHOD(Subscribe) (
    THIS_
    UINT16 uRuleNumber
) PURE;

uRuleNumber
The rule number to which to subscribe.

IHXASMSStream::Unsubscribe
Informs the Helix Universal Server core to unsubscribe to a particular rule number for this stream.

STDMETHOD (Unsubscribe) (  
    THIS_
    UINT16 uRuleNumber
) PURE;

uRuleNumber
The rule number to which to unsubscribe.
IHXASMStreamSink

Purpose: Informs the renderer of Adaptive Stream Management (ASM) rule subscription and unsubscription.

Implemented by: Rendering plug-in

Used by: Helix Client

Header file: hxasm.h

A rendering plug-in implements this interface if it sets itself up as a ASM stream sink object with IHXASMStream. Helix then uses the IHXASMStreamSink::OnSubscribe and IHXASMStreamSink::OnUnsubscribe methods to inform the renderer of stream rule subscription and unsubscription, respectively. This enables the renderer monitor for conditions not known to Helix architecture.


The IHXASMStreamSink interface contains the following methods:

• IHXASMStreamSink::OnSubscribe
• IHXASMStreamSink::OnUnsubscribe

As with all Component Object Model (COM) interfaces, the IHXASMStreamSink interface inherits the following IUnknown methods:

• IUnknown::AddRef
• IUnknown::QueryInterface
• IUnknown::Release

IHXASMStreamSink::OnSubscribe

Informs a renderer that a stream rule subscription has occurred.

STDMETHOD (OnSubscribe) ( 
    THIS_ 
    UINT16 uRuleNumber 
) PURE;

uRuleNumber
The rule number of the subscription.

IHXASMStreamSink::OnUnsubscribe

Informs a renderer that a stream rule unsubscription has occurred.

STDMETHOD (OnUnsubscribe) ( 
    THIS_ 
    UINT16 uRuleNumber 
) PURE;

uRuleNumber
The rule number of the unsubscription.
**IHXAsyncEnumAuthenticationDB**

<table>
<thead>
<tr>
<th>Purpose</th>
<th>Provides enumeration of authentication data.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Implemented by</td>
<td>Database plug-in</td>
</tr>
<tr>
<td>Used by</td>
<td>Server core</td>
</tr>
<tr>
<td>Header file</td>
<td>hxdb.h</td>
</tr>
</tbody>
</table>

This interface enumerates the users that exist in a database. The response interface is `IHXAsyncEnumAuthenticationDBResponse`.

The `IHXAsyncEnumAuthenticationDB` interface contains the following methods:

- `IHXAsyncEnumAuthenticationDB::Clone`
- `IHXAsyncEnumAuthenticationDB::Next`
- `IHXAsyncEnumAuthenticationDB::Reset`
- `IHXAsyncEnumAuthenticationDB::Skip`

As with all Component Object Model (COM) interfaces, the `IHXAsyncEnumAuthenticationDB` interface inherits the following `IUnknown` methods:

- `IUnknown::AddRef`
- `IUnknown::QueryInterface`
- `IUnknown::Release`

**IHXAsyncEnumAuthenticationDB::Clone**

Makes a new enumerator of this collection.

```cpp
STDMETHOD(Clone) ( 
    THIS_ 
    REF(IHXAsyncEnumAuthenticationDB*) pAsyncEnumAuthenticationDBNew 
) PURE;
```

**pAsyncEnumAuthenticationDBNew**

Returns a pointer to an `IHXAsyncEnumAuthenticationDB` interface that manages the new enumerator.

**IHXAsyncEnumAuthenticationDB::Next**

Retrieves the next item in the collection.

```cpp
STDMETHOD(Next) ( 
    THIS_ 
    IHXAsyncEnumAuthenticationDBResponse* pAsyncEnumAuthenticationDBResponseNew 
) PURE;
```

**pAsyncEnumAuthenticationDBResponseNew**

Pointer to an `IHXAsyncEnumAuthenticationDBResponse` interface that manages the response to this method.

**IHXAsyncEnumAuthenticationDB::Reset**

Resets this enumerator to the beginning of the collection.
STDMETHOD(Reset) (  
    THIS_  
    IHXAsyncEnumAuthenticationDBResponse* pAsyncEnumAuthenticationDBResponseNew  
) PURE;

pAsyncEnumAuthenticationDBResponseNew  
    Pointer to an IHXAsyncEnumAuthenticationDBResponse interface that manages the response to this method.

IHXAsyncEnumAuthenticationDB::Skip

Skips the indicated number of items in the collection and then retrieves the next item.

STDMETHOD(Skip) (  
    THIS_  
    IHXAsyncEnumAuthenticationDBResponse* pAsyncEnumAuthenticationDBResponseNew,  
    UINT32 ulNumToSkip  
) PURE;

pAsyncEnumAuthenticationDBResponseNew  
    Pointer to an IHXAsyncEnumAuthenticationDBResponse interface that manages the response to this method.

ulNumToSkip  
    The number of items to skip. This method retrieves the ulNumToSkip + 1 item.
IHXAsyncEnumAuthenticationDBResponse

Purpose: Enumerates authentication data.
Implemented by: Database user
Used by: PPV database plug-in
Header file: hxdb.h

This interface is used in plug-ins for pay-per-view systems, along with all of the other authentication database interfaces. This is the response interface for IHXAsyncEnumAuthenticationDB.

The IHXAsyncEnumAuthenticationDBResponse interface contains the following methods:

- IHXAsyncEnumAuthenticationDBResponse::ResetDone
- IHXAsyncEnumAuthenticationDBResponse::NextDone
- IHXAsyncEnumAuthenticationDBResponse::SkipDone

As with all Component Object Model (COM) interfaces, the IHXAsyncEnumAuthenticationDBResponse interface inherits the following IUnknown methods:

- IUnknown::AddRef
- IUnknown::QueryInterface
- IUnknown::Release

IHXAsyncEnumAuthenticationDBResponse::ResetDone

Reports the status of the call to IHXAsyncEnumAuthenticationDB::Reset.

STDMETHOD(ResetDone) (THIS_ HX_RESULT ResultStatus) PURE;

ResultStatus
The status of the IHXAsyncEnumAuthenticationDB::Reset operation. A value of HXR_OK indicates that the operation has been completed successfully.

IHXAsyncEnumAuthenticationDBResponse::NextDone

Reports the status of the call to IHXAsyncEnumAuthenticationDB::Next. If the call was successful, the principal ID is valid.

STDMETHOD(NextDone) (THIS_ HX_RESULT ResultStatus,
IXHBuffer* pBufferNextPrincipalID) PURE;

ResultStatus
The status of the IHXAsyncEnumAuthenticationDB::Next operation. A value of HXR_OK indicates that the operation was completed successfully.
pBufferNextPrincipalID
   Pointer to an IHXBuffer interface that manages the next item in the collection.

IHXAsyncEnumAuthenticationDBResponse::SkipDone

Reports the status of the call to IHXAsyncEnumAuthenticationDB::Skip. If the call was successful, the principal ID is valid.

    STDMETHOD(SkipDone) (
        THIS_ 
        HX_RESULT ResultStatus, 
        IHXBuffer* pBufferNextPrincipalID 
    ) PURE;

ResultStatus
   The status of the IHXAsyncEnumAuthenticationDB::Skip operation. A value of HXR_OK indicates that the operation was completed successfully.

pBufferNextPrincipalID
   Pointer to an IHXBuffer interface that manages the next item, after the number skipped, in the collection.
IHXAsyncIOSelection

Purpose: Provides callbacks for UNIX input/output (I/O) events handled by `select()`.

Implemented by: Helix architecture

Used by: UNIX plug-in

Header file: `hxengin.h`

This interface is part of Network Services. In UNIX, Helix implements this interface to enable plug-ins to get callbacks based on I/O events that are normally handled by `select()`. Through its two methods, this interface enables plug-ins to set up or cancel callbacks that are executed when a file descriptor is ready for reading or writing, or has an exception.

For More Information: See “Chapter 6: Network Services” beginning in Volume 1, on page 81. See also IHXCallback.

The IHXAsyncIOSelection interface contains the following methods:

- IHXAsyncIOSelection::Add
- IHXAsyncIOSelection::Remove

As with all Component Object Model (COM) interfaces, the IHXAsyncIOSelection interface inherits the following IUnknown methods:

- IUnknown::AddRef
- IUnknown::QueryInterface
- IUnknown::Release

IHXAsyncIOSelection::Add

Adds a callback associated with the given file descriptor when the descriptor is ready for a read or a write, or has an exception. This method is only available in UNIX, and replaces the functionality of `select`.

```c
STDMETHOD(Add) (
    THIS
    IHXCallback* pCallback,
    INT32 lFileDescriptor,
    UINT32 ulType
) PURE;
```

`pCallback`

Pointer to an IHXCallback interface implemented by the user of this method. The IHXCallback::Func method is called when the passed in file descriptor is ready for reading or writing, or has an exception.

`IFileDescriptor`

An input/output (I/O) descriptor on which to perform a select.
ulType
Indicates whether the callback should be used on read, write, or exception. Can be any combination of the following:

- HXAIO_READ
- HXAIO_WRITE
- HXAIO_EXCEPTION

IHXAAsyncIOSelection::Remove
Removes the callback associated with the given file descriptor from the event handler. This method is only available in UNIX, and is intended to replace the functionality of select.

STDMETHOD(Remove) (
    THIS_,
    INT32 lFileDescriptor,
    UINT32 ulType
) PURE;

lFileDescriptor
The input/output (I/O) file descriptor on which the select was performed.

ulType
Indicates whether the callback was used on a read, write, or exception. Can be any combination of the following:

- HXAIO_READ
- HXAIO_WRITE
- HXAIO_EXCEPTION
IHXAudioCrossFade

Purpose: Cross-fades two audio streams.
Implemented by: Helix Architecture audio player
Used by: Audio renderer plug-in
Header file: hxausvc.h

This interface is used to cross-fade across two audio streams. This interface is exposed by IHXAudioPlayer.
The IHXAudioCrossFade interface contains the IHXAudioCrossFade::CrossFade method.
As with all Component Object Model (COM) interfaces, the IHXAudioCrossFade interface inherits the following IUnknown methods:
• IUnknown::AddRef
• IUnknown::QueryInterface
• IUnknown::Release

IHXAudioCrossFade::CrossFade
Cross-fades two audio streams.

STDMETHOD(CrossFade) (  
    THIS_
    IHXAudioStream* pStreamFrom,  
    IHXAudioStream* pStreamTo,  
    UINT32 ulFromCrossFadeStartTime,  
    UINT32 ulToCrossFadeStartTime,  
    UINT32 ulCrossFadeDuration  
) PURE;

pStreamFrom
Point to an IHXAudioStream interface that manages the stream with which a cross-fade is to begin.

pStreamTo
Point to an IHXAudioStream interface that manages the stream to which to be cross-faded.

ulFromCrossFadeStartTime
The “from” stream time when the cross-fade is to be started.

ulToCrossFadeStartTime
The “to” stream time when the cross-fade is to be started.

ulCrossFadeDuration
The time span during which the cross-fade needs to occur.
**IHXAudioDevice**

Purpose: Exports audio data to an audio device.
Implemented by: Audio device object
Used by: Audio Services
Header file: hxausvc.h

This is the interface to the object that exports data to the audio device. It is currently to be used only by Audio Services.

The IRAMAudioDevice interface contains the following methods:
- IHXAudioDevice::CheckFormat
- IHXAudioDevice::Close
- IHXAudioDevice::Drain
- IHXAudioDevice::GetCurrentAudioTime
- IHXAudioDevice::GetVolume
- IHXAudioDevice::InitVolume
- IHXAudioDevice::Open
- IHXAudioDevice::Pause
- IHXAudioDevice::Reset
- IHXAudioDevice::Resume
- IHXAudioDevice::SetVolume
- IHXAudioDevice::Write

As with all Component Object Model (COM) interfaces, the IHXAudioDevice interface inherits the following IUnknown methods:
- IUnknown::AddRef
- IUnknown::QueryInterface
- IUnknown::Release

**IHXAudioDevice::CheckFormat**

Checks the input format against the audio device format.

```c
HRESULT CheckFormat ( 
    const HXAudioFormat* pAudioFormat 
) PURE;
```

*pAudioFormat
Pointer to an HXAudioFormat structure that manages the audio device format.

**IHXAudioDevice::Close**

Closes the audio device.
STDMETHOD(Close) (
    THIS_
    const BOOL bFlush
) PURE;

bFlush
    If TRUE, reset the audio device before closing. If FALSE, drain the audio device before closing.

IHXAUDIODEVICE::Drain
    Plays the buffers already written to the audio device before closing it.
    STDMETHOD(Drain) (    
    THIS
) PURE;

IHXAUDIODEVICE::GetCurrentAudioTime
    Gets the current system audio time.
    STDMETHOD(GetCurrentAudioTime) (    
    THIS_
    REF(ULONG32) ulCurrentTime
) PURE;

ulCurrentTime
    Returns the number of milliseconds that have elapsed since the start of playback.

IHXAUDIODEVICE::GetVolume
    Gets the audio device volume level.
    STDMETHOD_(UINT16,GetVolume) (    
    THIS
) PURE;

IHXAUDIODEVICE::InitVolume
    Informs the audio stream of the client’s volume range. The audio stream maps the client’s volume range into the audio device volume range. Returns TRUE if volume is supported by this audio device.
    STDMETHOD_(BOOL,InitVolume) (    
    THIS_
    const UINT16 uMinVolume,
    const UINT16 uMaxVolume
) PURE;

uMinVolume
    The minimum volume of the client’s volume range.

uMaxVolume
    The maximum volume of the client’s volume range.
IHXAUDIODEVICE::Open

Initializes the audio device and prepares it for writing using the given audio format.

STDMETHOD(Open) (  
    THIS_  
    const HXAudioFormat* pAudioFormat,  
    IHXAUDIODEVICERESPONSE* pStreamResponse  
) PURE;

pAudioFormat
Pointer to an HXAudioFormat structure that contains the audio format.

pStreamResponse
Pointer to an IHXAUDIODEVICERESPONSE interface that manages the response to various methods in this interface.

IHXAUDIODEVICE::Pause

Pauses the audio device.

STDMETHOD(Pause) (  
    THIS  
) PURE;

IHXAUDIODEVICE::Reset

Resets the audio device.

STDMETHOD(Reset) (  
    THIS  
) PURE;

IHXAUDIODEVICE::Resume

Starts or resumes an audio playback.

STDMETHOD(Resume) (  
    THIS  
) PURE;

IHXAUDIODEVICE::SetVolume

Sets the audio device volume level.

STDMETHOD(SetVolume) (  
    THIS_  
    const UINT16 uVolume  
) PURE;

uVolume
The volume level at which the audio device is set.
IHXAudioDevice::Write

Writes to an audio buffer.

STDMETHOD(Write) (  
    THIS_  
    const HXAudioData* pAudioData  
) PURE;

pAudioData
    Pointer to an HXAudioData structure to which to write the audio data.
IHXAudioDeviceManager

Purpose: Replaces the default audio device.
Implemented by: Audio Services
Used by: Audio device object
Header file: hxausvc.h

This interface replaces the audio device and adds an optional final hook. A top-level client might want to replace the default implementation of the audio device provided by the core if the client needs to mix the audio samples with some external audio source or wants to render in some special environment, such as 3D.

The IHXAudioDeviceManager interface contains the following methods:

- IHXAudioDeviceManager::SetFinalHook
- IHXAudioDeviceManager::GetAudioFormat
- IHXAudioDeviceManager::Remove
- IHXAudioDeviceManager::RemoveFinalHook
- IHXAudioDeviceManager::Replace

As with all Component Object Model (COM) interfaces, the IHXAudioDeviceManager interface inherits the following IUnknown methods:

- IUnknown::AddRef
- IUnknown::QueryInterface
- IUnknown::Release

IHXAudioDeviceManager::GetAudioFormat

Returns the audio format in which the audio device is opened. This method fills in the preallocated HXAudioFormat structure passed in to this method.

STDMETHOD(GetAudioFormat) (THIS_ HXAudioFormat* pAudioFormat ) PURE;

pAudioFormat
Pointer to an HXAudioFormat structure that contains the audio format information.

IHXAudioDeviceManager::Remove

Removes the audio device given to the manager in the earlier call to IHXAudioDeviceManager::Replace.

STDMETHOD(Remove) (THIS_ IHXAudioDevice* pAudioDevice ) PURE;

pAudioDevice
Pointer to the IHXAudioDevice interface that manages the audio device to be removed.
**IHXAudioDeviceManager::RemoveFinalHook**

Removes the final hook interface that was instantiated in IHXAudioDeviceManager::SetFinalHook.

```cpp
STDMETHOD(RemoveFinalHook) (
    THIS_
    IHXAudioHook* pHook
) PURE;
```

`pHook`

Pointer to the IHXAudioHook interface that manages the final hook to be removed.

**IHXAudioDeviceManager::Replace**

Replaces the default implementation of the audio device by the given audio device interface.

```cpp
STDMETHOD(Replace) (
    THIS_
    IHXAudioDevice* pAudioDevice
) PURE;
```

`pAudioDevice`

Pointer to an IHXAudioDevice interface that manages the audio device with which to replace the default implementation.

**IHXAudioDeviceManager::SetFinalHook**

Provides one last chance to modify data being written to the audio device. This hook enables the user to change the audio format that is to be written to the audio device. You can do this in the call to IHXAudioHook::OnInit.

```cpp
STDMETHOD(SetFinalHook) (
    THIS_
    IHXAudioHook* pHook
) PURE;
```

`pHook`

Pointer to an IHXAudioHook interface that manages access to the audio buffers.
This is the response interface to IHXAUDIODevice. It is currently implemented only by Audio Services.

The IHXAUDIODeviceResponse interface contains the IHXAUDIODeviceResponse::OnTimeSync method.

As with all Component Object Model (COM) interfaces, the IHXAUDIODeviceResponse interface inherits the following IUnknown methods:

• IUnknown::AddRef
• IUnknown::QueryInterface
• IUnknown::Release

IHXAUDIODeviceResponse::OnTimeSync

Notifies the user of the amount of audio data (in milliseconds) that has been played since the start of the presentation. This method is called by IHXAUDIODevice when audio playback occurs.

STDMETHOD(OnTimeSync) (
    THIS_
    ULONG32 ulTimeEnd
) PURE;

ulTimeEnd
    The time (in milliseconds) corresponding to the amount of data that has been played since the start of the presentation.
IHXAudioHook

Purpose: Provides access to audio buffers.
Implemented by: Audio rendering plug-ins
Used by: Audio Services
Header file: hxausvc.h

An audio rendering plug-in implements this interface to receive pointers to IHXBuffer objects of premixed or postmixed audio data. The plug-in can then create a new buffer of modified data and pass that back to Audio Services for mixing or playing on the output device. The plug-in sets up the premix audio hook through IHXAudioStream and the postmix hook through IHXAudioPlayer.

For More Information: See “Audio Services” in Volume 1, on page 171.

The IHXAudioHook interface contains the following interfaces:

- IHXAudioHook::OnBuffer
- IHXAudioHook::OnInit

As with all Component Object Model (COM) interfaces, the IHXAudioHook interface inherits the following IUnknown methods:

- IUnknown::AddRef
- IUnknown::QueryInterface
- IUnknown::Release

IHXAudioHook::OnBuffer

Intercepts audio data being rendered by audio renderers (who might modify the data, if necessary). Audio Services calls IHXAudioHook::OnBuffer with audio data packets. The renderer should not modify the data in the pData member of the HXAudioData structure pointed to by pAudioInData. If the renderer wants to write a modified version of the data back to Audio Services, then it should create its own IHXBuffer, modify the data, then associate this buffer with the pData member of the HXAudioData structure pointed to by pAudioOutData.

STDMETHOD(OnBuffer) (THIS_
HXAudioData* pAudioInData,
HXAudioData* pAudioOutData
) PURE;
pAudioInData
Pointer to an HXAudioData structure that contains the audio input data.
pAudioOutData
Pointer to an HXAudioData structure that contains the audio output data. This data might be a modified version of the input data.
**IHXAudioHook::OnInit**

Contains the audio data format of the audio data that will be provided in the IHXAudioHook::OnBuffer method.

```cpp
STDMETHOD(OnInit) (
    THIS_ 
    HXAudioFormat* pFormat
) PURE;
```

**pFormat**

Pointer to an HXAudioFormat structure that manages the audio data format.
**IHXAudioHookManager**

Purpose: Adds a hook to the audio device layer.
Implemented by: Client core
Used by: Plug-ins
Header file: hxausvc.h

This interface adds hooks to the audio device. Therefore, if you add this hook, you get access to the data that is about to be written to the audio device.

The IHXAudioHookManager interface contains the following methods:

- IHXAudioHookManager::AddHook
- IHXAudioHookManager::RemoveHook

As with all Component Object Model (COM) interfaces, the IHXAudioHookManager interface inherits the following IUnknown methods:

- IUnknown::AddRef
- IUnknown::QueryInterface
- IUnknown::Release

**IHXAudioHookManager::AddHook**

Adds an audio hook.

```
STDMETHOD(AddHook) (
    THIS_ 
    IHXAudioHook* pHook 
) PURE;
```

**pHook**

Pointer to an IHXAudioHook interface that manages the audio hook.

**IHXAudioHookManager::RemoveHook**

Removes an audio hook.

```
STDMETHOD(RemoveHook) ( 
    THIS_ 
    IHXAudioHook* pHook 
) PURE;
```

**pHook**

Pointer to the IHXAudioHook interface that manages the audio hook to be removed.
An audio rendering plug-in uses this interface to register with Audio Services. The plug-in can then create audio streams, “hook” postmixed audio data for modification by IHXAudioHook, and control volume levels through IHXVolume. After creating audio stream objects, the plug-in can access the objects through IHXAudioStream.

For More Information: See “Audio Services” in Volume 1, on page 171.

The IHXAudioPlayer interface contains the following methods:

- IHXAudioPlayer::AddPostMixHook
- IHXAudioPlayer::CreateAudioStream
- IHXAudioPlayer::GetAudioStream
- IHXAudioPlayer::GetAudioStreamCount
- IHXAudioPlayer::GetAudioVolume
- IHXAudioPlayer::GetDeviceVolume
- IHXAudioPlayer::RemovePostMixHook
- IHXAudioPlayer::RemoveStreamInfoResponse
- IHXAudioPlayer::SetStreamInfoResponse

As with all Component Object Model (COM) interfaces, the IHXAudioPlayer interface inherits the following IUnknown methods:

- IUnknown::AddRef
- IUnknown::QueryInterface
- IUnknown::Release

**IHXAudioPlayer::AddPostMixHook**

Hooks audio data after all the audio streams in this audio player have been mixed.

```c
STDMETHOD(AddPostMixHook) (
    THIS_,
    IHXAudioHook* pHook,
    const BOOL bDisableWrite,
    const BOOL bFinal
) PURE;
```

**pHook**

Pointer to an IHXAudioHook interface that gets the postprocessed audio buffers and their associated audio format.

**bDisableWrite**

If TRUE, you cannot write mixed data.
bFinal
If TRUE, this is the last hook.

**IHXAudioPlayer**:CreateAudioStream

Creates an audio stream.

```cpp
STDMETHOD(CreateAudioStream) ( 
    THIS_,
    IHXAudioStream** pAudioStream
) PURE;
```

*pAudioStream*
Pointer to an IHXAudioStream interface that manages the audio stream information and capabilities.

**IHXAudioPlayer**:GetAudioStream

Gets an audio stream at the given position.

```cpp
STDMETHOD_(IHXAudioStream*,GetAudioStream) ( 
    THIS_,
    UINT16 uIndex
) PURE;
```

*uIndex*
The position of the audio stream.

**IHXAudioPlayer**:GetAudioStreamCount

Gets the number of audio streams currently active in the audio player. Because you can add streams in mid-presentation, this method can return different values on different calls. If the user needs to know about all the streams as they get added to the player, the IHXAudioStreamInfoResponse interface should be implemented and passed in IHXAudioPlayer:SetStreamInfoResponse.

```cpp
STDMETHOD_(UINT16,GetAudioStreamCount) ( 
    THIS
) PURE;
```

**IHXAudioPlayer**:GetAudioVolume

Gets the audio player's volume interface. This volume controls the volume level of all the mixed audio streams for this audio player.

```cpp
STDMETHOD_(IHXVolume*,GetAudioVolume) ( 
    THIS
) PURE;
```

**IHXAudioPlayer**:GetDeviceVolume

Gets the audio device volume interface. This volume controls the audio device volume levels.

```cpp
STDMETHOD_(IHXVolume*,GetDeviceVolume) ( 
    THIS
) PURE;
```
**IHXAudioPlayer::RemovePostMixHook**

Removes an already added post hook.

```cpp
STDMETHOD(RemovePostMixHook) (
    THIS_ IHXAudioHook* pHook
) PURE;
```

pHook
  Pointer to an IHXAudioHook interface that manages the post hook to be removed.

**IHXAudioPlayer::RemoveStreamInfoResponse**

Removes the stream information response interface.

```cpp
STDMETHOD(RemoveStreamInfoResponse) (
    THIS_ IHXAudioStreamInfoResponse* pResponse
) PURE;
```

pResponse
  Pointer to the IHXAudioStreamInfoResponse interface to remove.

**IHXAudioPlayer::SetStreamInfoResponse**

Sets a stream information response interface. A client must have already implemented an IHXAudioStreamInfoResponse interface, and calls this method with the IHXAudioStreamInfoResponse interface as the parameter. The audio player calls IHXAudioStreamInfoResponse::OnStream with the total number of audio streams associated with this audio player.

```cpp
STDMETHOD(SetStreamInfoResponse) (
    THIS_ IHXAudioStreamInfoResponse* pResponse
) PURE;
```

pResponse
  Pointer to an IHXAudioStreamInfoResponse interface that manages the total number of streams associated with this audio player.
IHXAudioPlayerResponse

Purpose: Provides audio playback time synchronization.
Implemented by: Helix Client
Used by: Audio Services
Header file: hxausvc.h

This response interface to IHXAudioPlayer is currently implemented only by the Helix client to receive audio synchronization information. The client passes this synchronization information to rendering plug-ins through IHXRenderer::OnTimeSync.


The IHXAudioPlayerResponse interface contains the IHXAudioPlayerResponse::OnTimeSync method.

As with all Component Object Model (COM) interfaces, the IHXAudioPlayerResponse interface inherits the following IUnknown methods:

• IUnknown::AddRef
• IUnknown::QueryInterface
• IUnknown::Release

IHXAudioPlayerResponse::OnTimeSync

Contains the required synchronization information. This method is called with the current audio playback time.

STDMETHOD(OnTimeSync) (THIS_ ULONG32 ulTimeEnd ) PURE;

ulTimeEnd

The current audio playback time, in milliseconds.
IHXAudioPushdown

Purpose: Sets up the audio pushdown time.
Implemented by: Audio services
Used by: Top-level client
Header file: hxausvc.h

This interface reduces the amount of audio data that is pushed to the audio device before starting playback, which could be used to reduce overall latency in the system. This interface is queried from the IHXClientEngine interface.

The IHXAudioPushdown interface contains the IHXAudioPushdown::SetAudioPushdown method.

As with all Component Object Model (COM) interfaces, the IHXAudioPushdown interface inherits the following IUnknown methods:

- IUnknown::AddRef
- IUnknown::QueryInterface
- IUnknown::Release

IHXAudioPushdown::SetAudioPushdown

Sets the minimum audio pushdown value. This is the amount of audio data being written to the audio device before starting playback.

STDMETHOD(SetAudioPushdown) ( 
    THIS_ 
    UINT32 ulAudioPushdown
) PURE;

ulAudioPushdown
    The minimum audio pushdown, in milliseconds.
IHXAudioStream

Purpose: Provides access to audio stream objects.
Implemented by: Audio stream object (Audio Services)
Used by: Audio rendering plug-ins
Header file: hxausvc.h

An audio rendering plug-in uses this interface to access audio stream objects. These objects contain the audio stream data before Audio Services mixes the audio and sends it to the output device. The IHXAudioStream interface enables the plug-in to play audio, hook audio data for premix processing with IHXAudioHook, receive audio stream information, and get the IHXVolume interface for a stream.

For More Information: See “Audio Services” in Volume 1, on page 171.

The IHXAudioStream interface contains the following methods:

- IHXAudioStream::AddDryNotification
- IHXAudioStream::AddPreMixHook
- IHXAudioStream::GetAudioVolume
- IHXAudioStream::GetStreamInfo
- IHXAudioStream::Init
- IHXAudioStream::RemovePreMixHook
- IHXAudioStream::Write

As with all Component Object Model (COM) interfaces, the IHXAudioStream interface inherits the following IUnknown methods:

- IUnknown::AddRef
- IUnknown::QueryInterface
- IUnknown::Release

IHXAudioStream::AddDryNotification

Adds a notification response interface to get notifications when an audio stream is running dry.

```
STDMETHOD(AddDryNotification) (
    THIS_
    IHXDryNotification* pNotification
) PURE;
```

pNotification
Pointer to an IHXDryNotification interface that manages the notification of when an audio stream is running out of data.

IHXAudioStream::AddPreMixHook

Hooks audio stream data prior to mixing.
STDMETHOD(AddPreMixHook) (  
  THIS_  
  IHXAudioHook* pHook,  
  const BOOL bDisableWrite  
) PURE;

pHook  
Pointer to an IHXAudioHook interface that manages the premixed audio data.

bDisableWrite  
If TRUE, this audio stream data is prevented from mixing with other audio stream data associated with this audio player.

IHXAudioStream::GetAudioVolume  
Gets the audio stream’s volume interface. This volume controls the volume level for this audio stream.

STDMETHOD_(IHXVolume*, GetAudioVolume) (  
  THIS  
) PURE;

IHXAudioStream::GetStreamInfo  
 Gets specific information about this audio stream.

STDMETHOD_(IHXValues*, GetStreamInfo) (  
  THIS  
) PURE;

IHXAudioStream::Init  
Initializes an audio stream with the given audio format.

STDMETHOD(Init) (  
  THIS_  
  const HXAudioFormat* pAudioFormat,  
  IHXValues* pValues  
) PURE;

pAudioFormat  
Pointer to an HXAudioFormat structure that manages the audio format.

pValues  
Pointer to an IHXValues interface that manages the stream identification information.

IHXAudioStream::RemovePreMixHook  
Removes a specified premix hook.

STDMETHOD(RemovePreMixHook) (  
  THIS_  
  IHXAudioHook* pHook  
) PURE;
pHook
Point to the IHXAudioHook interface that manages the premix hook to be removed.

IHXAudioStream::Write

Writes audio data to Audio Services.

```cpp
STDMETHOD(Write) ( 
    THIS_ 
    HXAudioData* pAudioData
) PURE;
```

pAudioData
Pointer to an HXAudioData structure to which to write the audio data.

**Note:** If the renderer loses packets and there is no loss correction, then the renderer should write the next packet using a meaningful start time. Audio Services plays silence where packets are missing.
IHXAUDIOSTREAM2

Purpose: Provides access to audio stream objects.
Implemented by: Audio stream object (Audio Services)
Used by: Audio rendering plug-ins
Header file: hxausvc.h

An audio rendering plug-in uses this interface to access audio stream objects. These objects contain the audio stream data before Audio Services mixes the audio and sends it to the output device. The IHXAUDIOSTREAM2 interface enables you to get the audio format and remove the dry stream notification interface.

The IHXAUDIOSTREAM2 interface contains the following methods:

- IHXAUDIOSTREAM2::GetAudioFormat
- IHXAUDIOSTREAM2::RemoveDryNotification

As with all Component Object Model (COM) interfaces, the IHXAUDIOSTREAM2 interface inherits the following IUnknown methods:

- IUnknown::AddRef
- IUnknown::QueryInterface
- IUnknown::Release

IHXAUDIOSTREAM2::GetAudioFormat

Returns the input audio format of the data written by the renderer. This method fills in the preallocated HXAudioFormat structure passed in.

STDMETHOD(GetAudioFormat) (THIS_ HXAudioFormat* pAudioFormat ) PURE;

pAudioFormat
Pointer to an HXAudioFormat structure that manages the audio format information.

IHXAUDIOSTREAM2::RemoveDryNotification

Removes the notification response interface during the stream switching.

STDMETHOD(RemoveDryNotification) (THIS_ IHXDryNotification* pNotification ) PURE;

pNotification
Pointer to the IHXDryNotification interface to remove.
**IHXAudioStreamInfoResponse**

Purpose: Informs the renderer of the number of audio streams.
Implemented by: Audio rendering plug-ins
Used by: Audio player object (Audio Services)
Header file: hxausvc.h

An audio rendering plug-in can implement this interface to receive notification of the total number of streams associated with an audio player object. After the plug-in registers as the information response object through IHXAudioPlayer, Audio Services calls IHXAudioStreamInfoResponse::OnStream for each audio stream, passing the plug-in a pointer to each stream object.

The IHXAudioStreamInfoResponse interface contains the IHXAudioStreamInfoResponse::OnStream method.

As with all Component Object Model (COM) interfaces, the IHXAudioStreamInfoResponse interface inherits the following IUnknown methods:

- IUnknown::AddRef
- IUnknown::QueryInterface
- IUnknown::Release

**IHXAudioStreamInfoResponse::OnStream**

Gets notification of streams associated with this player. Use IHXAudioPlayer::SetStreamInfoResponse to register your implementation with the Audio Player. After the player has been initialized, it calls IHXAudioStreamInfoResponse::OnStream multiple times to pass all the streams. Because you can add a stream in mid-presentation, an IHXAudioStreamInfoResponse interface should be written to handle IHXAudioStreamInfoResponse::OnStream in the midst of the presentation as well.

```c
STDMETHOD(OnStream) ( 
    THIS_ 
    IHXAudioStream* pAudioStream
) PURE;
```

pAudioStream
Pointer to an IHXAudioStream interface that manages the audio stream associated with this player.
**IHXAuthenticationDBAccess**

**IHXAuthenticationDBAccess::_NewEnum**

Makes a new enumerator of this collection. This enumerator can then be used to find all the principals in a particular database.

```c++
STDMETHOD(_NewEnum) (THIS_ 
    REF(IHXAsyncEnumAuthenticationDB*) pAsyncEnumAuthenticationDBNew
) PURE;
```

*pAsyncEnumAuthenticationDBNew* Returns a pointer to an IHXAsyncEnumAuthenticationDB interface that provides enumeration of authentication data.

**IHXAuthenticationDBAccess::CheckExistence**

Verifies the existence of a principal. This method is obsolete and should not be used in any new applications.

```c++
STDMETHOD(CheckExistence) (THIS_ 
    IHXAuthenticationDBAccessResponse* pAuthenticationDBAccessResponseNew, 
    IHXBuffer* pBufferPrincipalID
) PURE;
```
pAuthenticationDBAccessResponseNew
 Pointer to an IHXAuthenticationDBAccessResponse interface that manages the response to this method.

pBufferPrincipalID
 Pointer to an IHXBuffer interface that manages the identification of the principal to be verified.

IHXAuthenticationDBAccess::GetCredentials

Accesses the credentials for the specified principal. In the simplest use, this means getting the password for a specific user ID. When credentials are encrypted, it is up to the users of this method to do their own decrypting.

STDMETHOD(GetCredentials) ( THIS_
  IHXAuthenticationDBAccessResponse* pAuthenticationDBAccessResponseNew,
  IHXBuffer* pBufferPrincipalID
) PURE;

pAuthenticationDBAccessResponseNew
 Pointer to an IHXAuthenticationDBAccessResponse interface that manages the response to this method.

pBufferPrincipalID
 Pointer to an IHXBuffer interface that manages the identification of the principal.
IHXAuthenticationDBAccessResponse

Purpose: Accesses authentication data.
Implemented by: Authentication plug-in
Used by: Database plug-in
Header file: hxdb.h

All database calls are designed to be asynchronous. This interface provides the asynchronous response to the IHXAuthenticationDBAccess interface.

The IHXAuthenticationDBAccessResponse interface contains the following methods:
- IHXAuthenticationDBAccessResponse::ExistenceCheckDone
- IHXAuthenticationDBAccessResponse::GetCredentialsDone

As with all Component Object Model (COM) interfaces, the IHXAuthenticationDBAccessResponse interface inherits the following IUnknown methods:
- IUnknown::AddRef
- IUnknown::QueryInterface
- IUnknown::Release

IHXAuthenticationDBAccessResponse::ExistenceCheckDone

Responds to a test for the existence of a specified principal. This method is obsolete and should not be used in any new applications.

STDMETHOD(ExistenceCheckDone) ( 
    THIS_ 
    HX_RESULT ResultStatus, 
    IHXBuffer* pBufferPrincipalID 
) PURE;

ResultStatus
The status of the IHXAuthenticationDBAccess::CheckExistence operation. A value of HXR_OK indicates that the operation was completed successfully.

pBufferPrincipalID
Pointer to an IHXBuffer interface that manages the identification of the principal whose existence is being checked.

IHXAuthenticationDBAccessResponse::GetCredentialsDone

Responds to a request for credentials for a specified principal. An authentication plug-in might, for example, compare these credentials from the database to the credentials supplied by the client and either begin to play a clip or send an RTSP header indicating that access is denied.
STDMETHOD(GetCredentialsDone) (THIS_
HX_RESULT ResultStatus,
IHXBuffer* pBufferPrincipalID,
IHXBuffer* pBufferCredentials
) PURE;

ResultStatus
The status of the IHXAuthenticationDBAccess::GetCredentials operation. A value of HXR_OK indicates
that the operation was completed successfully.

pBufferPrincipalID
Pointer to an IHXBuffer interface that manages the identification of the principal.

pBufferCredentials
Pointer to an IHXBuffer interface that manages the credentials.
IHXAuthenticationDBManager

Purpose: Provides storage for authentication data.
Implemented by: Database plug-in
Used by: File system plug-in
Header file: hxdb.h

This interface is used by a file system plug-in to add users to a database, typically to populate an authentication realm. The response interface is IHXAuthenticationDBManagerResponse.

The IHXAuthenticationDBManager interface contains the following methods:

• IHXAuthenticationDBManager::AddPrincipal
• IHXAuthenticationDBManager::RemovePrincipal
• IHXAuthenticationDBManager::SetCredentials

As with all Component Object Model (COM) interfaces, the IHXAuthenticationDBManager interface inherits the following IUnknown methods:

• IUnknown::AddRef
• IUnknown::QueryInterface
• IUnknown::Release

IHXAuthenticationDBManager::AddPrincipal

Adds the specified user to the database, if the user is not already there.

STDMETHOD(AddPrincipal) (  
    THIS_  
    IHXAuthenticationDBManagerResponse* pAuthenticationDBManagerResponseNew,  
    IHXBuffer* pBufferPrincipalID  
) PURE;

pAuthenticationDBManagerResponseNew
    Pointer to an IHXAuthenticationDBManagerResponse interface that manages the response to this method.

pBufferPrincipalID
    Pointer to an IHXBuffer interface that manages the identification of the user to be added to the database.

IHXAuthenticationDBManager::RemovePrincipal

Removes the specified user from the database, if the user is there.

STDMETHOD(RemovePrincipal) (  
    THIS_  
    IHXAuthenticationDBManagerResponse* pAuthenticationDBManagerResponseNew,  
    IHXBuffer* pBufferPrincipalID  
) PURE;
pAuthenticationDBManagerResponseNew
   Pointer to an IHXAuthenticationDBManagerResponse interface that manages the response to this method.

pBufferPrincipalID
   Pointer to an IHXBuffer interface that manages the identification of the user to remove from the database.

IHXAuthenticationDBManager::SetCredentials

Replaces the credentials for the specified user. Usually the credentials are a password (which might be encrypted). It is not the database's job to protect this data; when necessary, authentication plug-ins should protect this data before storing it.

STDMETHOD(SetCredentials) (
   THIS,
   IHXAuthenticationDBManagerResponse* pAuthenticationDBManagerResponseNew,
   IHXBuffer* pBufferPrincipalID,
   IHXBuffer* pBufferCredentials
) PURE;

pAuthenticationDBManagerResponseNew
   Pointer to an IHXAuthenticationDBManagerResponse interface that manages the response to this method.

pBufferPrincipalID
   Pointer to an IHXBuffer interface that manages the identification of the user.

pBufferCredentials
   Pointer to an IHXBuffer interface that manages the credentials to be replaced.
IHXAuthenticationDBManagerResponse

Purpose: Manages storage for authentication data.
Implemented by: File system plug-in
Used by: Database plug-in
Header file: hxdb.h

This interface provides the asynchronous response to IHXAuthenticationDBManager. For example, a module might call a database plug-in's IHXAuthenticationDBManager::AddPrincipal method to add a user to the database, then in IHXAuthenticationDBManagerResponse::AddPrincipalDone, the module might send a response to the browser indicating that the user was successfully added.

The IHXAuthenticationDBManagerResponse interface contains the following methods:

- IHXAuthenticationDBManagerResponse::AddPrincipalDone
- IHXAuthenticationDBManagerResponse::RemovePrincipalDone
- IHXAuthenticationDBManagerResponse::SetCredentialsDone

As with all Component Object Model (COM) interfaces, the IHXAuthenticationDBManagerResponse interface inherits the following IUnknown methods:

- IUnknown::AddRef
- IUnknown::QueryInterface
- IUnknown::Release

IHXAuthenticationDBManagerResponse::AddPrincipalDone

Responds to a request to add a user to the database.

STDMETHOD(AddPrincipalDone) (  
THIS_  
HX_RESULT ResultStatus,  
IHXBuffer* pBufferPrincipalID  
) PURE;

ResultStatus
The status of the IHXAuthenticationDBManager::AddPrincipal operation. A value of HXR_OK indicates that the operation was completed successfully.

pBufferPrincipalID
Pointer to an IHXBuffer interface that manages the identification of the user that was added to the database.

IHXAuthenticationDBManagerResponse::RemovePrincipalDone

Responds to a request to remove a user from the database.

STDMETHOD(RemovePrincipalDone) (  
THIS_  
HX_RESULT ResultStatus,  
IHXBuffer* pBufferPrincipalID  
) PURE;
**ResultStatus**

The status of the `IHXAuthenticationDBManager::RemovePrincipal` operation. A value of `HXR_OK` indicates that the operation was completed successfully.

**pBufferPrincipalID**

Pointer to an `IHXBuffer` interface that manages the identification of the user that was removed from the database.

**IHXAuthenticationDBManagerResponse::SetCredentialsDone**

Responds to a request to set the user’s credentials.

```c
HRESULT SetCredentialsDone(IHXBuffer* pBufferPrincipalID);
```

**ResultStatus**

The status of the `IHXAuthenticationDBManager::SetCredentials` operation. A value of `HXR_OK` indicates that the operation was completed successfully.

**pBufferPrincipalID**

Pointer to an `IHXBuffer` interface that manages the identification of the user whose credentials were replaced.
IHXAuthenticationManager

Purpose: Directs the top-level client to get the user name and password.
Implemented by: Top-level client
Used by: Client core
Header file: hxauth.h

When authentication is required for a requested file, the Helix client core calls IHXAuthenticationManager::HandleAuthenticationRequest to instruct the top-level client to gather the user name and password (or other authentication parameters). The top-level client typically receives these from the user through a pop-up dialog box. The response interface is IHXAuthenticationManagerResponse.


The IHXAuthenticationManager interface contains the IHXAuthenticationManager::HandleAuthenticationRequest method.

As with all Component Object Model (COM) interfaces, the IHXAuthenticationManager interface inherits the following IUnknown methods:

- IUnknown::AddRef
- IUnknown::QueryInterface
- IUnknown::Release

IHXAuthenticationManager::HandleAuthenticationRequest

GETS a user name and password, or other authentication parameters.

STDMETHOD(HandleAuthenticationRequest) (  
    THIS_  
    IHXAuthenticationManagerResponse* pResponse  
) PURE;

pResponse
    Pointer to an IHXAuthenticationManagerResponse interface that manages the response to the IHXAuthenticationManager interface.
IHXAuthenticationManager2

Purpose: Provides an additional means of authenticating users.
Implemented by: Top-level client
Used by: Client core
Header file: hxauth.h

When authentication is required for a requested file, the Helix client core calls IHXAuthenticationManager2::HandleAuthenticationRequest2 to instruct the top-level client to gather the user name and password (or other authentication parameters). The top-level client typically receives these from the user through a pop-up dialog box. The response interface is IHXAuthenticationManagerResponse.


The IHXAuthenticationManager2 interface contains the IHXAuthenticationManager2::HandleAuthenticationRequest2 method.

As with all Component Object Model (COM) interfaces, the IHXAuthenticationManager2 interface inherits the following IUnknown methods:
- IUnknown::AddRef
- IUnknown::QueryInterface
- IUnknown::Release

IHXAuthenticationManager2::HandleAuthenticationRequest2

Gets a user name and password, or other authentication parameters. This method also includes an IHXValues list for sending information to the authentication manager, for example for support of proxy authentication which might include a “pseudonym” header or other information.

STDMETHOD(HandleAuthenticationRequest2) (  
THIS_  
IHXAuthenticationManagerResponse* pResponse,  
IHXValues* pHeader  
) PURE;

pResponse
Pointer to an IHXAuthenticationManagerResponse interface that manages the response to this interface.

pHeader
Pointer to an IHXValues interface that manages any additional information.
IHXAuthenticationManagerResponse

Purpose: Passes the user name and password to the client core.
Implemented by: Client core
Used by: Top-level client
Header file: hxauth.h

After gathering authentication parameters (user name and password), the client calls
IHXAuthenticationManagerResponse::AuthenticationRequestDone to pass the parameters to the client core.
The client core then transports the parameters to Helix Universal Server.


The IHXAuthenticationManagerResponse interface contains the
IHXAuthenticationManagerResponse::AuthenticationRequestDone method.

As with all Component Object Model (COM) interfaces, the IHXAuthenticationManagerResponse interface inherits the following IUnknown methods:

- IUnknown::AddRef
- IUnknown::QueryInterface
- IUnknown::Release

IHXAuthenticationManagerResponse::AuthenticationRequestDone

Contains the user name and password requested by the
IHXAuthenticationManager::HandleAuthenticationRequest method.

STDMETHOD(AuthenticationRequestDone) (  
    THIS_  
    HX_RESULT result,  
    const char* pUserName,  
    const char* pPassword  
) PURE;

result
The result of the authentication request. A value of HXR_OK indicates that the operation was completed successfully.

pUserName
Pointer to the user name.

pPassword
Pointer to the password.
**IHXAuthenticator**

This interface is obsolete and should not be used in any new programming.

**IHXAuthenticator::Authenticate**

This method is obsolete and should not be used in any new programming.

**IHXAuthenticator::AuthValuesReady**

This method is obsolete and should not be used in any new programming.

**IHXAuthenticator::InitAuthenticator**

This method is obsolete and should not be used in any new programming.

**IHXAuthenticator::GenerateAuthRequest**

This method is obsolete and should not be used in any new programming.
**IHXAuthenticatorRequest**

This interface is obsolete and should not be used in any new programming.

**IHXAuthenticatorRequest::GetAuthValues**

This method is obsolete and should not be used in any new programming.
IHXAuthenticatorResponse

This interface is obsolete and should not be used in any new programming.

IHXAuthenticatorResponse::AuthenticateDone

This method is obsolete and should not be used in any new programming.
IHXAuthResponse

Purpose: Provides authentication status.
Implemented by: Remote broadcast services users
Used by: Remote broadcast services
Header file: hxbrcst.h

This interface indicates to the calling IHXRemoteBroadcastServices::InitRemoteBroadcast—Advanced method that the authorization process for initializing remote broadcast services has completed, and returns the status of the authorization process to the calling method.

The IHXAuthResponse interface contains the IHXAuthResponse::AuthDone method.

As with all Component Object Model (COM) interfaces, the IHXAuthResponse interface inherits the following IUnknown methods:
• IUnknown::AddRef
• IUnknown::QueryInterface
• IUnknown::Release

IHXAuthResponse::AuthDone

Indicates the authorization process has completed and returns the status of the authorization.

STDMETHOD(AuthDone) ( 
    THIS_ 
    BOOL bAuthSucceeded 
) PURE;

bAuthSucceeded
    If TRUE, authorization succeeded. If FALSE, authorization failed.
IHXAutoConfig

This interface is obsolete and should not be used in any new programming.

IHXAutoConfig::Abort

This method is obsolete and should not be used in any new programming.

IHXAutoConfig::Close

This method is obsolete and should not be used in any new programming.

IHXAutoConfig::DoAutoConfig

This method is obsolete and should not be used in any new programming.

IHXAutoConfig::Init

This method is obsolete and should not be used in any new programming.
IHXAutoConfigResponse

This interface is obsolete and should not be used in any new programming.

IHXAutoConfigResponse::OnBegin

This method is obsolete and should not be used in any new programming.

IHXAutoConfigResponse::OnComplete

This method is obsolete and should not be used in any new programming.

IHXAutoConfigResponse::OnProgress

This method is obsolete and should not be used in any new programming.
If a file format plug-in implements this interface, its corresponding rendering plug-in can send it data in an IHXPacket object.

The IHXBackChannel interface contains the IHXBackChannel::PacketReady method.

As with all Component Object Model (COM) interfaces, the IHXBackChannel interface inherits the following IUnknown methods:
- IUnknown::AddRef
- IUnknown::QueryInterface
- IUnknown::Release

## IHXBackChannel::PacketReady

Contains a back channel packet sent from the rendering plug-in to the file format plug-in.

```cpp
STD_METHOD(PacketReady) (  
    THIS_  
    IHXPacket* pPacket  
) PURE;

pPacket
    Pointer to an IHXPacket interface that manages data from the rendering plug-in.
```
IHXBroadcastFormatObject

Purpose: Provides methods of communicating to broadcast plug-ins.
Implemented by: Broadcast plug-ins
Used by: Helix Universal Server
Header file: hxformt.h

This is the main interface Helix Universal Server uses to communicate with a broadcast plug-in. At system startup, Helix Universal Server uses this interface to get functional information about each broadcast plug-in. With this information and the requested URL’s mount point, it determines which broadcast plug-in to use to stream a requested live feed. Additional methods instruct the plug-in to send stream header information or stream packets. The response interface is IHXFormatResponse.

For More Information: See “Creating a Monitor Plug-in” in Volume 1, on page 124. See also the information on the FS Mount parameter in Helix Universal Server Configuration File Reference.

The IHXBroadcastFormatObject interface contains the following methods:

- IHXBroadcastFormatObject::GetBroadcastFormatInfo
- IHXBroadcastFormatObject::GetFileHeader
- IHXBroadcastFormatObject::GetStreamHeader
- IHXBroadcastFormatObject::InitBroadcastFormat
- IHXBroadcastFormatObject::StartPackets
- IHXBroadcastFormatObject::StopPackets

As with all Component Object Model (COM) interfaces, the IHXBroadcastFormatObject interface inherits the following IUnknown methods:

- IUnknown::AddRef
- IUnknown::QueryInterface
- IUnknown::Release

IHXBroadcastFormatObject::GetBroadcastFormatInfo

Retrieves a string that uniquely identifies this broadcast format.

STDMETHOD(GetBroadcastFormatInfo) (const char* pToken) PURE;

pToken

Returns a pointer to the string that identifies this broadcast format.

IHXBroadcastFormatObject::GetFileHeader

Gets the file header.
STDMETHOD(GetFileHeader) (THIS) PURE;

IHXBroadcastFormatObject::GetStreamHeader

Gets the file header. When the header is ready, this method should call IHXFormatResponse::StreamHeaderReady to pass the file header back to the server.

STDMETHOD(GetStreamHeader) (THIS UINT16 unStreamNumber) PURE;

unStreamNumber
The stream number from which to fill in the file header information.

IHXBroadcastFormatObject::InitBroadcastFormat

Initializes the broadcast format response for the supplied URL.

STDMETHOD(InitBroadcastFormat) (THIS const char* pURL, IHXFormatResponse* pFormatResponse) PURE;

pURL
Pointer to the URL to which to stream the data.

pFormatResponse
Pointer to an IHXFormatResponse interface that manages the response to the IHXBroadcastFormatObject methods.

IHXBroadcastFormatObject::StartPackets

Starts sending packets for a particular stream in the file. When the packet is available, the broadcast plug-in should call IHXFormatResponse::PacketReady to pass the packet back to the server.

STDMETHOD(StartPackets) (THIS UINT16 unStreamNumber) PURE;

unStreamNumber
The stream number from which to send the packet.

IHXBroadcastFormatObject::StopPackets

Stops sending packets for a particular stream in the file.
STDMETHOD(StopPackets) ( 
    THIS_,
    UINT16 unStreamNumber
) PURE;

unStreamNumber
    The stream number from which to stop the packets.
**IHXBroadcastMapper**

Purpose: Determines the format of a live broadcast.

Implemented by: Broadcast plug-ins

Used by: Helix Universal Server

Header file: hxfiles.h

A broadcast plug-in implements this interface rather than IHXFileMimeMapper. When a client connects to a live broadcast, the broadcast plug-in creates a file object that Helix Universal Server uses to verify the connection. After verifying that the requested URL is for an existing broadcast, Helix Universal Server calls the file object’s IHXBroadcastMapper::FindBroadcastType method to receive the short name of the broadcast plug-in to use. The response interface is IHXBroadcastMapperResponse.

For More Information: See “Chapter 8: Live Broadcasting” beginning in Volume 1, on page 91.

The IHXBroadcastMapper interface contains the IHXBroadcastMapper::FindBroadcastType method.

As with all Component Object Model (COM) interfaces, the IHXBroadcastMapper interface inherits the following IUnknown methods:

- IUnknown::AddRef
- IUnknown::QueryInterface
- IUnknown::Release

**IHBroadcastMapper::FindBroadcastType**

Returns the short name of the broadcast plug-in’s name to the broadcast mapper response interface.

```cpp
STDMETHOD(FindBroadcastType) (
    THIS_ 
    const char* pURL, 
    IHXBroadcastMapperResponse* pBroadcastMapperResponse)

pURL
    Pointer to the URL name.

pBroadcastMapperResponse
    Pointer to an IHXBroadcastMapperResponse interface that manages the response to this method.
```
A broadcast file object uses this response interface to \texttt{IHXBroadcastMapper} to notify Helix Universal Server of the short name of the broadcast plug-in to use for the requested live broadcast. The plug-in uses this interface rather than \texttt{IHXFileMimeMapperResponse}, which is used by standard file system plugins.


The \texttt{IHXBroadcastMapperResponse} interface contains the \texttt{IHXBroadcastMapperResponse::BroadcastTypeFound} method.

As with all Component Object Model (COM) interfaces, the \texttt{IHXBroadcastMapperResponse} interface inherits the following \texttt{IUnknown} methods:

- \texttt{IUnknown::AddRef}
- \texttt{IUnknown::QueryInterface}
- \texttt{IUnknown::Release}

\textbf{IHXBroadcastMapperResponse::BroadcastTypeFound}

Indicates the short name of the broadcast plug-in. This method is called by the file object when the initialization of the file is complete, and the broadcast type is available for the request file.

\begin{verbatim}
STDMETHOD(BroadcastTypeFound) (
    THIS_
    HX_RESULT status,
    const char* pBroadcastType
) PURE;
\end{verbatim}

\texttt{status}

Contains the response status. If this parameter is HXR\_OK, the call to \texttt{IHXBroadcastMapper::FindBroadcastType} completed successfully. If the file is not valid for the file system, this parameter returns HXR\_FAILED and \texttt{pBroadcastType} is set to NULL.

\texttt{pBroadcastType}

Pointer to the short name of the broadcast plug-in.
IHXBuffer

Purpose: Provides general-purpose buffers for passing data.  
Implemented by: Buffer object (Helix architecture) 
Used by: Any component  
Header file: ihxpckts.h 

This interface enables Helix objects to use variable-length data buffers managed through Component Object Model (COM) reference counting. Typically, Helix objects use these buffer objects to pass data. The preferred implementation is to use IHXCommonClassFactory to create the buffer objects.

For More Information: See IHXValues in Volume 2, on page 455. See also “Using IHXBuffer to Create Data Buffers” in Volume 1, on page 31.

The IHXBuffer interface contains the following methods:

- IHXBuffer::Get
- IHXBuffer::GetBuffer
- IHXBuffer::GetSize
- IHXBuffer::Set
- IHXBuffer::SetSize

As with all Component Object Model (COM) interfaces, the IHXBuffer interface inherits the following IUnknown methods:

- IUnknown::AddRef
- IUnknown::QueryInterface
- IUnknown::Release

IHXBuffer::Get

Retrieves the buffer data.

STDMETHOD(Get) ( 
    THIS_ 
    REF(UCHAR*) pData, 
    REF(ULONG32) ulLength 
) PURE;

pData
    Returns a pointer to the buffer data.

ulLength
    Returns the length of the buffer data.

IHXBuffer::GetBuffer

Obtains a pointer to the buffer.
STDMETHOD_(UCHAR*, GetBuffer) (THIS) PURE;

IHXBuffer::GetSize
Retrieves the buffer size.
STDMETHOD_(ULONG32, GetSize) (THIS) PURE;

IHXBuffer::Set
Writes data to the buffer.
STDMETHOD(Set) (THIS,
   const UCHAR* pData,
   ULONG32 ulLength
) PURE;

pData
A pointer to the data written to the buffer.
ulLength
The length of the buffer.

IHXBuffer::SetSize
Sets the buffer size.
STDMETHOD(SetSize) (THIS,
   ULONG32 ulLength
) PURE;

ulLength
The size of the buffer.
Monitor plug-ins can use callbacks, for example, to get information about the Helix Universal Server registry properties at regular intervals. Any component can receive a callback from the system scheduler through IHXCallback.

For More Information: See IHXScheduler in Volume 2, on page 373. See also “Creating a Monitor Plug-in” in Volume 1, on page 124.

The IHXCallback interface contains the IHXCallback::Func method.

As with all Component Object Model (COM) interfaces, the IHXCallback interface inherits the following IUnknown methods:

- IUnknown::AddRef
- IUnknown::QueryInterface
- IUnknown::Release

IHXCallback::Func

Indicates that the designated callback time has been reached.

STDMETHOD(Func) ( 
    THIS 
) PURE;
IHXChallenge

Purpose: Retrieves additional information from the client without further authorization.

Implemented by: Server core

Used by: Server plug-in

Header file: hxauthn.h

This method provides a means of exchanging additional information with a client without creating a new request. The information is managed by an IHXRequest interface. If the information is absent, the protocol that this request was made on does not support multi-message authentication (such as PNA). The response interface is IHXChallengeResponse.

The IHXChallenge interface contains the IHXChallenge::SendChallenge method.

As with all Component Object Model (COM) interfaces, the IHXChallenge interface inherits the following IUnknown methods:

- IUnknown::AddRef
- IUnknown::QueryInterface
- IUnknown::Release

IHXChallenge::SendChallenge

Requests additional information from the client.

STDMETHOD(SendChallenge) (THIS_
   IHXChallengeResponse* pChallengeResponseSender,
   IHXRequest* pRequestChallenge
) PURE;

pChallengeResponseSender
  Pointer to an IHXChallengeResponse interface that manages the response to this method.

pRequestChallenge
  Pointer to an IHXRequest interface that manages the additional information. The IHXRequest interface pointed to by this parameter should be the same as that which was passed to the IHXServerAuthConversation::MakeChallenge method. It should contain CString values for each MIME header that is sent to the client.
**IHXChallengeResponse**

Purpose: Receives additional information from the client.

Implemented by: Server authenticator

Used by: Server core

Header file: hxauthn.h

This interface provides the results of the authentication procedure.

The IHXChallengeResponse interface contains the IHXChallengeResponse::ResponseReady method.

As with all Component Object Model (COM) interfaces, the IHXChallengeResponse interface inherits the following IUnknown methods:

- IUnknown::AddRef
- IUnknown::QueryInterface
- IUnknown::Release

**IHXChallengeResponse::ResponseReady**

Returns the additional information requested from the challenge interface.

```
STDMETHOD(ResponseReady) ( 
    THIS_
    IHXRequest* pRequestResponse
) PURE;
```

**pRequestResponse**

Pointer to an IHXRequest interface that manages the additional information. This request interface should be the same as that passed to the IHXChallenge::SendChallenge and IHXServerAuthConversation::MakeChallenge methods.
IHXClientAdviseSink

Purpose: Notifies the top-level client of the status of a presentation.
Implemented by: Top-level client
Used by: Player object (Helix client core)
Header file: hxclsnk.h

The top-level client implements this interface to receive notifications from the client core about changes in a presentation’s playback status. The top-level client can thereby learn when the client core is buffering a presentation, contacting a host, pausing playback, and so on. The top-level client registers as an advise sink for a specific player object through that object’s IHXPlayer::AddAdviseSink method.


The IHXClientAdviseSink interface contains the following methods:

- IHXClientAdviseSink::OnBegin
- IHXClientAdviseSink::OnBuffering
- IHXClientAdviseSink::OnContacting
- IHXClientAdviseSink::OnPause
- IHXClientAdviseSink::OnPosLength
- IHXClientAdviseSink::OnPostSeek
- IHXClientAdviseSink::OnPreSeek
- IHXClientAdviseSink::OnPresentationClosed
- IHXClientAdviseSink::OnPresentationOpened
- IHXClientAdviseSink::OnStatisticsChanged
- IHXClientAdviseSink::OnStop

As with all Component Object Model (COM) interfaces, the IHXClientAdviseSink interface inherits the following IUnknown methods:

- IUnknown::AddRef
- IUnknown::QueryInterface
- IUnknown::Release

IHXClientAdviseSink::OnBegin

Informs the client that a begin or resume has just occurred. The top-level client is informed of the first time in the stream’s time line after the resume.

STDMETHOD(OnBegin) (THIS_ULONGLONG ulTime)

IHXClientAdviseSink::OnBegin
ulTime
The time in the stream’s time line from which to begin or resume.

IHXClientAdviseSink::OnBuffering

Informs the client that buffering of data is occurring. The top-level client is informed of the reason for the buffering (startup of stream, seek has occurred, network congestion, and so on), as well as the percentage of the buffering process that has been completed.

STDMETHOD(OnBuffering) ( 
    THIS_ 
    ULONG32 ulFlags, 
    UINT16 unPercentComplete 
) PURE;

ulFlags
The reason for the buffering. Can be any of the following:

- BUFFERING_START_UP
- BUFFERING_SEEK
- BUFFERING_CONGESTION
- BUFFERING_LIVE_PAUSE

unPercentComplete
The percentage of the buffering process that has been completed.

IHXClientAdviseSink::OnContacting

Informs the client that a host (or hosts) is being contacted.

STDMETHOD(OnContacting) ( 
    THIS_ 
    const char* pHostName 
) PURE;

pHostName
Pointer to the name of the host being contacted.

IHXClientAdviseSink::OnPause

Informs the client that a pause has just occurred. The top-level client is informed of the last time in the stream's time line before the pause.

STDMETHOD(OnPause) ( 
    THIS_ 
    ULONGLONG ulTime 
) PURE;

ulTime
The time in the stream’s time line before the pause occurred.
**IHXClientAdviseSink::OnPosLength**

Advises the client that the position or length of the current playback context has changed.  

```cpp
STDMETHOD(OnPosLength) (
    THIS_,
    UINT32 ulPosition,
    UINT32 ulLength
) PURE;
```

- **ulPosition**
  - The new position of the playback.

- **ulLength**
  - The new length of the playback.

**IHXClientAdviseSink::OnPostSeek**

Informs the client that a seek has just occurred. The top-level client is informed of the last time in the stream’s time line before the seek, as well as the first new time for the stream’s time line after the seek.

```cpp
STDMETHOD(OnPostSeek) (
    THIS_,
    ULONG32 ulOldTime,
    ULONG32 ulNewTime
) PURE;
```

- **ulOldTime**
  - The end of the stream’s time line before the current seek.

- **ulNewTime**
  - The beginning of the stream’s time line after the current seek.

**IHXClientAdviseSink::OnPreSeek**

Informs the client that a seek is about to occur. The top-level client is informed of the last time in the stream’s time line before the seek, as well as the first new time in the stream’s time line after the seek is completed.

```cpp
STDMETHOD(OnPreSeek) (
    THIS_,
    ULONG32 ulOldTime,
    ULONG32 ulNewTime
) PURE;
```

- **ulOldTime**
  - The end of the stream’s time line before the current seek.

- **ulNewTime**
  - The beginning of the stream’s time line after the current seek.

**IHXClientAdviseSink::OnPresentationClosed**

Advises the client that a presentation has been closed.
STDMETHOD(OnPresentationClosed) ( 
    THIS 
) PURE;

**IHXClientAdviseSink::OnPresentationOpened**

Advises the client that a presentation has been opened.

STDMETHOD(OnPresentationOpened) ( 
    THIS 
) PURE;

**IHXClientAdviseSink::OnStatisticsChanged**

Advises the client that the presentation statistics have changed.

STDMETHOD(OnStatisticsChanged) ( 
    THIS 
) PURE;

**IHXClientAdviseSink::OnStop**

Informs the client that a stop has just occurred.

STDMETHOD(OnStop) ( 
    THIS 
) PURE;
IHXClientAuthConversation

Purpose: Performs the client side of an authentication protocol.
Implemented by: Authentication plug-in
Used by: Clients
Header file: hxauthn.h

This interface responds to an authentication request from the server. The response interface is IHXClientAuthResponse.

The IHXClientAuthConversation interface contains the following methods:

- IHXClientAuthConversation::Authenticated
- IHXClientAuthConversation::IsDone
- IHXClientAuthConversation::MakeResponse

As with all Component Object Model (COM) interfaces, the IHXClientAuthConversation interface inherits the following IUnknown methods:

- IUnknown::AddRef
- IUnknown::QueryInterface
- IUnknown::Release

IHXClientAuthConversation::Authenticated

Reports whether the server accepted or rejected the credentials.

STDMETHOD(Authenticated) (
    0
    BOOL bAuthenticated
) PURE;

bAuthenticated
    If TRUE, the credentials were successfully authenticated.

IHXClientAuthConversation::IsDone

Determines whether the conversation is complete. This method is useful because some protocols have more than one message exchange. Returns TRUE when the conversation is complete.

STDMETHOD_(BOOL,IsDone) (
    0
) PURE;

IHXClientAuthConversation::MakeResponse

Creates a response to the specified request for authentication from the server. For conversations that require multiple round trips, this method is called for each trip.
STDMETHOD(MakeResponse) (  
   THIS_,  
   IHXClientAuthResponse* pClientAuthResponseRequester,  
   IHXRequest* pRequestChallengeHeaders  
) PURE;

pClientAuthResponseRequester
   Pointer to an IHXClientAuthResponse interface that receives the responses generated by this method.

pRequestChallengeHeaders
   Pointer to an IHXRequest interface that manages the server challenge headers.
IHXClientAuthResponse

Purpose: Provides authentication information.
Implemented by: Client core
Used by: Authentication plug-in
Header file: hxauthn.h

This interface is used by plug-ins to respond to authentication requests from the client core. The IHXClientAuthResponse interface contains the IHXClientAuthResponse::ResponseReady method.

As with all Component Object Model (COM) interfaces, the IHXClientAuthResponse interface inherits the following IUnknown methods:
- IUnknown::AddRef
- IUnknown::QueryInterface
- IUnknown::Release

IHXClientAuthResponse::ResponseReady

Reports the success or failure of IHXClientAuthConversation::MakeResponse.

STDMETHOD(ResponseReady) ( 
    THIS_ 
    HX_RESULT ResultStatus, 
    IHXRequest* pRequestResponseHeaders 
) PURE;

ResultStatus
The status of the IHXClientAuthConversation::MakeResponse operation. A value of HXR_OK indicates that the operation has completed successfully.

pRequestResponseHeaders
Pointer to an IHXRequest interface that manages the request response headers. This IHXRequest interface should be the same as that passed in IHXClientAuthConversation::MakeResponse, and should contain CString values for each MIME header that needs to be sent to the server.
IHXClientEngine

Purpose: Provides access to the client core.
Implemented by: Client core
Used by: Top-level client and rendering plug-ins
Header file: hxcore.h

This interface to the client core object enables the top-level client or a rendering plug-in to create new IHXPlayer objects. A rendering plug-in can do this, for example, to start a new presentation timeline.


The IHXClientEngine interface contains the following methods:
- IHXClientEngine::ClosePlayer
- IHXClientEngine::CreatePlayer
- IHXClientEngine::GetPlayer
- IHXClientEngine::GetPlayerCount
- IHXClientEngine::EventOccurred

As with all Component Object Model (COM) interfaces, the IHXClientEngine interface inherits the following IUnknown methods:
- IUnknown::AddRef
- IUnknown::QueryInterface
- IUnknown::Release

IHXClientEngine::ClosePlayer

Destroys an instance of an IHXPlayer interface when the engine has finished using the player.

STDMETHOD(ClosePlayer) (
    THIS_
    IHXPlayer* pPlayer
) PURE;

pPlayer
Points to the IHXPlayer interface being closed.

IHXClientEngine::CreatePlayer

Creates a new instance of an IHXPlayer interface. The top-level client creates a new player object for each presentation. A rendering plug-in can create a new player object to, for example, start a new time and insert an instantaneous sound into the presentation.

STDMETHOD(CreatePlayer) (
    THIS_
    REF(IHXPlayer*) pPlayer
) PURE;
pPlayer
   Returns a pointer to the IHXPlayer interface being created.

IHXClientEngine::GetPlayer

   Retrieves the requested IHXPlayer instance supported by this client engine instance. The top-level client or plug-in uses this method after getting the player object count.

   STDMETHOD(GetPlayer) (  
      THIS_  
      UINT16 nPlayerNumber,  
      REF(IUnknown*) pUnknown
   ) PURE;

   nPlayerNumber
      The number of the player instance.

   pUnknown
      Returns a pointer to a context from which other interfaces can be queried.

IHXClientEngine::GetPlayerCount

   Returns the current number of IHXPlayer instances supported by this client engine instance.

   STDMETHOD_(UINT16, GetPlayerCount) (  
      THIS
   ) PURE;

IHXClientEngine::EventOccurred

   Passes operating system events to all players.

   STDMETHOD(EventOccurred) (  
      THIS_  
      HXxEvent* pEvent
   ) PURE;

   pEvent
      Pointer to a HXxEvent structure that defines the cross-platform event.
In UNIX, top-level clients should use the `IHXClientEngineSelector::Select` method in place of the UNIX `select` function. To the top-level client, this method appears identical to the UNIX `select`, as it includes file descriptor sets and a time-out value. Note, however, that using this method instead of UNIX `select` makes it possible for the client core to perform the necessary actions to enable it to select on the sets.


The `IHXClientEngineSelector` interface contains the `IHXClientEngineSelector::Select` method.

As with all Component Object Model (COM) interfaces, the `IHXClientEngineSelector` interface inherits the following `IUnknown` methods:

- `IUnknown::AddRef`
- `IUnknown::QueryInterface`
- `IUnknown::Release`

### `IHXClientEngineSelector::Select`

Selects events. Top-level clients under UNIX should use this instead of the UNIX `select` function.

```c
STDMETHOD_(INT32, Select) (
    THIS_
    INT32 n,
    fd_set *readfds,
    fd_set *writefds,
    fd_set *exceptfds,
    struct timeval* timeout
) PURE;
```

- **n**
  The highest-numbered descriptor in any of the three descriptor sets, plus 1.

- **readfds**
  Pointer to the descriptors to be watched to see whether characters become available for reading.

- **writefds**
  Pointer to the descriptors to be watched to see whether it is all right to immediately write on them.

- **exceptfds**
  Pointer to the descriptors to be watched for exceptions.
timeout

Pointer to a timeval structure that is an upper bound on the amount of time elapsed before this method returns. This value can be 0 (zero), which causes this method to return immediately. If this value is NULL (no time-out), this method can block indefinitely.
**IHXClientEngineSetup**

Purpose: Overrides certain client core interfaces.
Implemented by: Client core
Used by: Top-level client, rendering plug-ins
Header file: hxcore.h


The IHXClientEngineSetup interface contains the IHXClientEngineSetup::Setup method.

As with all Component Object Model (COM) interfaces, the IHXClientEngineSetup interface inherits the following IUnknown methods:

- IUnknown::AddRef
- IUnknown::QueryInterface
- IUnknown::Release

**IHXClientEngineSetup::Setup**

Overrides certain basic interfaces implemented by the core. Interfaces that can currently be overridden are IHXPreferences and IHXHyperNavigate. This method is used by top-level clients.

```c
STDMETHOD(Setup) (  
    THIS_
    IUUnknown* pContext
) PURE;
```  

pContext

Pointer to a context from which other interfaces are queried.
IHXCommonClassFactory

Purpose: Creates Helix objects.
Implemented by: Helix architecture
Used by: Any component
Header file: hxcomm.h

Any Helix component can use this interface to create a Helix object. This is the preferred method for creating objects used by multiple components. A component can use the C++ new operator to create objects that it alone manipulates, however. When Helix initializes a component, it passes the component a pointer to the system context. The component can then use this pointer to query for IHXCommonClassFactory and call IHXCommonClassFactory::CreateInstance.


The IHXCommonClassFactory interface contains the following methods:
- IHXCommonClassFactory::CreateInstance
- IHXCommonClassFactory::CreateInstanceAggregatable

As with all Component Object Model (COM) interfaces, the IHXCommonClassFactory interface inherits the following IUnknown methods:
- IUnknown::AddRef
- IUnknown::QueryInterface
- IUnknown::Release

IHXCommonClassFactory::CreateInstance

Creates instances of common interfaces that the system supports, such as IHXBuffer, IHXPacket, IHXValues, and so on. This method is similar to Windows’ CoCreateInstance function in its purpose, except it only creates instances of commonly used interface types, such as IHXBuffer, IHXPacket, IHXValues, and so on.

STDMETHOD(CreateInstance) (
  THIS_,
  REFCLSID rclsid,
  void** ppUnknown
) PURE;

rclsid
  The class identifier.

ppUnknown
  Pointer to the instance of the interface.

Note: Aggregation is never used. Therefore an outer unknown is not passed to this function, and you do not need to code for this situation.
IHXCommonClassFactory::CreateInstanceAggregatable

Creates instances of common interfaces (that can be aggregated) that the system supports. This method is similar to Window’s CoCreateInstance function in its purpose, except it only creates instances of commonly used interface types, such as IHXBuffer, IHXPacket, IHXValues, and so on.

Unlike IHXCommonClassFactory::CreateInstance, this method will create internal objects that support aggregation.

The output interface is always the nondelegating IUnknown.

STDMETHOD(CreateInstanceAggregatable) (
    THIS_ 
    REFCLSID rclsid,
    REF(IUnknown*) ppUnknown,
    IUnknown* pUnkOuter
) PURE;

rclsid
The class identifier.

ppUnknown
Returns a pointer to a context from which other interfaces can be queried.

pUnkOuter
Pointer to an IUnknown interface that identifies the outer unknown used for aggregation purposes.
IHXConfigFile

Purpose: Manages the server's configuration file.
Implemented by: Server core
Used by: Server plug-ins
Header file: hxcfg.h

This interface manages the configurations stored in the server's configuration file.
The IHXConfigFile interface contains the following methods:

- IHXConfigFile::GetFilename
- IHXConfigFile::LoadFrom
- IHXConfigFile::Reload
- IHXConfigFile::Save
- IHXConfigFile::SaveAs
- IHXConfigFile::SetFilename

As with all Component Object Model (COM) interfaces, the IHXConfigFile interface inherits the following IUnknown methods:

- IUnknown::AddRef
- IUnknown::QueryInterface
- IUnknown::Release

IHXConfigFile::GetFilename
Retrieves the current, default file.

STDMETHOD(GetFilename) (THIS_ REF(IHXBuffer*) pFilename) PURE;

pFilename
Returns a pointer to an IHXBuffer interface that manages the current, default file name.

IHXConfigFile::LoadFrom
Loads the specified configuration file and sets that file as the default for future reloads and saves.

STDMETHOD(LoadFrom) (THIS_ IHXBuffer* filename) PURE;

filename
Pointer to an IHXBuffer interface that manages the file name of the configuration file.

IHXConfigFile::Reload
Reloads the current, default configuration file.
STDMETHOD(Reload) (THIS) PURE;

IHXConfigFile::Save

Writes the current configuration to the current default file.

STDMETHOD(Save) (THIS) PURE;

IHXConfigFile::SaveAs

Writes the configuration to the named file, and sets that file as the default.

STDMETHOD(SaveAs) (THIS_IHXBuffer* pFilename) PURE;

pFilename
Pointer to an IHXBuffer interface that manages the file name.

IHXConfigFile::SetFilename

Sets the current, default file, but does not read it or change its contents.

STDMETHOD(SetFilename) (THIS_IHXBuffer* pFilename) PURE;

pFilename
Pointer to an IHXBuffer interface that manages the current, default file name.
**IHXConnectionlessControl**

Purpose: Allows use of the remote broadcast library without a TCP connection to the server.

Implemented by: Remote broadcast library

Used by: Remote broadcast applications

Header file: \hxencod.h

This interface causes the remote broadcast library to connect to the server without a TCP control channel. Using `IUnknown::QueryInterface`, the remote broadcast application can query `IHXEncoderResponse` for this interface. The application can then call `IHXConnectionlessControl::EnableConnectionlessControl` before calling `IHXEncoderResponse::InitEncoderResponse`.

The `IHXConnectionlessControl` interface contains the following methods:

- `IHXConnectionlessControl::ConnectionCheckFailed`
- `IHXConnectionlessControl::EnableConnectionlessControl`
- `IHXConnectionlessControl::SetConnectionTimeout`

As with all Component Object Model (COM) interfaces, the `IHXConnectionlessControl` interface inherits the following `IUnknown` methods:

- `IUnknown::AddRef`
- `IUnknown::QueryInterface`
- `IUnknown::Release`

**IHXConnectionlessControl::ConnectionCheckFailed**

This method is obsolete and should no longer be used.

```
STDMETHOD(ConnectionCheckFailed) ( 
    THIS_
    HX_RESULT status
) PURE;
```

**IHXConnectionlessControl::EnableConnectionlessControl**

Informs the remote broadcast library that when it connects to the server it should do so without using a TCP control channel.

```
STDMETHOD(EnableConnectionlessControl) ( 
    THIS
) PURE;
```

**IHXConnectionlessControl::SetConnectionTimeout**

Sets how long the remote broadcast library will wait for a ping response from the server.
STDMETHOD(SetConnectionTimeout) (
    THIS_,
    UINT32 uSeconds
) PURE;

_uSeconds
    The connection time-out, in seconds.
IHXContextMenu

Purpose: Provides context menus.
Implemented by: Top-level client
Used by: Rendering plug-in
Header file: hxcmenu.h

This interface enables the renderer to show a context menu and enables the top-level client to add client-specific commands unknown to the renderer to that menu. The response interface is IHXContextMenuResponse.

The IHXContextMenu interface contains the following methods:

- IHXContextMenu::AddChildMenu
- IHXContextMenu::AddMenuItem
- IHXContextMenu::AddSeparator
- IHXContextMenu::ChangeMenuItem
- IHXContextMenu::EndChildMenu
- IHXContextMenu::InitContextMenu
- IHXContextMenu::ShowMenu

As with all Component Object Model (COM) interfaces, the IHXContextMenu interface inherits the following IUnknown methods:

- IUnknown::AddRef
- IUnknown::QueryInterface
- IUnknown::Release

IHXContextMenu::AddChildMenu

Adds a child menu. This method is not currently implemented.

STDMETHOD(AddChildMenu) ( 
    THIS_
    const char* pMenuText
) PURE;

IHXContextMenu::AddMenuItem

Adds a menu item to the context menu.

STDMETHOD(AddMenuItem) ( 
    THIS_
    UINT16 commandID,
    const char* pMenuItemText,
    BOOL bChecked,
    BOOL bRadioOn,
    BOOL bDisabled
) PURE;
commandID
The unique identification for the command.

pMenuItemText
Pointer to the text associated with this particular menu item.

bChecked
If TRUE, a check is added to the side of this menu item. If FALSE, the check is removed from beside this menu item.

bRadioOn
If TRUE, a radio button is added to the side of this menu item. If FALSE, the radio button is removed from beside this menu item.

bDisabled
If TRUE, the menu item is disabled. If FALSE, the menu item is enabled.

IHXContextMenu::AddSeparator
Adds a graphical separator between menu items.

STDMETHOD(AddSeparator) (
    THIS
) PURE;

IHXContextMenu::ChangeMenuItem
Changes the indicated menu item.

STDMETHOD(ChangeMenuItem) (
    THIS, 
    UINT16 commandID, 
    const char* pMenuItemText, 
    BOOL bChecked, 
    BOOL bRadioOn, 
    BOOL bDisabled
) PURE;

commandID
The unique identification for the command.

pMenuItemText
Pointer to the text associated with this particular menu item.

bChecked
If TRUE, a check is added to the side of this menu item. If FALSE, the check is removed from beside this menu item.

bRadioOn
If TRUE, a radio button is added to the side of this menu item. If FALSE, the radio button is removed from beside this menu item.

bDisabled
If TRUE, the menu item is disabled. If FALSE, the menu item is enabled.
**IHXContextMenu::EndChildMenu**

Removes the child menu. This method is not currently implemented.

```cpp
STDMETHOD(EndChildMenu) (THIS ) PURE;
```

**IHXContextMenu::InitContextMenu**

Initializes the context menu to a blank menu and sets the name of the sub-menu for the renderer, if appropriate. This clears any previously added menu items and sub-menus.

```cpp
STDMETHOD(InitContextMenu) (THIS const char* pMenuText ) PURE;
```

- **pMenuText**
  - Pointer to the text of the menu item to be added to the renderer’s list of items.

**IHXContextMenu::ShowMenu**

Shows the setup context menu at the specified point.

```cpp
STDMETHOD(ShowMenu) (THIS IHXContextMenuResponse* pResponse, HXxPoint ptPopup ) PURE;
```

- **pResponse**
  - Pointer to an IHXContextMenuResponse interface that manages the response to this interface.

- **ptPopup**
  - An HXxPoint structure that defines the location of the context menu.
**IHXContextMenuResponse**

**Purpose:** Responds to context menu commands and cancellations.

**Implemented by:** Rendering plug-in

**Used by:** Top-level client

**Header file:** hxcmenu.h

This interface is called when the context menu is dismissed.

The IHXContextMenuResponse interface contains the following methods:

- IHXContextMenuResponse::OnCanceled
- IHXContextMenuResponse::OnCommand

As with all Component Object Model (COM) interfaces, the IHXContextMenuResponse interface inherits the following IUnknown methods:

- IUnknown::AddRef
- IUnknown::QueryInterface
- IUnknown::Release

**IHXContextMenuResponse::OnCanceled**

Informs the renderer that the context menu was closed without a command being chosen from the renderer’s set of commands.

```cpp
STDMETHOD(OnCanceled) (THIS) PURE;
```

**IHXContextMenuResponse::OnCommand**

Informs the renderer that a command was chosen from the context menu.

```cpp
STDMETHOD(OnCommand) (THIS, UINT16 commandID) PURE;
```

**commandID**

The unique identification for the command.
IHXCcopyRegistry

Purpose: Copies from one registry key to another.
Implemented by: Server core
Used by: Server plug-ins
Header file: hxmon.h

This interface copies information from one registry key to another. This interface is queried from IHXRegistry.

The IHXCcopyRegistry interface contains the IHXCcopyRegistry::CopyByName method.

As with all Component Object Model (COM) interfaces, the IHXCcopyRegistry interface inherits the following IUnknown methods:

- IUnknown::AddRef
- IUnknown::QueryInterface
- IUnknown::Release

IHXCcopyRegistry::CopyByName

Copies from one registry key to another.

STDMETHOD (CopyByName) ( THIS_,
   const char* pFrom,
   const char* pTo
) PURE;

pFrom
   Pointer to the registry key from which to copy.

pTo
   Pointer to the registry key to which to copy.
This interface gets information used for authentication of the client that is not necessarily only the plain-text user name and password. This interface is queried from the response interface passed into IHXClientAuthConversation::MakeResponse, which uses it to request that the current user enters his or her credentials. The response interface is IHXCredRequestResponse.

The IHXCredRequest interface contains the IHXCredRequest::GetCredentials method.

As with all Component Object Model (COM) interfaces, the IHXCredRequest interface inherits the following IUnknown methods:

- IUnknown::AddRef
- IUnknown::QueryInterface
- IUnknown::Release

IHXCredRequest::GetCredentials

Requests the user’s credentials. You can use this method to present a dialog box to the user, asking for his or her user name and password, or you can use it to supply the credentials from the command line (from an encoding tool, for example).

STDMETHOD(GetCredentials) ( 
THIS_
 IHXCredRequestResponse* pCredRequestResponseRequester, 
IHXValues* pValuesCredentialRequest
 ) PURE;

pCredRequestResponseRequester
Pointer to an IHXCredRequestResponse interface that manages the response to the plug-in’s request for credentials.

pValuesCredentialRequest
Pointer to an IHXValues interface that manages related credential data. While ignored at this time, this parameter should contain CString properties that describe the reason for the request (such as the URL, the realm, the authentication protocol, how secure it is, and so on). In the future, this data will be displayed to the user.
IHXCredRequestResponse

Purpose: Returns the user name and password values.
Implemented by: Authentication plug-in
Used by: Client authenticator
Header file: hxauthn.h

This interface receives the credentials requested in the IHXCredRequest::GetCredentials method.
The IHXCredRequestResponse interface contains the IHXCredRequestResponse::CredentialsReady method.
As with all Component Object Model (COM) interfaces, the IHXCredRequestResponse interface inherits the following IUnknown methods:
• IUnknown::AddRef
• IUnknown::QueryInterface
• IUnknown::Release

IHXCredRequestResponse::CredentialsReady

Reports the status of the call to IHXCredRequest::GetCredentials.

STDMETHOD(CredentialsReady) (
    THIS_
    HX_RESULT ResultStatus,
    IHXValues* pValuesCredentials
) PURE;

ResultStatus
The status of the IHXCredRequest::GetCredentials operation. A value of HXR_OK indicates that the operation has completed successfully.

pValuesCredentials
Pointer to an IHXValues interface that handles the requested credentials. These values are usually in the form of a CString for the user name and a CString for the password.
**IHXDatabaseManager**

Purpose: Provides access to databases.

Implemented by: Database manager

Used by: Authentication plug-in

Header file: hxdb.h

This interface provides plug-ins with an instance of the supported database.

The IHXDatabaseManager interface contains the IHXDatabaseManager::GetInstanceFromID method.

As with all Component Object Model (COM) interfaces, the IHXDatabaseManager interface inherits the following IUnknown methods:

- IUnknown::AddRef
- IUnknown::QueryInterface
- IUnknown::Release

**IHXDatabaseManager::GetInstanceFromID**

Returns a database object configured as defined for the specified database ID in the server configuration file.

```cpp
STDMETHOD(GetInstanceFromID) (THIS_
    IHXBuffer* pBufferID,
    REF(IUnknown*) pUnknownDatabase
) PURE;
```

**pBufferID**

Pointer to an IHXBuffer interface that manages the database ID.

**pUnknownDatabase**

Returns a pointer to the IUnknown interface that identifies the instance of the database object.
**IHXDataConvert**

Purpose: Provides data and header conversion.
Implemented by: Server side plug-in
Used by: Server core
Header file: hxdtcvt.h

This interface is called by the server core when transmitting data, such as file headers, stream headers, and data packets, to the client. The methods of this interface perform the actual conversion of the data stream. The response interface is IHXDataConvertResponse.

For More Information: See “Chapter 5: Data Conversion and Reversion Plug-Ins” beginning in Volume 1, on page 73.

The IHXDataConvert interface contains the following methods:
- IHXDataConvert::AddMulticastControlConverter
- IHXDataConvert::ControlBufferReady
- IHXDataConvert::ConvertData
- IHXDataConvert::ConvertFileHeader
- IHXDataConvert::ConvertStreamHeader
- IHXDataConvert::DataConvertInit
- IHXDataConvert::Done
- IHXDataConvert::GetConversionMimeType
- IHXDataConvert::SetMulticastTransportConverter

As with all Component Object Model (COM) interfaces, the IHXDataConvert interface inherits the following IUnknown methods:
- IUnknown::AddRef
- IUnknown::QueryInterface
- IUnknown::Release

**IHXDataConvert::AddMulticastControlConverter**

Adds one of many possible data conversion interfaces that will be handling the header conversions. In this method (as opposed to IHXDataConvert::SetMulticastTransportConverter), the data conversion object is handling the data conversions for all of the players (but only once, because it is multicast).

STDMETHOD(AddMulticastControlConverter) (  
    THIS_  
    IHXDataConvert* pConverter
  ) PURE;

pConverter
  Pointer to an IHXDataConvert interface that manages the header conversion.
**IHXDataConvert::ControlBufferReady**

Receives a control channel buffer sent from the IHXDataRevert interface on the player.

```cpp
STDMETHOD(ControlBufferReady) (THIS_ IHXBuffer* pBuffer) PURE;
```

*pBuffer*

Pointer to an IHXBuffer interface that manages control channel information from the player.

**IHXDataConvert::ConvertData**

Converts the data in the indicated packet.

```cpp
STDMETHOD(ConvertData) (THIS_ IHXPacket* pPacket) PURE;
```

*pPacket*

Pointer to an IHXPacket interface that manages the data to be converted.

*Note:* If you call the IHXDataConvertResponse::ConvertedDataReady method with a result of HXR_OK and a NULL buffer, the system uses the original packet data (that is, no conversion takes place).

**IHXDataConvert::ConvertFileHeader**

Converts the indicated file header data.

```cpp
STDMETHOD(ConvertFileHeader) (THIS_ IHXValues* pFileHeader) PURE;
```

*pFileHeader*

Pointer to an IHXValues interface that manages the file header to be converted.

*Note:* If you call the IHXDataConvertResponse::ConvertedFileHeaderReady method with a result of HXR_OK and a NULL buffer, the system uses the original header data (that is, no conversion takes place).

**IHXDataConvert::ConvertStreamHeader**

Converts the indicated stream header data.

```cpp
STDMETHOD(ConvertStreamHeader) (THIS_ IHXValues* pStreamHeader) PURE;
```

*pStreamHeader*

Pointer to an IHXValues interface that manages the stream header to be converted.
Note: If you call the IHXDataConvertResponse::ConvertedStreamHeaderReady method with a result of HXR_OK and a NULL buffer, the system uses the original header data (that is, no conversion takes place).

**IHXDataConvert::DataConvertInit**

Initializes the data conversion interface.

```cpp
STDMETHOD(DataConvertInit) ( 
    IHXDataConvertResponse* pResponse
) PURE;
```

*pResponse*

Pointer to an IHXDataConvertResponse interface that manages the responses to this interface’s methods.

**IHXDataConvert::Done**

Indicates that the data conversion process is completed.

```cpp
STDMETHOD(Done) ( 
    THIS
) PURE;
```

**IHXDataConvert::GetConversionMimeType**

Gets the conversion MIME type for the conversion session.

```cpp
STDMETHOD(GetConversionMimeType) ( 
    REF(const char*) pConversionType
) PURE;
```

*pConversionType*

Returns a pointer to the conversion MIME type for this session.

**IHXDataConvert::SetMulticastTransportConverter**

Creates a new instance of a data conversion interface. In this case, the current data conversion interface is only handling the header conversions for the player. The new data conversion interface created by this method handles the data to be converted.

```cpp
STDMETHOD(SetMulticastTransportConverter) ( 
    IHXDataConvert* pConverter
) PURE;
```

*pConverter*

Pointer to an IHXDataConvert interface that manages the data to be converted.
IHXDataConvertResponse

Purpose: Returns the status of the data conversion operation.
Implemented by: Server core
Used by: Server side plug-in
Header file: hxdtcvt.h

This interface returns the status and conversion data. You can have the data returned unchanged, or you can have it returned after being modified (such as by being encrypted).

For More Information: See “Chapter 5: Data Conversion and Reversion Plug-Ins” beginning in Volume 1, on page 73.

The IHXDataConvertResponse interface contains the following methods:

- IHXDataConvertResponse::ConvertedDataReady
- IHXDataConvertResponse::ConvertedFileHeaderReady
- IHXDataConvertResponse::ConvertedStreamHeaderReady
- IHXDataConvertResponse::DataConvertInitDone
- IHXDataConvertResponse::SendControlBuffer

As with all Component Object Model (COM) interfaces, the IHXDataConvertResponse interface inherits the following IUnknown methods:

- IUnknown::AddRef
- IUnknown::QueryInterface
- IUnknown::Release

IHXDataConvertResponse::ConvertedDataReady

Provides asynchronous notification that the data conversion interface has finished converting the stream data packet.

STDMETHOD(ConvertedDataReady) (  
  THIS_  
  HX_RESULT status,  
  IHXPacket* pPacket  
) PURE;

status
  The status of the IHXDataConvert::ConvertData operation. A value of HXR_OK indicates that the operation was completed successfully.

pPacket
  Pointer to an IHXPacket interface that manages the stream data.

IHXDataConvertResponse::ConvertedFileHeaderReady

Provides asynchronous notification that the data conversion interface has finished converting the file header.
STDMETHOD(ConvertedFileHeaderReady) (  
    THIS_  
    HX_RESULT status,  
    IHXValues* pFileHeader  
) PURE;

status  
The status of the IHXDataConvert::ConvertFileHeader operation. A value of HXR_OK indicates that the operation was completed successfully.

pFileHeader  
Pointer to an IHXValues interface that manages the file header.

IHXDataConvertResponse::ConvertedStreamHeaderReady  
Provides asynchronous notification that the data conversion interface has finished converting the stream header.

STDMETHOD(ConvertedStreamHeaderReady) (  
    THIS_  
    HX_RESULT status,  
    IHXValues* pStreamHeader  
) PURE;

status  
The status of the IHXDataConvert::ConvertStreamHeader operation. A value of HXR_OK indicates that the operation was completed successfully.

pFileHeader  
Pointer to an IHXValues interface that manages the stream header.

IHXDataConvertResponse::DataConvertInitDone  
Provides asynchronous notification that the data conversion interface has finished initializing.

STDMETHOD(DataConvertInitDone) (  
    THIS_  
    HX_RESULT status  
) PURE;

status  
The status of the IHXDataConvert::DataConvertInit operation. A value of HXR_OK indicates that the operation was completed successfully.

IHXDataConvertResponse::SendControlBuffer  
Sends an arbitrary buffer to the data reversion interface on the player.

STDMETHOD(SendControlBuffer) (  
    THIS_  
    IHXBuffer* pBuffer  
) PURE;
pBuffer
Pointer to an IHXBuffer interface that manages the data sent to the data reversion interface.
IHXDataConvertSystemObject

Purpose: Controls data conversion plug-ins.
Implemented by: Data conversion plug-ins
Used by: Helix Universal Server
Header file: hxdtcvt.h

All data conversion plug-ins must implement this interface. Helix Universal Server uses this interface to initialize the data conversion plug-in. It calls the IHXDataConvertSystemObject::CreateDataConvert method to create data conversion objects accessed through IHXDataConvert.

For More Information: See “Chapter 5: Data Conversion and Reversion Plug-Ins” beginning in Volume 1, on page 73.

The IHXDataConvertSystemObject interface contains the following methods:

- IHXDataConvertSystemObject::CreateDataConvert
- IHXDataConvertSystemObject::GetDataConvertInfo
- IHXDataConvertSystemObject::InitDataConvertSystem

As with all Component Object Model (COM) interfaces, the IHXDataConvertSystemObject interface inherits the following IUnknown methods:

- IUnknown::AddRef
- IUnknown::QueryInterface
- IUnknown::Release

IHXDataConvertSystemObject::CreateDataConvert

Creates a new data conversion object.

STDMETHOD(CreateDataConvert) (THIS_ IUnknown** ppConvObj)

ppConvObj
Pointer to the object that implements the IHXDataConvert interface.

IHXDataConvertSystemObject::GetDataConvertInfo

Returns information needed to properly instantiate the data conversion plug-in.

STDMETHOD(GetDataConvertInfo) (THIS_ REF(const char*) pShortName)

pShortName
A short, human-readable name of the form “company-dcname”. For example, this parameter could be set to “rn-dataconvert”.

IHXDataConvertSystemObject
**IHXDataConvertSystemObject::InitDataConvertSystem**

Passes in options from the server configuration file (`rmserver.cfg`) for this specific plug-in. Any parameters under the second level lists under the `DataConvertMount` list name are passed in as options by this method.

```c
STDMETHOD(InitDataConvertSystem) ( 
    THIS_ 
    IHXValues* pOptions 
) PURE;
```

**pOptions**

Pointer to an `IHXValues` interface that manages the options passed in from the server configuration file.
IHXDataRevert

Purpose: Restores data and header information.
Implemented by: Client-side plug-in
Used by: Client core
Header file: hxdtcvt.h

This interface restores any information that was converted on the server, using the IHXDataConvert interface, to its original form.

For More Information: See “Chapter 5: Data Conversion and Reversion Plug-Ins” beginning in Volume 1, on page 73.

The IHXDataRevert interface contains the following methods:

- IHXDataRevert::ControlBufferReady
- IHXDataRevert::DataRevertInit
- IHXDataRevert::GetDataRevertInfo
- IHXDataRevert::RevertData
- IHXDataRevert::RevertFileHeader
- IHXDataRevert::RevertStreamHeader

As with all Component Object Model (COM) interfaces, the IHXDataRevert interface inherits the following IUnknown methods:

- IUnknown::AddRef
- IUnknown::QueryInterface
- IUnknown::Release

IHXDataRevert::ControlBufferReady

Passes in a control channel buffer received from corresponding data conversion interface on the server.

STDMETHOD(ControlBufferReady) (  
    THIS_  
    IHXBuffer* pBuffer  
) PURE;

pBuffer
    Pointer to an IHXBuffer interface that manages the control data received from the server.

IHXDataRevert::DataRevertInit

Initializes the data reversion interface.

STDMETHOD(DataRevertInit) (  
    THIS_  
    IHXDataRevertResponse* pResponse  
) PURE;
pResponse
Pointer to an IHXDataRevertResponse interface that manages the responses to this interface’s methods.

IHXDataRevert::GetDataRevertInfo
Notifies the player core about which data conversion MIME types can be handled.

STDMETHOD(GetDataRevertInfo) (THIS_REF(const char**) ppConversionMimeTypes) PURE;

ppConversionMimeTypes
Returns the address of a pointer for the data conversion MIME types.

IHXDataRevert::RevertData
Restores the converted stream data to its original form.

STDMETHOD(RevertData) (THIS_IHXPacket* pPacket) PURE;

pPacket
Pointer to an IHXPacket interface that manages the data to be restored.

IHXDataRevert::RevertFileHeader
Restores the converted file header to its original format.

STDMETHOD(RevertFileHeader) (THIS_IHXValues* pFileHeader) PURE;

pFileHeader
Pointer to an IHXValues interface that manages the file header to be restored.

IHXDataRevert::RevertStreamHeader
Restores the converted stream header to its original format.

STDMETHOD(RevertStreamHeader) (THIS_IHXValues* pStreamHeader) PURE;

pStreamHeader
Pointer to an IHXValues interface that manages the stream header to be restored.
IHXDataRevertResponse

Purpose: Acknowledges the restoration of data and header information.
Implemented by: Client core
Used by: Client-side plug-in
Header file: hxdtcvt.h

This interface provides an asynchronous response to the IHXDataRevert methods.

For More Information: See “Chapter 5: Data Conversion and Reversion Plug-Ins” beginning in Volume 1, on page 73.

The IHXDataRevertResponse interface contains the following methods:

- IHXDataRevertResponse::DataRevertInitDone
- IHXDataRevertResponse::RevertedDataReady
- IHXDataRevertResponse::RevertedFileHeaderReady
- IHXDataRevertResponse::RevertedStreamHeaderReady
- IHXDataRevertResponse::SendControlBuffer

As with all Component Object Model (COM) interfaces, the IHXDataRevertResponse interface inherits the following IUnknown methods:

- IUnknown::AddRef
- IUnknown::QueryInterface
- IUnknown::Release

IHXDataRevertResponse::DataRevertInitDone

Provides asynchronous notification that the data reversion interface has finished initializing and can begin processing headers.

STDMETHOD(DataRevertInitDone) (THIS_ HX_RESULT status ) PURE;

status

The status of the IHXDataRevert::DataRevertInit operation. A value of HXR_OK indicates that the operation was completed successfully.

IHXDataRevertResponse::RevertedDataReady

Provides asynchronous notification that the data reversion interface has finished reverting the stream data.

STDMETHOD(RevertedDataReady) (THIS_ HX_RESULT status,
IHXPacket* pPacket ) PURE;
**status**
The status of the `IHXDataRevert::RevertData` operation. A value of `HXR_OK` indicates that the operation was completed successfully.

**pPacket**
Pointer to an `IHXPacket` interface that manages the reverted stream data.

**IHXDataRevertResponse::RevertedFileHeaderReady**
Provides asynchronous notification that the data reversion interface has finished reverting the file headers.

```cpp
STDMETHOD(RevertedFileHeaderReady) (
    THIS_ 
    HX_RESULT status,
    IHXValues* pHeader
) PURE;
```

**status**
The status of the `IHXDataRevert::RevertFileHeader` operation. A value of `HXR_OK` indicates that the operation was completed successfully.

**pHeader**
Pointer to an `IHXValues` interface that manages the reverted file header.

**IHXDataRevertResponse::RevertedStreamHeaderReady**
Provides asynchronous notification that the data reversion interface has finished reverting the stream headers.

```cpp
STDMETHOD(RevertedStreamHeaderReady) (
    THIS_ 
    HX_RESULT status,
    IHXValues* pHeader
) PURE;
```

**status**
The status of the `IHXDataRevert::RevertStreamHeader` operation. A value of `HXR_OK` indicates that the operation was completed successfully.

**pHeader**
Pointer to an `IHXValues` interface that manages the reverted stream header.

**IHXDataRevertResponse::SendControlBuffer**
Sends an arbitrary control buffer to the data conversion interface on the server.

```cpp
STDMETHOD(SendControlBuffer) (
    THIS_ 
    IHXBuffer* pBuffer
) PURE;
```

**pBuffer**
Pointer to an `IHXBuffer` interface that manages the data sent to the data conversion interface.
IHXDirHandler

Purpose: Allows you to create or read a specified directory.
Implemented by: File system plug-ins
Used by: Any plug-in that can gain access to the file system
Header file: hxfiles.h

This interface is for making a directory or reading a directory to get information from the file system. To access a particular file in a directory, use the IHXFileSystemManager interface.

The IHXDirHandler interface contains the following methods:
- IHXDirHandler::CloseDirHandler
- IHXDirHandler::InitDirHandler
- IHXDirHandler::MakeDir
- IHXDirHandler::ReadDir

As with all Component Object Model (COM) interfaces, the IHXDirHandler interface inherits the following IUnknown methods:
- IUnknown::AddRef
- IUnknown::QueryInterface
- IUnknown::Release

IHXDirHandler::CloseDirHandler

Closes the directory handler resource and releases all resources associated with the object that implements this interface.

STDMETHOD(CloseDirHandler) (THIS)

IHXDirHandler::InitDirHandler

Associates an IHXDirHandlerResponse interface with the IHXDirHandler interface. This interface reports when various IHXDirHandler methods have completed.

STDMETHOD(InitDirHandler) (THIS,
IHXDirHandlerResponse* pDirResponse)

pDirResponse
Pointer to an IHXDirHandlerResponse interface that manages the response to this interface.

IHXDirHandler::MakeDir

Creates the directory.
STDMETHOD(MakeDir) (THIS) PURE;

IHXDirHandler::ReadDir

Reads the directory.

STDMETHOD(ReadDir) (THIS) PURE;
IHXDirHandlerResponse

Purpose: Returns directory information.
Implemented by: File system plug-ins
Used by: Any plug-in that can gain access to the file system
Header file: hxfiles.h

This is the response interface to IHXDirHandler.
The IHXDirHandlerResponse interface contains the following methods:
   • IHXDirHandlerResponse::CloseDirHandlerDone
   • IHXDirHandlerResponse::InitDirHandlerDone
   • IHXDirHandlerResponse::MakeDirDone
   • IHXDirHandlerResponse::ReadDirDone

As with all Component Object Model (COM) interfaces, the IHXDirHandlerResponse interface inherits
the following IUnknown methods:
   • IUnknown::AddRef
   • IUnknown::QueryInterface
   • IUnknown::Release

IHXDirHandlerResponse::CloseDirHandlerDone

Indicates that the closing the directory handler was completed.
STDMETHOD(CloseDirHandlerDone) ( THIS_ HX_RESULT status ) PURE;

status
   The status of the IHXDirHandler::CloseDirHandler operation. A value of HXR_OK indicates that the
   operation was completed successfully.

IHXDirHandlerResponse::InitDirHandlerDone

Indicates that the initialization of the IHXDirHandlerResponse interface was completed.
STDMETHOD(InitDirHandlerDone) ( THIS_ HX_RESULT status ) PURE;

status
   The status of the IHXDirHandler::InitDirHandler operation. A value of HXR_OK indicates that the
   operation was completed successfully.

IHXDirHandlerResponse::MakeDirDone

Indicates that the attempt to create a directory was completed.
**STDMethod(MakeDirDone)**

```c
THIS_
HX_RESULT status
) PURE;
```

**status**
The status of the IHXDirHandler::MakeDir operation. A value of HXR_OK indicates that the operation was completed successfully.

**IHXDirHandlerResponse::ReadDirDone**

Indicates that the read operation from the directory was completed and that a buffer is available.

```c
STDMethod(ReadDirDone) ( 
THIS_
HX_RESULT status,
IHXBuffer* pBuffer
) PURE;
```

**status**
The status of the IHXDirHandler::ReadDir operation. A value of HXR_OK indicates that the operation was completed successfully.

**pBuffer**
Pointer to a IHXBuffer interface that manages the directory information.
IHXDrawFocus

Purpose: Highlights the site that has keyboard focus.
Implemented by: Site object (client core)
Used by: Rendering plug-ins
Header file: hwin.h

This interface is called by a rendering plug-in to optionally draw a visible highlight around the area on
the site that currently has keyboard focus. In addition, when the site receives an HX_LOSE_FOCUS
event, this interface is called to remove the visible highlight.

The IHXDrawFocus interface contains the following methods:

- IHXDrawFocus::ClearFocus
- IHXDrawFocus::SetFocusEllipse
- IHXDrawFocus::SetFocusPolygon
- IHXDrawFocus::SetFocusRect
- IHXDrawFocus::SetStyle

As with all Component Object Model (COM) interfaces, the IHXDrawFocus interface inherits the
following IUnknown methods:

- IUnknown::AddRef
- IUnknown::QueryInterface
- IUnknown::Release

IHXDrawFocus::ClearFocus

Clears the current focus drawing.

STDMETHOD(ClearFocus) ( THIS ) PURE;

IHXDrawFocus::SetFocusEllipse

Draws an ellipse around the focus. This method is not currently implemented, and is intended for
future use.

STDMETHOD(SetFocusEllipse) ( THIS_,
    HXxRect* pRect ) PURE;

pRect
    Pointer to an HXxRect structure that defines the outer limits of the ellipse.

IHXDrawFocus::SetFocusPolygon

Draws a polygon around the focus.
STDMETHOD(SetFocusPolygon) (  
    THIS_  
    HXxPoint* pPoints,  
    ULONG32 numPoints  
) PURE;

  pPoints
  Pointer to an HXxPoint structure that defines the location of the first point on the polygon.

  numPoints
  Defines the number of points that make up the polygon (for example, a polygon made up of three points would form a triangle).

IHXDrawFocus::SetFocusRect

  Draws a rectangle around the focus.
  STDMETHOD(SetFocusRect) (  
    THIS_  
    HXxRect* pRect  
) PURE;

  pRect
  Pointer to an HXxRect structure that defines the corners of the rectangle.

IHXDrawFocus::SetStyle

  Sets the focus style.
  STDMETHOD(SetStyle) (  
    THIS_  
    IHXValues* pProperties  
) PURE;

  pProperties
  Pointer to an IHXValues interface that manages the focus style. The following ULONG properties can be set:

  • LINE_STYLE
    One of either RMA_SOLID_LINE, RMA_DASHED_LINE, RMA_DOTTED_LINE, or RMA_CUSTOM_LINE.

  • LINE_WIDTH
    Width of the line in pixels.

  • RED
    A value of 0 to 255 indicating the color of the primary pixel.

  • GREEN
    A value of 0 to 255 indicating the color of the primary pixel.

  • BLUE
    A value of 0 to 255 indicating the color of the primary pixel.
- RED_OFF
  A value of 0 to 255 indicating the color of the secondary pixel.
- GREEN_OFF
  A value of 0 to 255 indicating the color of the secondary pixel.
- BLUE_OFF
  A value of 0 to 255 indicating the color of the secondary pixel.
- CUSTOM_LINE_ENTRIES
  The number of ULONG32s in CUSTOM_LINE_PATTERN.

In addition, the following IHXBuffer properties can be set:

- CUSTOM_LINE_PATTERN
  A list of ULONG32s describing the number of primary and secondary pixels (for example, 4241 = "----..----.” where “.” is a primary pixel and “.” is a secondary pixel) and CUSTOM_LINE_ENTRIES equals 4. Secondary pixels are not drawn if RED_OFF, GREEN_OFF, and BLUE_OFF are not set.
IHXDryNotification

Purpose: Notifies the renderer of a dry stream.
Implemented by: Audio rendering plug-ins
Used by: Audio Services
Header file: hxausvc.h

Audio rendering plug-ins should implement this interface if they need to receive notification when an audio stream is running dry. A dry stream occurs when Audio Services must write data to the audio device, but it does not have enough data to write. The plug-in sets up a dry stream response object with IHXAudioStream::AddDryNotification.


The IHXDryNotification interface contains the IHXDryNotification::OnDryNotification method.

As with all Component Object Model (COM) interfaces, the IHXDryNotification interface inherits the following IUnknown methods:

- IUnknown::AddRef
- IUnknown::QueryInterface
- IUnknown::Release

IHXDryNotification::OnDryNotification

Indicates that it is time to write data to the audio device but there is not enough data in the audio stream. The renderer can then decide whether to add more data to the audio stream; this should be performed synchronously within the call to this method. However, the renderer is not required to write any data. If no data is available, silence will be played instead.

STDMETHOD(OnDryNotification) ( 
    THIS_ 
    UINT32 ulCurrentStreamTime, 
    UINT32 ulMinimumDurationRequired 
) PURE;

ulCurrentStreamTime
The time in the stream timeline when the next packet is expected.

ulMinimumDurationRequired
The minimum length of the data required to fill the audio stream being sent to the audio device to prevent silence from occurring.
**IHXEncoder**

**Note:** This interface is obsolete and should not be used in any new programming. Use the Remote Broadcast Services interfaces instead.

**For More Information:** See “Remote Broadcast Services” in Volume 1, on page 97.

Purpose: Instructs the broadcast application to send live stream data.

Implemented by: Remote broadcast application

Used by: Remote Broadcast Library

Header file: hxencod.h

Helix’s Remote Broadcast Library uses this interface to instruct a remote broadcast application to send data for a live stream. The response interface is IHXEncoderResponse.

The IHXEncoder interface contains the following methods:

- IHXEncoder::GetFileHeader
- IHXEncoder::GetStreamHeader
- IHXEncoder::InitEncoderResponseDone
- IHXEncoder::StartPackets
- IHXEncoder::StopPackets

As with all Component Object Model (COM) interfaces, the IHXEncoder interface inherits the following IUnknown methods:

- IUnknown::AddRef
- IUnknown::QueryInterface
- IUnknown::Release

**IHXEncoder::GetFileHeader**

Retrieves the header for the entire presentation. Most importantly, it should contain the number of streams present. This method is similar to the IHXFileFormatObject::GetFileHeader method.

```cpp
STDMETHOD(GetFileHeader) ( 
    THIS
) PURE;
```

**IHXEncoder::GetStreamHeader**

Retrieves a header for each stream in the presentation. This method is similar to the IHXFileFormatObject::GetStreamHeader method.

```cpp
STDMETHOD(GetStreamHeader) ( 
    THIS,
    UINT16 unStream
) PURE;
```

unStream

The stream from which to retrieve the header.
**IHXEncoder::InitEncoderResponseDone**

Indicates that the call to IHXEncoderResponse::InitEncoderResponse was completed.

```c
STDMETHOD(InitEncoderResponseDone) ( 
    THIS_ 
    HX_RESULT result 
) PURE;

result
The result of the IHXEncoderResponse::InitEncoderResponse operation. A value of HXR_OK indicates that the operation was completed successfully. If an error value is returned, it should be considered critical and the encoder should act appropriately (that is, it could exit, or it could prompt for new parameters and start over, and so on).
```

**IHXEncoder::StartPackets**

Informs the encoder that it should start sending packets for a particular stream.

```c
STDMETHOD(StartPackets) ( 
    THIS_ 
    UINT16 unStream 
) PURE;

unStream
The stream to which to send the packets.
```

**IHXEncoder::StopPackets**

Indicates that the encoder should stop sending packets for a particular stream.

```c
STDMETHOD(StopPackets) ( 
    THIS_ 
    UINT16 unStream 
) PURE;

unStream
The stream to which to stop sending packets.
```
IHXEncoderCompletion

**Note:** This interface is obsolete and should not be used in any new programming. Use the Remote Broadcast Services interfaces instead.

**For More Information:** See “Remote Broadcast Services” in Volume 1, on page 97.

Purpose: Informs the broadcast application that the stream has stopped.

Implemented by: Remote broadcast application

Used by: Remote Broadcast Library

Header file: hxencod.h

Helix’s Remote Broadcast Library uses this interface to inform a remote broadcast application that it has stopped sending a broadcast stream.

The IHXEncoderCompletion interface contains the IHXEncoderCompletion::EncoderDone method.

As with all Component Object Model (COM) interfaces, the IHXEncoderCompletion interface inherits the following IUnknown methods:

- IUnknown::AddRef
- IUnknown::QueryInterface
- IUnknown::Release

**IHXEncoderCompletion::EncoderDone**

Indicates that the connection has been shut down.

```plaintext
STDMETHOD(EncoderDone) ( 
    THIS_ 
    HX_RESULT result 
) PURE;

result 
    The completion status.
```
IHXEncoderResponse

Note: This interface is obsolete and should not be used in any new programming. Use the Remote Broadcast Services interfaces instead.


Purpose: Informs an encoder object of broadcast application actions.
Implemented by: Remote Broadcast Library
Used by: Remote broadcast application
Header file: hxencod.h

This is the response interface to IHXEncoder. A remote broadcast application uses this interface to reply to instructions sent by the Remote Broadcast Library.

The IHXEncoderResponse interface contains the following methods:
• IHXEncoderResponse::FileHeaderReady
• IHXEncoderResponse::GetTime
• IHXEncoderResponse::InitEncoderResponse
• IHXEncoderResponse::PacketReady
• IHXEncoderResponse::Process
• IHXEncoderResponse::StreamDone
• IHXEncoderResponse::StreamHeaderReady

As with all Component Object Model (COM) interfaces, the IHXEncoderResponse interface inherits the following IUnknown methods:
• IUnknown::AddRef
• IUnknown::QueryInterface
• IUnknown::Release

IHXEncoderResponse::FileHeaderReady

Returns the header from the IHXEncoder::GetFileHeader operation.

STDMETHOD(FileHeaderReady) (
    THIS_
    HX_RESULT result,
    IHXValues* pHeader
) PURE;

result
The result of the IHXEncoder::GetFileHeader operation. A value of HXR_OK indicates that the operation was completed successfully.

pHeader
Pointer to an IHXValues interface that manages the file header that was filled in.
**IHXEncoderResponse::GetTime**

Retrieves the time, in milliseconds, based on the system clock.

```cpp
STDMETHOD_(UINT32,GetTime) ( THIS ) PURE;
```

**IHXEncoderResponse::InitEncoderResponse**

Connects the remote broadcast library to a server to begin the encoding session.

```cpp
STDMETHOD(InitEncoderResponse) ( THIS_,
    const char* pHost,
    UINT16 unPort,
    IHXRequest* pRequest,
    const char* pUsername,
    const char* pPassword,
    IHXEncoder* pEncoder
) PURE;
```

- **pHost**
  Pointer to the host name or IP address of the server.

- **unPort**
  The server port on which the broadcast plug-in listens. This parameter is defined by the plug-in’s `FSMount` setting.

- **pRequest**
  Pointer to an `IHXRequest` interface that manages information associated with the request, including the file name and any name and value pairs to be sent with the request.

- **pUsername**
  Pointer to the name the broadcast application uses to connect to the broadcast plug-in. This should be `encoder` if a password is used.

- **pPassword**
  Pointer to the password the broadcast application uses to connect to the broadcast plug-in. This parameter is defined by the plug-in’s `FSMount` setting.

- **pEncoder**
  Pointer to the broadcast application.

**IHXEncoderResponse::PacketReady**

Returns the packet.

```cpp
STDMETHOD(PacketReady) ( THIS_,
    HX_RESULT result,
    IHXPacket* pPacket
) PURE;
```
result
The result of the packet operation. A value of HXR_OK indicates that the operation was completed successfully.

pPacket
Pointer to an IHXPacket interface that contains the packet information.

IHXEncoderResponse::Process
Performs any necessary processing. This method should be called after each packet is created or at regular intervals (approximately every 5 seconds) if packets are not being sent.

STDMETHOD(Process) (THIS ) PURE;

IHXEncoderResponse::StreamDone
Indicates that the proper number of packets have been sent and that the stream has stopped.

STDMETHOD(StreamDone) (THIS_ UINT16 unStream ) PURE;

unStream
The stream that was receiving the packets.

IHXEncoderResponse::StreamHeaderReady
Returns the header from the IHXEncoder::GetStreamHeader operation.

STDMETHOD(StreamHeaderReady) (THIS_ HX_RESULT result, IHXValues* pHeader ) PURE;

result
The result of the IHXEncoder::GetStreamHeader operation. A value of HXR_OK indicates that the operation was completed successfully.

pHeader
Pointer to an IHXValues interface that manages the file header that was filled in.
IHXEncoderResponseCompletion

Note: This interface is obsolete and should not be used in any new programming. Use the Remote Broadcast Services interfaces instead.


Purpose: Indicates that encoding has been completed.
Implemented by: Remote broadcast library
Used by: Remote broadcast application
Header file: hxencod.h

This interface notifies the remote broadcast library that the remote broadcast application has completed the encoding process.

The IHXEncoderResponseCompletion interface contains the IHXEncoderResponseCompletion::EncoderResponseDone method.

As with all Component Object Model (COM) interfaces, the IHXEncoderResponseCompletion interface inherits the following IUnknown methods:

• IUnknown::AddRef
• IUnknown::QueryInterface
• IUnknown::Release

IHXEncoderResponseCompletion::EncoderResponseDone

Indicates that the encoding process has finished.

STDMETHOD(EncoderResponseDone) (THIS
) PURE;
**IHXErrorMessages**

Purpose: Sends error messages to Helix.

Implemented by: Helix architecture

Used by: Any component

Header file: hxerror.h

Any Helix component can use this interface to report errors to Helix. Errors have a severity level from 0 to 7 and are logged in Helix Universal Server’s error log or displayed on the client in a pop-up dialog box.

For More Information: See “Chapter 7: Status Codes and Errors” beginning in Volume 1, on page 85.

The IHXErrorMessages interface contains the following methods:

- IHXErrorMessages::GetErrorText
- IHXErrorMessages::Report

As with all Component Object Model (COM) interfaces, the IHXErrorMessages interface inherits the following IUnknown methods:

- IUnknown::AddRef
- IUnknown::QueryInterface
- IUnknown::Release

**IHXErrorMessages::GetErrorText**

Gets the text description of a Helix Architecture (HX) error code. Returns an IHXBuffer interface that contains the error text.

```c
STDMETHOD_(IHXBuffer*, GetErrorText) (THIS_ HX_RESULT ulHXCode) PURE;
```

ulHXCode
Contains the Helix Architecture (HX) error code.

**IHXErrorMessages::Report**

Reports an error, event, or status message.

```c
STDMETHOD(Report) (THIS_ const UINT8 unSeverity, HX_RESULT ulHXCode, const ULONG32 ulUserCode, const char* pUserString, const char* pMoreInfoURL ) PURE;
```
unSeverity
Type of report. This value impacts how the player, tool, or server reacts to the report. Depending on the error type, an error message with the Helix Architecture (HX) code and a string translation of that code is displayed. The error dialog box includes a "more info" section that displays the user code and string, and a link to the more info URL. In the server these messages are logged to the log file. One of the following values:

• HXLOG_EMERG
  Indicates a panic condition. The server or client will halt or restart.

• HXLOG_ALERT
  Indicates a condition that should be corrected immediately and needs user intervention to prevent problems.

• HXLOG_CRIT
  Indicates critical conditions.

• HXLOG_ERR
  Indicates errors.

• HXLOG_WARNING
  Indicates warning messages.

• HXLOG_NOTICE
  Indicates conditions that are not error conditions, but should possibly be handled specially.

• HXLOG_INFO
  Indicates informational messages.

• HXLOG_DEBUG
  Indicates messages that contain information normally of use only when debugging a program.

ulHXCode
Helix Architecture (HX) error code, specifically a HX_RESULT code found in hxresult.h. This error code is translated to a text representation for display in an error dialog box or log file.

ulUserCode
User-specific error code. This is not translated to a text representation. This error code can be any value the caller wants; it will be logged or displayed but not interpreted.

pUserString
Pointer to a user-specific error string. This is not translated or modified. This can be any value the caller wants; it will be logged or displayed but not interpreted.

pMoreInfoURL
Pointer to a user-specific more information URL string.
IHXErrorSink

Purpose: Receives error messages.
Implemented by: Error-logging plug-in, top-level client
Used by: Helix architecture
Header file: hxerror.h

If a component sets itself up as an error sink with IHXErrorSinkControl, it receives Helix error messages through the IHXErrorSink::ErrorOccurred method.

For More Information: See “Chapter 7: Status Codes and Errors” beginning in Volume 1, on page 85.

The IHXErrorSink interface contains the IHXErrorSink::ErrorOccurred method.

As with all Component Object Model (COM) interfaces, the IHXErrorSink interface inherits the following IUnknown methods:
  - IUnknown::AddRef
  - IUnknown::QueryInterface
  - IUnknown::Release

IHXErrorSink::ErrorOccurred

Reports an error, event, or status message. This method is called after you have registered your error sink with an IHXErrorSinkControl interface (either in the server or player core).

STDMETHOD(ErrorOccurred) (THIS_ const UINT8 unSeverity, const ULONG32 ulHXCode, const ULONG32 ulUserCode, const char* pUserString, const char* pMoreInfoURL ) PURE;

unSeverity
Type of report. This value impacts how the player, tool, or server reacts to the report. Depending on the error type, an error message with the Helix Architecture (HX) code and a string translation of that code is displayed. The error dialog box includes a "more info" section that displays the user code and string, and a link to the more info URL. In the server these messages are logged to the log file. One of the following values:
  - HXLOG_EMERG
    Indicates a panic condition. The server or client will halt or restart.
  - HXLOG_ALERT
    Indicates a condition that should be corrected immediately and needs user intervention to prevent problems.
  - HXLOG_CRIT
Indicates critical conditions.
- HXLOG_ERR
  Indicates errors.
- HXLOG_WARNING
  Indicates warning messages.
- HXLOG_NOTICE
  Indicates conditions that are not error conditions, but should possibly be handled especially carefully.
- HXLOG_INFO
  Indicates informational messages.
- HXLOG_DEBUG
  Indicates messages that contain information normally of use only when debugging a program.

ulHXCode
Helix Architecture (HX) error code, specifically a HX_RESULT code found in hxresult.h. This error code is translated to a text representation for display in an error dialog box or log file.

ulUserCode
User-specific error code. This is not translated to a text representation. This error code can be any value the caller wants; it will be logged or displayed but not interpreted.

pUserString
Pointer to a user-specific error string. This is not translated or modified. This can be any value the caller wants; it will be logged or displayed but not interpreted.

pMoreInfoURL
Pointer to a user-specific more information URL string.
IHXErrorSinkControl

**Purpose:** Sets up a component to receive error messages.

**Implemented by:** Helix architecture

**Used by:** Error-logging plug-in

**Header file:** hxerror.h

A component sets itself up as an error sink with this interface, specifying the severity levels of errors for which it will receive messages. Helix Universal Server or the client then uses IHXErrorSink to send the component messages about errors as they occur.

**For More Information:** See “Chapter 7: Status Codes and Errors” beginning in Volume 1, on page 85.

The IHXErrorSinkControl interface contains the following methods:

- IHXErrorSinkControl::AddErrorSink
- IHXErrorSinkControl::RemoveErrorSink

As with all Component Object Model (COM) interfaces, the IHXErrorSinkControl interface inherits the following IUnknown methods:

- IUnknown::AddRef
- IUnknown::QueryInterface
- IUnknown::Release

**IHXErrorSinkControl::AddErrorSink**

Adds an IHXErrorSink interface that manages error messages. This method also enables you to set a range of severity levels for which you will receive reports. The default severity range is HXLOG_EMERG to HXLOG_INFO (0 to 6).

```
STDMETHOD(AddErrorSink) (  
    THIS_  
    IHXErrorSink* pErrorSink,  
    const UINT8 unLowSeverity,  
    const UINT8 unHighSeverity  
) PURE;
```

- **pErrorSink**
  Pointer to an IHXErrorSink interface that manages the error messages.

- **unLowSeverity**
  The low end of the range of severity levels to report.

- **unHighSeverity**
  The high end of the range of severity levels to report.

**Note:** You can specify any invalid range (such as unLowSeverity = 1 and unHighSeverity = 0) if you don’t want to receive any errors.
IHXErrorSinkControl::RemoveErrorSink

Removes an error sink.

STDMETHOD(RemoveErrorSink) (THIS_ IHXErrorSink* pErrorSink ) PURE;

pErrorSink

Pointer to the IHXErrorSink interface to be removed.
IHXEventHook

Purpose: Receives handle event calls for a site.
Implemented by: Plug-ins and top-level clients
Used by: Event hook manager, site manager
Header file: hxwin.h

This interface enables a plug-in or top-level client to watch for and react to events for sites used by the plug-in. For example, an image map plug-in can watch for an HX_MOUSE_MOVE and mouse button messages.

The IHXEventHook interface contains the following methods:

• IHXEventHook::HandleEvent
• IHXEventHook::SiteAdded
• IHXEventHook::SiteRemoved

As with all Component Object Model (COM) interfaces, the IHXEventHook interface inherits the following IUnknown methods:

• IUnknown::AddRef
• IUnknown::QueryInterface
• IUnknown::Release

IHXEventHook::HandleEvent

Indicates that an event has occurred at the specified site. The event hook gets called first, in layer order, for each event. If a hook sets the handled member of the HXxEvent structure, the event is handled and the lower layer hooks (the site itself does not receive the event).

STDMETHOD(HandleEvent) ( 
    THIS_ 
    IHXSite* pSite, 
    HXxEvent* pEvent 
) PURE;

pSite
    Pointer to an IHXSite interface that manages the site.

pEvent
    Pointer to an HXxEvent structure that describes the event.

IHXEventHook::SiteAdded

Indicates that a site has been added to the presentation.

STDMETHOD(SiteAdded) ( 
    THIS_ 
    IHXSite* pSite 
) PURE;
pSite
  Pointer to an IHXSite interface that manages the site being added.

IHXEventHook::SiteRemoved

Indicates that a site has been removed from the presentation.

STDMETHOD(SiteRemoved) (
    THIS_
    IHXSite* pSite
) PURE;

pSite
  Pointer to an IHXSite interface that manages the site to be removed.
IHXEventHookMgr

Purpose: Hooks events from a named region.
Implemented by: Site manager (Helix client core)
Used by: Plug-ins and top-level clients
Header file: hxwin.h

This interface adds and removes event hooks.

The IHXEventHookMgr interface contains the following methods:

• IHXEventHookMgr::AddHook
• IHXEventHookMgr::RemoveHook

As with all Component Object Model (COM) interfaces, the IHXEventHookMgr interface inherits the following IUnknown methods:

• IUnknown::AddRef
• IUnknown::QueryInterface
• IUnknown::Release

IHXEventHookMgr::AddHook

Adds an event hook response interface to the named site at the specified layer. Hooks at the same layer are in the order of most recent first.

STDMETHOD(AddHook) (THIS_ IHXEventHook* pHook, const char* pRegionName, UINT16 uLayer ) PURE;

pHook
Pointer to an IHXEventHook interface that manages the event hook.

pRegionName
Pointer to the name of the site to which to add the hook.

Note: If no name is given (that is, if this parameter is NULL), the hook is added to all sites.

uLayer
Indicates the order of the hook if more than one hook is added to a site.

IHXEventHookMgr::RemoveHook

Removes the hooks added by IHXEventHookMgr::AddHook.
STDMETHOD(RemoveHook) (  
    THIS_,  
    IHXEventHook* pHook,  
    const char* pRegionName,  
    UINT16 uLayer  
) PURE;

pHook
    Pointer to an IHXEventHook interface that manages the event hook to be removed.

pRegionName
    Pointer to the name of the region from which to remove the hook.

uLayer
    The layer of the hook to be removed.
IHXFastFileStats

Purpose: Provides the number of bytes that have been “fast-cached”.

Implemented by: Server core

Used by: Server core

Header file: hxfiles.h

This interface displays two lines of additional text about the number of bytes that have been fast-
cached to the standard output (StdOut) log information for Helix Universal Server.

The IHXFastFileStats interface contains the IHXFastFileStats::UpdateFileObjectStats method.

As with all Component Object Model (COM) interfaces, the IHXFastFileStats interface inherits the following IUnknown methods:

- IUnknown::AddRef
- IUnknown::QueryInterface
- IUnknown::Release

IHXFastFileStats::UpdateFileObjectStats

Allows file objects to request that, at close, they be informed about how many bytes, if any, have been
“fast cached” on their behalf.

StDMETHOD(UpdateFileObjectStats) ( 
   _THIS_
   UINT32 ulFastFileBytesSaved,
   UINT32 ulFastFileBytesNeeded
) PURE;

ulFastFileBytesSaved
The number of bytes that would have been read from the file object had the FastFile wrapper not
been in use (this includes some look ahead).

ulFastFileBytesNeeded
The amount of data actually read by the file system.
IHXFileAuthenticator

This interface is obsolete and should not be used in any new programming.

IHXFileAuthenticator::GetAuthenticator

This method is obsolete and should not be used in any new programming.

IHXFileAuthenticator::SetAuthenticator

This method is obsolete and should not be used in any new programming.
**IHXFileExists**

| Purpose: | Determines whether a given file exists in the file system. |
| Implemented by: | File objects (file system plug-ins) |
| Used by: | Helix Universal Server |
| Header file: | hxfiles.h |

When it initializes a file object, Helix Universal Server calls this interface’s method to determine whether the requested file exists on the file system. The file object responds through IHXFileExistsResponse.


The IHXFileExists interface contains the IHXFileExists::DoesExist method.

As with all Component Object Model (COM) interfaces, the IHXFileExists interface inherits the following IUnknown methods:
- IUnknown::AddRef
- IUnknown::QueryInterface
- IUnknown::Release

**IHXFileExists::DoesExist**

Indicates whether the requested file exists on the file system.

```cpp
STDMETHOD(DoesExist) (  
    THIS_  
    const char* pPath,  
    IHXFileExistsResponse* pFileResponse  
) PURE;
```

*pPath*

Pointer to the path of the requested file.

*pFileResponse*

Pointer to an IHXFileExistsResponse interface that manages the response to this method.
IHXFileExistsResponse

Purpose: Notifies Helix Universal Server if a given file exists.
Implemented by: Helix Universal Server
Used by: File objects (file system plug-ins)
Header file: hxfiles.h

A file object uses this response interface to IHXFileExists to notify Helix Universal Server that a file does or does not exist on its file system.


The IHXFileExistsResponse interface contains the IHXFileExistsResponse::DoesExistDone method.

As with all Component Object Model (COM) interfaces, the IHXFileExistsResponse interface inherits the following IUnknown methods:
- IUnknown::AddRef
- IUnknown::QueryInterface
- IUnknown::Release

IHXFileExistsResponse::DoesExistDone

Indicates whether a given file does or does not exist in the file system.

STDMETHOD(DoesExistDone) ( 
  THIS_,
  BOOL bExist
) PURE;

bExist
  If TRUE, the file does exist.
IHXFileFormatObject

Purpose: Provides methods of communicating with file format plug-ins.
Implemented by: File format plug-ins
Used by: Helix architecture
Header file: hxformt.h

This is the main interface Helix Universal Server or the Helix client uses to communicate with a file format plug-in. At system startup, Helix uses this interface to return functional information about each file format plug-in. Based on this information, it determines which plug-in it needs to use to stream a request for a specific file type. Additional methods instruct the plug-in to send stream header information or stream packets.

For More Information: See IHXFormatResponse in Volume 2, on page 166. See also “File Format Plug-ins” in Volume 1, on page 56.

The IHXFileFormatObject interface contains the following methods:
• IHXFileFormatObject::Close
• IHXFileFormatObject::GetFileFormatInfo
• IHXFileFormatObject::GetFileHeader
• IHXFileFormatObject::GetPacket
• IHXFileFormatObject::GetStreamHeader
• IHXFileFormatObject::InitFileFormat
• IHXFileFormatObject::Seek

As with all Component Object Model (COM) interfaces, the IHXFileFormatObject interface inherits the following IUnknown methods:
• IUnknown::AddRef
• IUnknown::QueryInterface
• IUnknown::Release

IHXFileFormatObject::Close

Manages any cleanup required before closing the plug-in. All references to objects should be released and memory deallocated. This method is called when the playback is finished or stopped.

STDMETHOD(Close) (THIS)

IHXFileFormatObject::GetFileFormatInfo

Returns crucial information required to associate this plug-in with a given MIME type. This information tells the core which file format plug-in to use for a particular URL. The method is called when the Helix Architecture (HX) core application is started.
STDMETHOD(GetFileFormatInfo) (  
    THIS_,  
    REF(const char**) pFileMimeTypes,  
    REF(const char**) pFileExtensions,  
    REF(const char**) pFileOpenNames  
) PURE;

pFileMimeTypes  
    Returns a pointer to the file MIME types supported by this plug-in.

pFileExtensions  
    Returns a pointer to the file extensions supported by this plug-in.

pFileOpenNames  
    Returns a pointer to the file open names supported by this plug-in.

IHXFileFormatObject::GetFileHeader  
    Returns the file header information. This method is called after the file has been initialized.

STDMETHOD(GetFileHeader) (  
    THIS  
) PURE;

IHXFileFormatObject::GetPacket  
    Returns the packet data for a particular stream. This method is called each time the server or client core needs another packet.

STDMETHOD(GetPacket) (  
    THIS_,  
    UINT16 unStreamNumber  
) PURE;

unStreamNumber  
    Number of the stream from which to get the packet.

IHXFileFormatObject::GetStreamHeader  
    Returns the stream header information for a particular stream. This method is called (after the file header has been read) for each stream in the file format.

STDMETHOD(GetStreamHeader) (  
    THIS_,  
    UINT16 unStreamNumber  
) PURE;

unStreamNumber  
    Number of the stream from which to get the stream header.

IHXFileFormatObject::InitFileFormat  
    Initializes the file. This method is called whenever a URL associated with this plug-in is opened.
STDMETHOD(InitFileFormat) ( 
    THIS_
    IHXRequest* pRequest,
    IHXFormatResponse* pFormatResponse,
    IHXFileObject* pFileObject
) PURE;

pRequest
    Pointer to an IHXRequest interface that manages the request.

pFormatResponse
    Pointer to an IHXFormatResponse interface that manages the response to various methods in this interface.

IHXFileObject
    Pointer to an IHXFileObject interface that manages the file object for the mount point of the request.

IHXFileFormatObject::Seek

Places the file at a specified position.

STDMETHOD(Seek) ( 
    THIS_
    ULONG32 ulOffset
) PURE;

ulOffset
    The offset, in milliseconds, from the beginning of the presentation to which to position the file.
**IHXFileMimeMapper**

Purpose: Determines a file’s MIME type.

Implemented by: File objects (file system plug-ins)

Used by: Helix Universal Server

Header file: hxfiles.h

When it initializes a file object, Helix Universal Server calls this interface’s method to determine the MIME type of the requested file. If the file object does not implement this interface or cannot determine the file MIME type, Helix Universal Server determines the MIME type from the file extension. The file object responds through IHXFileMimeMapperResponse.

**For More Information:** See “File System Plug-ins” in Volume 1, on page 50. For broadcast objects, see IHXBroadcastMapper in Volume 2, on page 63.

The IHXFileMimeMapper interface contains the IHXFileMimeMapper::FindMimeType method.

As with all Component Object Model (COM) interfaces, the IHXFileMimeMapper interface inherits the following IUnknown methods:

- IUnknown::AddRef
- IUnknown::QueryInterface
- IUnknown::Release

**IHXFileMimeMapper::FindMimeType**

Determines the MIME type of the requested file.

```cpp
STDMETHOD(FindMimeType) ( 
    THIS_ 
    const char* pURL, 
    IHXFileMimeMapperResponse* pMimeMapperResponse 
) PURE;
```

**pURL**

The URL from which MIME information is being requested.

**pMimeMapperResponse**

Pointer to an IHXFileMimeMapperResponse interface the plug-in uses to respond to this method.
A file object uses this response interface to IHXFileMimeMapper to notify Helix Universal Server of a requested file's MIME type. With this information, Helix Universal Server determines what file format plug-in to use to stream the file.


The IHXFileMimeMapperResponse interface contains the IHXFileMimeMapperResponse::MimeTypeFound method.

\[\text{• As with all Component Object Model (COM) interfaces, the IHXFileMimeMapperResponse interface inherits the following } \text{IUnknown methods:}\]

\[\text{• IUnknown::AddRef}\]
\[\text{• IUnknown::QueryInterface}\]
\[\text{• IUnknown::Release}\]

**IHXFileMimeMapperResponse::MimeTypeFound**

Indicates whether the MIME type was found by the IHXFileMimeMapper::FindMimeType method. This method is called by the IHXFileObject interface when the initialization of the file is complete and the MIME type is available for the request file. If the file is not valid for the file system, the status **HXR_FAILED** should be returned, with a MIME type of NULL. If the file is valid but the MIME type is unknown, then the status **HXR_OK** should be returned with a MIME type of NULL.

\[\text{STDMETHOD(MimeTypeFound)} (\]
\[\text{ THIS_}\]
\[\text{ HX_RESULT status,}\]
\[\text{ const char* pMimeType}\]
\[\text{ ) PURE;}\]

**status**

The status of the IHXFileMimeMapper::FindMimeType operation. A value of **HXR_OK** indicates that the operation was completed successfully.

**pMimeType**

Pointer to the MIME type for the requested file.
IHXFileMove

Purpose: Provides a means of moving files.
Implemented by: File objects (file system plug-ins)
Used by: Any component with access to file objects
Header file: hxfiles.h

This interface is queried from an object that supports the IHXFileObject interface. Call the IHXFileMove::Move method to move the file to a different location in the file system.

Note: Not all file system plug-ins implement this feature.

The IHXFileMove interface contains the IHXFileMove::Move method.
As with all Component Object Model (COM) interfaces, the IHXFileMove interface inherits the following IUnknown methods:
• IUnknown::AddRef
• IUnknown::QueryInterface
• IUnknown::Release

IHXFileMove::Move

Moves a file to a different location in the file system.

STDMETHOD(Move) ( const char* pNewFilePathName ) PURE;

pNewFilePathName
Pointer to the new location for the file.
IHXFileObject

Purpose: Controls file objects.
Implemented by: File objects (file system plug-ins)
Used by: Helix Universal Server and file format plug-ins
Header file: hxfiles.h

This is the main interface to file objects created by file system plug-ins. Helix Universal Server uses this interface to initialize the file object. It then passes the object to a file format plug-in that uses the object to retrieve file data for streaming. The response interface implemented by the file format plug-in is IHXFileResponse.

For More Information: See “Chapter 4: File Handling” beginning in Volume 1, on page 49.

The IHXFileObject interface contains the following methods:
  • IHXFileObject::Advise
  • IHXFileObject::Close
  • IHXFileObject::GetFilename
  • IHXFileObject::Init
  • IHXFileObject::Read
  • IHXFileObject::Seek
  • IHXFileObject::Write

As with all Component Object Model (COM) interfaces, the IHXFileObject interface inherits the following IUnknown methods:
  • IUnknown::AddRef
  • IUnknown::QueryInterface
  • IUnknown::Release

IHXFileObject::Advise

Passes information to the file object advising it how that file object is going to be used. The file object will respond with HXR_OK if that usage is acceptable, otherwise it returns HXR_PREFER_LINEAR to prevent random access.

STDMETHOD(Advise) ( 
    THIS_ 
    ULONG32 ulInfo 
) PURE;

ulInfo
  Indicates the supported usage types. Currently, the only acceptable parameter is the following:
  HX_FILEADVISE_RANDOMACCESS

IHXFileObject::Close

Closes the file resource and releases all resources associated with the object.
STDMETHOD(Close) ( 
    THIS  
) PURE;

IHXFileObject::GetFilename

Returns the file name (without any path information) associated with a specified file object.

STDMETHOD(GetFilename) ( 
    THIS_  
    REF(const char*) pFilename  
) PURE;

pFilename
Returns a pointer to the file name.

Note: The returned file name should be copied immediately if it is to be used at a later time.

IHXFileObject::Init

Associates a file object interface with the file response interface it should notify of operation completeness. This method should also check for validity of the object (for example, by opening it if it is a local file).

STDMETHOD(Init) ( 
    THIS_  
    ULONG32 ulFlags,  
    IHXFileResponse* pFileResponse  
) PURE;

ulFlags
The file type, which can be any of the following:

- HX_FILE_BINARY
- HX_FILE_NOTRUNC
- HX_FILE_READ
- HX_FILE_WRITE

pFileResponse
Pointer to an IHXFileResponse interface that manages the response to various methods in this interface.

IHXFileObject::Read

Reads a buffer of data of the specified length from the file and asynchronously returns it to the caller using the IHXFileResponse interface passed in to IHXFileObject::Init.

STDMETHOD(Read) ( 
    THIS_  
    ULONG32 ulCount  
) PURE;
ulCount
The length of the data buffer to read.

IHXFileObject::Seek

Seeks to an offset in the file and asynchronously notifies the caller that the operation was completed using the IHXFileResponse interface passed in to IHXFileObject::Init.

STDMETHOD(Seek) (  
    THIS_  
    ULONG32 ulOffset,  
    BOOL bRelative  
) PURE;

ulOffset
The offset in the file to which to seek.

bRelative
A return value of TRUE indicates that this is a relative seek. Otherwise, this is an absolute seek.

IHXFileObject::Write

Writes a buffer of data to the file and asynchronously notifies the caller that the operation was completed using the IHXFileResponse interface passed in to IHXFileObject::Init.

STDMETHOD(Write) (  
    THIS_  
    IHXBuffer* pBuffer  
) PURE;

pBuffer
Pointer to an IHXBuffer interface that manages the data to be written to the file.
IHXFileRemove

**Purpose:** Provides for the deletion of files.

**Implemented by:** File objects (file system plug-ins)

**Used by:** Any component with access to file objects

**Header file:** hxfiles.h

This interface is queried from an object that supports the IHXFileObject interface. Call the IHXFileRemove::Remove method to delete the file from the file system.

**Note:** Not all file system plug-ins implement this feature.

The IHXFileRemove interface contains the IHXFileRemove::Remove method.

As with all Component Object Model (COM) interfaces, the IHXFileRemove interface inherits the following IUnknown methods:

- IUnknown::AddRef
- IUnknown::QueryInterface
- IUnknown::Release

**IHXFileRemove::Remove**

Removes a file from the file system.

```cpp
STDMETHOD(Remove) (  
    THIS  
    ) PURE;
```
IHXFileRename

Purpose: Provides for the renaming of files.
Implemented by: File objects (file system plug-ins)
Used by: Any component with access to file objects
Header file: hxfiles.h

This interface is queried from an object that supports the IHXFileObject interface. Call the IHXFileRename::Rename method to rename the file in the file system.

Note: Not all file system plug-ins implement this feature.

The IHXFileRename interface contains the IHXFileRename::Rename method.

As with all Component Object Model (COM) interfaces, the IHXFileRename interface inherits the following IUnknown methods:

• IUnknown::AddRef
• IUnknown::QueryInterface
• IUnknown::Release

IHXFileRename::Rename

Renames the specified file.

STDMETHOD(Rename) (THIS_ const char* pNewFileName) PURE;

pNewFileName
Pointer to the new file name.
IHXFileResponse

<table>
<thead>
<tr>
<th>Purpose:</th>
<th>Informs file format plug-ins of completed file operations.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Implemented by:</td>
<td>File format plug-ins</td>
</tr>
<tr>
<td>Used by:</td>
<td>File objects (file system plug-ins)</td>
</tr>
<tr>
<td>Header file:</td>
<td>hxfiles.h</td>
</tr>
</tbody>
</table>

This is the response interface to IHXFileObject, the main interface to file objects created by a file system plug-in. A file format plug-in uses IHXFileObject to seek for and read file data on a file system. The file format plug-in implements the IHXFileResponse interface to receive asynchronous notification that the file object has finished a file operation.

**For More Information:** See “Chapter 4: File Handling” beginning in Volume 1, on page 49.

The IHXFileResponse interface contains the following methods:

- IHXFileResponse::CloseDone
- IHXFileResponse::InitDone
- IHXFileResponse::ReadDone
- IHXFileResponse::SeekDone
- IHXFileResponse::WriteDone

As with all Component Object Model (COM) interfaces, the IHXFileResponse interface inherits the following IUnknown methods:

- IUnknown::AddRef
- IUnknown::QueryInterface
- IUnknown::Release

IHXFileResponse::CloseDone

Indicates that the IHXFileObject::Close method has finished.

```cpp
STDMETHOD(CloseDone) (
    THIS_
    HX_RESULT status
) PURE;

status
    The status of the IHXFileObject::Close operation. A value of HXR_OK indicates that the operation was completed successfully.
```

IHXFileResponse::InitDone

Indicates that the IHXFileObject::Init method has finished. If the file is not valid for the file system, return a status of HXR_FAILED.

```cpp
STDMETHOD(InitDone) (
    THIS_
    HX_RESULT status
) PURE;
```
status
The status of the IHXFileObject::Init operation. A value of HXR_OK indicates that the operation was completed successfully.

IHXFileResponse::ReadDone
Indicates that the last read from the file was completed and that a buffer is available.

STDMETHOD(ReadDone) (
    THIS_
    HX_RESULT status,
    IHXBuffer* pBuffer
) PURE;

status
The status of the IHXFileObject::Read operation. A value of HXR_OK indicates that the operation was completed successfully.

pBuffer
Pointer to the IHXBuffer interface that manages the data that was read from the file.

IHXFileResponse::SeekDone
Indicates that the last seek in the file was completed.

STDMETHOD(SeekDone) (
    THIS_
    HX_RESULT status
) PURE;

status
The status of the IHXFileObject::Seek operation. A value of HXR_OK indicates that the operation was completed successfully.

IHXFileResponse::WriteDone
Indicates that the last write to the file was completed.

STDMETHOD(WriteDone) (
    THIS_
    HX_RESULT status
) PURE;

status
The status of the IHXFileObject::Write operation. A value of HXR_OK indicates that the operation was completed successfully.
IHXFileStat

Purpose: Requests file statistics from the file system plug-ins.

Implemented by: File objects (file system plug-ins)

Used by: File format plug-ins

Header file: hxfiles.h

A file format plug-in uses this interface to request file statistics, such as file size and creation time, from a file object. The file object then uses the response interface IHXFileStatResponse to send the asynchronous response data to the file format plug-in.


The IHXFileStat interface contains the IHXFileStat::Stat method.

As with all Component Object Model (COM) interfaces, the IHXFileStat interface inherits the following IUnknown methods:

• IUnknown::AddRef
• IUnknown::QueryInterface
• IUnknown::QueryInterface

IHXFileStat::Stat

Requests the file statistics.

```cpp
STDMETHOD(Stat) (THIS_IHXFileStatResponse* pFileStatResponse) PURE;
```

pFileStatResponse

Pointer to an IHXFileStatResponse interface that manages the file statistics.
**IHXFileStatResponse**

Purpose: Sends file statistics to file format plug-ins.

Implemented by: File format plug-ins

Used by: File objects (file system plug-ins)

Header file: hxfiles.h

This is the response interface to IHXFileStat, which the file format plug-in can use to request file statistics, such as file size and creation time, from a file object. The file format plug-in implements IHXFileStatResponse to receive the asynchronous response data from the file object.

**For More Information:** See “File Format Plug-ins” in Volume 1, on page 56.

The IHXFileStatResponse interface contains the IHXFileStatResponse::StatDone method.

As with all Component Object Model (COM) interfaces, the IHXFileStatResponse interface inherits the following IUnknown methods:

- IUnknown::AddRef
- IUnknown::QueryInterface
- IUnknown::Release

**IHXFileStatResponse::StatDone**

Returns the file status information requested by the IHXFileStat::Stat method.

```c
STDMETHOD(StatDone) (  
  THIS_
  HX_RESULT status,
  UINT32 ulSize,
  UINT32 ulCreationTime,
  UINT32 ulAccessTime,
  UINT32 ulModificationTime,
  UINT32 ulMode
) PURE;
```

**status**

The status of the IHXFileStat::Stat operation. A value of HXR_OK indicates that the operation was completed successfully.

**ulSize**

The size of the file, in bytes.

**ulCreationTime**

The time at which the file was created. This parameter uses the number of seconds that have elapsed since January 1, 1970.

**ulAccessTime**

The last time the file was accessed by either a read or a write operation. This parameter uses the number of seconds that have elapsed since January 1, 1970.
ulModificationTime
The last time the file was modified with new data. This parameter uses the number of seconds that
have elapsed since January 1, 1970.

ulMode
Gives information about the file type, which can be any of the following:

- HX_S_IFMT
  A Mount file type.
- HX_S_IFDIR
  A Directory file type.
- HX_S_IFCHR
  A Character Device file type.
- HX_S_IFIFO
  A Named Pipe file type.
- HX_S_IFREG
  A Regular File file type.
IHXFileSystemManager

Purpose: Gets file objects for relative files.
Implemented by: File system manager objects (Helix architecture)
Used by: File format plug-ins, Helix Universal Server
Header file: hxfiles.h

A file format plug-in can create a file system manager object and use this interface if it needs to create additional file objects for a file, or access a different file or set of files residing in locations relative to a requested file. The file system manager object requests new file objects from the original file object through IHXGetFileFromSamePool. It passes the file format plug-in pointers to the relative file objects through IHXFileSystemManagerResponse.

For More Information: See “Chapter 4: File Handling” beginning in Volume 1, on page 49. See also IHXDirHandler.

The IHXFileSystemManager interface contains the following methods:
- IHXFileSystemManager::GetDirObjectFromURL
- IHXFileSystemManager::GetFileObject
- IHXFileSystemManager::GetNewFileObject
- IHXFileSystemManager::GetRelativeFileObject
- IHXFileSystemManager::Init

As with all Component Object Model (COM) interfaces, the IHXFileSystemManager interface inherits the following IUUnknown methods:
- IUUnknown::AddRef
- IUUnknown::QueryInterface
- IUUnknown::Release

IHXFileSystemManager::GetDirObjectFromURL

This method is no longer used and should return HXR_NOTIMPL.

STDMETHOD(GetDirObjectFromURL) (THIS_ const char* pURL) PURE;

IHXFileSystemManager::GetFileObject

Attempts to locate an existing file by calling the IHXFileExists::DoesExist method created from each file system, and returns that object through IHXFileSystemManagerResponse::FileObjectReady.

STDMETHOD(GetFileObject) (THIS_ IHXRequest* pRequest,
IHXAuthenticator* pAuthenticator) PURE;
pRequest
  Pointer to an IHXRequest interface that manages the request for the file object you want.

pAuthenticator
  This parameter is obsolete and is always set to NULL.

IHXFileSystemManager::GetNewFileObject

Attempts to create a new file. The first file system that matches the mount point or protocol for the path in the request object creates the file which is then returned through IHXFileSystemManagerResponse::FileObjectReady. This is especially useful if you want to open a brand new file for writing. This method is similar to IHXFileSystemManager::GetFileObject except that no IHXFileExists::DoesExist checks are performed.

STDMETHOD(GetNewFileObject) (  
  THIS_  
  IHXRequest* pRequest,
  IHXAuthenticator* pAuthenticator
) PURE;

pRequest
  Pointer to an IHXRequest interface that manages the request for the file object you want.

pAuthenticator
  This parameter is obsolete and is always set to NULL.

IHXFileSystemManager::GetRelativeFileObject

Gets a file object off the same mount point as the current file system.

STDMETHOD(GetRelativeFileObject) (  
  THIS_  
  IUnknown* pOriginalObject,
  const char* pPath
) PURE;

pOriginalObject
  Pointer to the IUnknown interface that identifies the instance of the origin file object (IHXFileObject).

pPath
  Pointer to the requested URL.

IHXFileSystemManager::Init

Associates a file system manager response interface to this file system manager interface.

STDMETHOD(Init) (  
  THIS_  
  IHXFileSystemManagerResponse* pFileManagerResponse
) PURE;
pFileManagerResponse
   Pointer to an IHXFileSystemManagerResponse interface that manages the response to various methods in this interface.
IHXFileSystemManagerResponse

Purpose: Receives pointers to relative file objects.

Implemented by: File format plug-ins, Helix Universal Server

Used by: File system manager objects (Helix architecture)

Header file: hxfiles.h

This is the response interface to IHXFileSystemManager, which a file format plug-in uses to get relative file objects. The plug-in implements IHXFileSystemManagerResponse to receive pointers to the relative file objects.


The IHXFileSystemManagerResponse interface contains the following methods:

- IHXFileSystemManagerResponse::DirObjectReady
- IHXFileSystemManagerResponse::FileObjectReady
- IHXFileSystemManagerResponse::InitDone

As with all Component Object Model (COM) interfaces, the IHXFileSystemManagerResponse interface inherits the following IUnknown methods:

- IUnknown::AddRef
- IUnknown::QueryInterface
- IUnknown::Release

IHXFileSystemManagerResponse::DirObjectReady

This method is no longer used and should return HXR_NOTIMPL.

STDMETHOD(DirObjectReady) ( 
   THIS_ 
   HX_RESULT status, 
   IUnknown* pDirObject 
) PURE;

IHXFileSystemManagerResponse::FileObjectReady

Indicates that the IHXFileSystemManager::GetFileObject, IHXFileSystemManager::GetNewFileObject, or IHXFileSystemManager::GetRelativeFileObject method has finished.

STDMETHOD(FileObjectReady) ( 
   THIS_ 
   HX_RESULT status, 
   IUnknown* pObject 
) PURE;

status

The status of the operation. A value of HXR_OK indicates that the operation was completed successfully.
pObject
     Pointer to an IUnknown interface that identifies the instance of the file object.

IHXFileSystemManagerResponse::InitDone
     Indicates that the IHXFileSystemManager::Init method has finished.
     STDMETHOD(InitDone) (   
      THIS_   
      HX_RESULT status   
     ) PURE;

status
     The status of the IHXFileSystemManager::Init operation. A value of HXR_OK indicates that the
     operation was completed successfully.
IHXFileSystemObject

Purpose: Controls file system plug-ins.
Implemented by: File system plug-ins
Used by: Helix Universal Server
Header file: hxfiles.h

All file system plug-ins must implement this interface. Helix Universal Server uses this interface to initialize the plug-ins. It calls the IHXFileSystemObject::CreateFile method to create file objects accessed through IHXFileObject.

For More Information: See “Chapter 4: File Handling” beginning in Volume 1, on page 49.

The IHXFileSystemObject interface contains the following methods:

- IHXFileSystemObject::CreateDir
- IHXFileSystemObject::CreateFile
- IHXFileSystemObject::GetFileSystemInfo
- IHXFileSystemObject::InitFileSystem

As with all Component Object Model (COM) interfaces, the IHXFileSystemObject interface inherits the following IUnknown methods:

- IUnknown::AddRef
- IUnknown::QueryInterface
- IUnknown::Release

IHXFileSystemObject::CreateDir

This method is no longer used and should return HXR_NOTIMPL.

STDMETHOD(CreateDir) (THIS_ IUnknown** ppDirObject ) PURE;

IHXFileSystemObject::CreateFile

Creates a new file object that manages all of the file I/O functionality of this class. This file object is eventually handed off to a file format plug-in that manages file I/O through this file object. This method is called when a URL with a protocol associated with this plug-in is opened.

STDMETHOD(CreateFile) (THIS_ IUnknown** ppFileObject ) PURE;

ppFileObject

Returns a pointer to the IUnknown interface that identifies the instance of the file object that was created.
**IHXFileSystemObject::GetFileSystemInfo**

Returns crucial information required to associate this plug-in with a given protocol. This information tells the server or client core which file system plug-in to use for a particular protocol.

```cpp
STDMETHOD(GetFileSystemInfo) (THIS_
    REF(const char*) pShortName,
    REF(const char*) pProtocol
) PURE;
```

- **pShortName**
  Returns a pointer to the short name. This parameter should be a short, human-readable name in the form of “company-fsname”—for example, `pShortName = "hx-local"`.

- **pProtocol**
  Returns a pointer to the protocol. For example, in the URL `file://myfile.txt`, the protocol would be “file”.

**IHXFileSystemObject::InitFileSystem**

Performs any additional initialization steps required for the file system. It is called prior to the `IHXFileSystemObject::CreateFile` request. Any options provided usually refer to mounting options related to the server, such as base path or authentication preferences.

```cpp
STDMETHOD(InitFileSystem) (THIS_
    IHXValues* pOptions
) PURE;
```

- **pOptions**
  Pointer to an `IHXValues` interface that manages the configuration options for the file system.
IHXFileViewSource

Purpose: Controls the source shown in the browser.
Implemented by: File format plug-ins
Used by: ViewSource
Header file: hxvsrc.h

This interface controls the source shown in the browser when a user attempts to view the source of a particular implementor's data type. The response interface is IHXFileViewSourceResponse.

The IHXFileViewSource interface contains the following methods:

- IHXFileViewSource::Close
- IHXFileViewSource::GetSource
- IHXFileViewSource::InitViewSource

As with all Component Object Model (COM) interfaces, the IHXFileViewSource interface inherits the following IUnknown methods:

- IUnknown::AddRef
- IUnknown::QueryInterface
- IUnknown::Release

IHXFileViewSource::Close

Closes the view source interface.

STDMETHOD(Close) (THIS) PURE;

IHXFileViewSource::GetSource

Queries for the source to be viewed.

STDMETHOD(GetSource) (THIS) PURE;

IHXFileViewSource::InitViewSource

Provides the information required to view the source information.

STDMETHOD(InitViewSource) (THIS,
  IHXFileObject* pFileObject,
  IHXFileViewSourceResponse* pResp,
  SOURCE_TYPE sourceType,
  IHXValues* pOptions
) PURE;
**pFileObject**
Pointer to an IHXFileObject interface that manages the file object whose source is to be viewed.

**pResp**
Pointer to an IHXFileViewSourceResponse interface that manages the responses to this interface.

**sourceType**
A SOURCE_TYPE enumerator that specifies what type of source is to be viewed. The following file formats are currently supported:

- **HTML_SOURCE**
The output is formatted to be viewed in a browser (currently, this type of source should be supported).

- **RAW_SOURCE**
The output is the raw file.

**pOptions**
Pointer to an IHXValues interface that manages information about the context from which view source was called. This information includes:

- **HidePaths**
  A ULONG32 value that, if FALSE, instructs the implementor of this interface to show the full paths for any embedded links.

- **ViewSourceURL**
  A CString value that shows the URL to use if you want to link to other sources.

- **CurrentPath**
  A CString value that shows the path of the current file. This value is used for creating relative links.

- **RamGenURL**
  A CString value that, if it exists, is a URL that links to a ramgen mount point. This enables a user to click a link in the source and stream the content to his or her player. RamGenURL is available only if HidePaths is turned off.
**IHXFileViewSourceResponse**

Purpose: Returns the source information.
Implemented by: ViewSource
Used by: File format plug-in
Header file: hxvsrc.h

This is the response interface for the IHXFileViewSource interface.
The IHXFileViewSourceResponse interface contains the following methods:
- IHXFileViewSourceResponse::CloseDone
- IHXFileViewSourceResponse::InitDone
- IHXFileViewSourceResponse::SourceReady

As with all Component Object Model (COM) interfaces, the IHXFileViewSourceResponse interface inherits the following IUnknown methods:
- IUnknown::AddRef
- IUnknown::QueryInterface
- IUnknown::Release

**IHXFileViewSourceResponse::CloseDone**

Returns the status of the close operation.

```cpp
STDMETHOD(CloseDone) (THIS_ HX_RESULT status) PURE;
```

**status**
The status of the IHXFileViewSource::Close operation. A value of HXR_OK indicates that the operation was completed successfully.

**IHXFileViewSourceResponse::InitDone**

Returns the status of the initialization operation.

```cpp
STDMETHOD(InitDone) (THIS_ HX_RESULT status) PURE;
```

**status**
The status of the IHXFileViewSource::InitViewSource operation. A value of HXR_OK indicates that the operation was completed successfully.

**IHXFileViewSourceResponse::SourceReady**

Indicates the status of the operation and, if successful, returns a pointer to the source information.
STDMETHOD(SourceReady) (  
    THIS_  
    HX_RESULT status,  
    IHXBuffer* pSource  
) PURE;

status  
The status of the IHXFileViewSource::GetSource operation. A value of HXR_OK indicates that the  
operation was completed successfully.

pSource  
Pointer to an IHXBuffer interface that manages the source information.
IHXFocusNavigation

Purpose: Navigates between and within keyboard focus sites.
Implemented by: Display renderer
Used by: Site object (client core)
Header file: hxwin.h

This interface allows programs, such as SMIL, to move the keyboard focus from one site to another. You can then use the IHXKeyboardFocus interface to provide information about the site. In addition, you can also call the client core’s IHXDrawFocus interface to optionally draw a visible highlight around the area that has focus.

The IHXFocusNavigation interface contains the following methods:

• IHXFocusNavigation::ActivateFocus
• IHXFocusNavigation::ClearFocus
• IHXFocusNavigation::GetFocusState
• IHXFocusNavigation::SetFocus

As with all Component Object Model (COM) interfaces, the IHXFocusNavigation interface inherits the following IUnknown methods:

• IUnknown::AddRef
• IUnknown::QueryInterface
• IUnknown::Release

IHXFocusNavigation::ActivateFocus

Activates the focused link. If no focus point exists, nothing is done.

STDMETHOD(ActivateFocus) ( THIS ) PURE;

IHXFocusNavigation::ClearFocus

Clears the current focus.

STDMETHOD(ClearFocus) ( THIS ) PURE;

IHXFocusNavigation::GetFocusState

Obtains the current focus state of the site. Returns either HXNoFocus if the site is not focused or HXFocused if the site is focused.

STDMETHOD_(HXFocusState,GetFocusState) ( THIS ) PURE;
IHXF\text{FocusNavigation}::\text{SetFocus}

Sets the focus to the given item.

\text{STDMETHOD(SetFocus)} (\text{THIS, }\text{H XFFocusContext eFocus}) \text{PURE;}

\textbf{eFocus}

An \text{H XFFocusContext} enumerator that indicates the type of focus. One of the following:

- \text{HXFirstFocus}
  
  Sets the focus to the first element of the renderer’s “group” that can have focus.

- \text{HXUpFocus}
  
  If there is a notion of up, sets the focus to that element.

- \text{HXDownFocus}
  
  If there is a notion of down, sets the focus to that element.

- \text{HXLeftFocus}
  
  If there is a notion of left, sets the focus to that element.

- \text{HXRightFocus}
  
  If there is a notion of right, sets the focus to that element.

- \text{HN extFocus}
  
  Sets the focus to the next element of the renderer's group that can have focus. If it is “next” and focus is at the last element, clear the focus and return an error.

- \text{HXPrevFocus}
  
  Sets the focus to the previous element of the renderer’s group that can have focus. If it is “prev” and focus is at the first element, clear the focus and return an error.

- \text{HXLastFocus}
  
  Sets the focus to the last element of a renderer's group of items that can have focus.
IHXFormatResponse

Purpose: Lets file format plug-ins communicate with Helix.
Implemented by: Helix architecture
Used by: File format and broadcast plug-ins
Header file: hxformt.h

This is the response interface to IHXFileFormatObject and IHXBroadcastFormatObject. During plug-in initialization, Helix Universal Server or the Helix Client (with file format plug-ins only) sets itself up as each plug-in’s response object. Plug-ins then use this interface to pass the system stream information.


The IHXFormatResponse interface contains the following methods:

• IHXFormatResponse::FileHeaderReady
• IHXFormatResponse::InitDone
• IHXFormatResponse::PacketReady
• IHXFormatResponse::SeekDone
• IHXFormatResponse::StreamDone
• IHXFormatResponse::StreamHeaderReady

As with all Component Object Model (COM) interfaces, the IHXFormatResponse interface inherits the following IUnknown methods:

• IUnknown::AddRef
• IUnknown::QueryInterface
• IUnknown::Release

IHXFormatResponse::FileHeaderReady

Indicates whether the request for a file header was completed successfully and, if so, returns the file header.

STDMETHOD(FileHeaderReady) ( 
    THIS_ 
    HX_RESULT status, 
    IHXValues* pHeader 
) PURE;

status
Contains the response status. If this parameter is HXR_OK, the call completed successfully.

pHeader
Pointer to an IHXValues interface that manages the file header information.

IHXFormatResponse::InitDone

Indicates whether the format object initialization was completed successfully.
STDMETHOD(InitDone) (  
    THIS_,  
    HX_RESULT status  
) PURE;

status  
Contains the response status. If this parameter is HXR_OK, the call completed successfully.

IHXFormatResponse::PacketReady  
Indicates whether the request for a packet was completed successfully and, if so, passes the packet back to the server or client core.

STDMETHOD(PacketReady) (  
    THIS_,  
    HX_RESULT status,  
    IHXPacket* pPacket  
) PURE;

status  
Contains the response status. If this parameter is HXR_OK, the call completed successfully.

pPacket  
Pointer to an IHXPacket interface that manages the packet data.

IHXFormatResponse::SeekDone  
Indicates whether the seek to a different part of the time line was completed successfully.

STDMETHOD(SeekDone) (  
    THIS_,  
    HX_RESULT status  
) PURE;

status  
Contains the response status. If this parameter is HXR_OK, the call completed successfully.

IHXFormatResponse::StreamDone  
Indicates that no more packets are available for this stream.

STDMETHOD(StreamDone) (  
    THIS_,  
    UINT16 unStreamNumber  
) PURE;

unStreamNumber  
The stream number of the stream that has completed.

IHXFormatResponse::StreamHeaderReady  
Indicates whether the request for a stream header was completed successfully and, if so, returns the stream header.
STDMETHOD(StreamHeaderReady) (  
    THIS_,  
    HX_RESULT status,  
    IHXValues* pHeader  
) PURE;

status  
    Contains the response status. If this parameter is HXR_OK, the call completed successfully.

pHeader  
    Pointer to an IHXValues interface that manages the stream header information.
IHXGenericPlugin

Purpose: Indicates that IHXPlugin::InitPlugin should be called immediately.
Implemented by: Plug-ins
Used by: Client/server core
Header file: hxplugn.h

A plug-in is marked as generic using this interface if it responds TRUE to the IHXGenericPlugin::IsGeneric method. The plug-in will then receive notification of server or client startup. IHXPlugin::InitPlugin will be called to perform any initialization steps that may be required.

The IHXGenericPlugin interface contains the IHXGenericPlugin::IsGeneric method.

As with all Component Object Model (COM) interfaces, the IHXGenericPlugin interface inherits the following IUnknown methods:

• IUnknown::AddRef
• IUnknown::QueryInterface
• IUnknown::Release

IHXGenericPlugin::IsGeneric

Indicates whether or not the plug-in is generic.

STDMETHOD(IsGeneric) ( 
    THIS_ 
    REFl(BOOL) bIsGeneric 
) PURE;

bIsGeneric

If TRUE, the plug-in is generic and will receive notification of server or client startup. If FALSE, the plug-in is not generic.
IHXGetFileFromSamePool

Purpose: Requests file objects for relative files.
Implemented by: File objects (file system plug-ins)
Used by: File system manager objects (Helix architecture)
Header file: hxfiles.h

When a file format plug-in needs to access a relative file, it creates an IHXFileSystemManager object that uses the IHXGetFileFromSamePool interface on the original file object to request a new file object. The original file object then returns a pointer to the new, uninitialized file object through IHXGetFileFromSamePoolResponse.


The IHXGetFileFromSamePool interface contains the IHXGetFileFromSamePool::GetFileObjectFromPool method.

As with all Component Object Model (COM) interfaces, the IHXGetFileFromSamePool interface inherits the following IUnknown methods:

- IUnknown::AddRef
- IUnknown::QueryInterface
- IUnknown::Release

IHXGetFileFromSamePool::GetFileObjectFromPool

Gets another file object from the same pool.

STDMETHOD(GetFileObjectFromPool) (THIS_
    IHXGetFileFromSamePoolResponse* response
) PURE;

response

Pointer to an IHXGetFileFromSamePoolResponse interface that manages the new, uninitialized file object.
**IHXGetFileFromSamePoolResponse**

Purpose: Returns pointers to relative file objects.

Implemented by: File system manager objects (Helix architecture)

Used by: File objects (file system plug-ins)

Header file: hxfiles.h

This is the response interface to IHXGetFileFromSamePool. A file object uses it to return to an IHXFileSystemManager object a pointer to a new relative file object.

*For More Information:* See “Chapter 4: File Handling” beginning in Volume 1, on page 49.

The IHXGetFileFromSamePoolResponse interface contains the IHXGetFileFromSamePoolResponse::FileObjectReady method.

As with all Component Object Model (COM) interfaces, the IHXGetFileFromSamePoolResponse interface inherits the following IUnknown methods:

- IUnknown::AddRef
- IUnknown::QueryInterface
- IUnknown::Release

**IHXGetFileFromSamePoolResponse::FileObjectReady**

Indicates whether the request for a new file object was completed successfully and returns another file object from the same pool.

```cpp
STDMETHOD(FileObjectReady) ( 
    THIS_ 
    HX_RESULT status, 
    IUnknown* ppUnknown 
) PURE;
```

*status* Contains the response status to the IHXGetFileFromSamePool::GetFileObjectFromPool operation. If this parameter is HXR_OK, the call completed successfully.

*ppUnknown* Pointer to the IUnknown interface that identifies the instance of the file object.
IHXGroup

Purpose: Manipulates tracks in a group.
Implemented by: Player object (Helix client core)
Used by: IHXGroupManager
Header file: hxgroup.h

IHXGroupManager uses this interface to add a track to a group, remove a track from a group, start or get a track in a group, get the number of tracks in a group, and get or set properties of a group.


The IHXGroup interface contains the following methods:

• IHXGroup::AddTrack
• IHXGroup::GetGroupProperties
• IHXGroup::GetTrack
• IHXGroup::GetTrackCount
• IHXGroup::RemoveTrack
• IHXGroup::SetGroupProperties

As with all Component Object Model (COM) interfaces, the IHXGroup interface inherits the following IUnknown methods:

• IUnknown::AddRef
• IUnknown::QueryInterface
• IUnknown::Release

IHXGroup::AddTrack

Adds a track to the group.

STDMETHOD(AddTrack) ( 
    THIS_ 
    IHXValues* pTrack 
    ) PURE;

pTrack
    Pointer to an IHXValues interface that manages the track to add.

IHXGroup::GetGroupProperties

Gets any group-specific information. This information is returned through an IHXValues interface. This method can return NULL.

STDMETHOD_(IHXValues*, GetGroupProperties) ( 
    THIS 
    ) PURE;
**IHXGroup::GetTrack**

Gets the indicated track in this group.

```cpp
STDMETHOD(GetTrack) ( 
    THIS_ 
    UINT16 uTrackIndex, 
    REF(IHXValues*) pTrack 
) PURE;
```

- **uTrackIndex**
  - The index of the track to retrieve.
- **pTrack**
  - Returns a pointer to an IHXValues interface that manages the track.

**IHXGroup::GetTrackCount**

Gets the number of tracks within this group.

```cpp
STDMETHOD_(UINT16,GetTrackCount) ( 
    THIS 
) PURE;
```

**IHXGroup::RemoveTrack**

Removes an already added track.

```cpp
STDMETHOD(RemoveTrack) ( 
    THIS_ 
    UINT16 uTrackIndex 
) PURE;
```

- **uTrackIndex**
  - The track to remove.

**IHXGroup::SetGroupProperties**

Sets any group-specific information, such as title, author, copyright, and so on.

```cpp
STDMETHOD(SetGroupProperties) ( 
    THIS_ 
    IHXValues* pProperties 
) PURE;
```

- **pProperties**
  - Pointer to an IHXValues interface that manages the group-specific information.
IHXGroup2

Purpose: Extended manipulation of tracks in a group.
Implemented by: Player object (Helix client core)
Used by: IHXGroupManager
Header file: hgxgroup.h

This interface is an extension of the IHXGroup interface, from which it is queried (IUnknown::QueryInterface), and is included to provide support for SMIL 2.0.

The IHXGroup2 interface contains the following methods:

- IHXGroup2::AddPrefetchSink
- IHXGroup2::AddPrefetchTrack
- IHXGroup2::AddTrackSink
- IHXGroup2::GetIHXTrack
- IHXGroup2::GetPrefetchTrack
- IHXGroup2::GetPrefetchTrackCount
- IHXGroup2::PrefetchTrackDone
- IHXGroup2::RemovePrefetchSink
- IHXGroup2::RemovePrefetchTrack
- IHXGroup2::RemoveTrackSink

As with all COM interfaces, the IHXGroup2 interface inherits the following IUnknown methods:

- IUnknown::AddRef
- IUnknown::QueryInterface
- IUnknown::Release

IHXGroup2::AddPrefetchSink

Adds an advise sink to monitor the prefetch status.

STDMETHOD(AddPrefetchSink) ( THIS_ IHXPrefetchSink* pSink ) PURE;

pSink
Pointer to an IHXPrefetchSink interface that manages the track’s prefetch status.

IHXGroup2::AddPrefetchTrack

Adds a track to prefetch.

STDMETHOD(AddPrefetchTrack) ( THIS_ IHXValues* pTrack, REF(UINT16) uPrefetchTrackId ) PURE;
pTrack
   Pointer to an IHXValues interface that manages the track to be prefetched.

uPrefetchTrackId
   Returns the identification of the prefetch track to be added.

IHXGroup2::AddTrackSink
   Adds an advise sink to monitor the status of the track.
   STDMETHOD(AddTrackSink) (THIS_ IHXTrackSink* pSink) PURE;
   pSink
      Pointer to an IHXTrackSink interface that manages the track’s status.

IHXGroup2::GetIHXTrack
   Get the specified track in this group.
   STDMETHOD(GetIHXTrack) (THIS_ UINT16 uTrackIndex, REF(IHXTrack*) pTrack) PURE;
   uTrackIndex
      The index of the track to be retrieved.
   pTrack
      Returns a pointer to the IHXTrack interface that manages the track to be retrieved from this group.

IHXGroup2::GetPrefetchTrack
   Gets the prefetched track.
   STDMETHOD(GetPrefetchTrack) (THIS_ UINT16 uPrefetchTrackId, REF(IHXValues*) pTrack) PURE;
   uPrefetchTrackId
      The identification of the track to be prefetched.
   pTrack
      Returns a pointer to the IHXValues interface that manages the specified track to be prefetched.

IHXGroup2::GetPrefetchTrackCount
   Gets the number of prefetch tracks.
STDMETHOD_(UINT16, GetPrefetchTrackCount) (THIS) PURE;

IHXGroup2::PrefetchTrackDone

Indicates the track prefetch has finished.
STDMETHOD(PrefetchTrackDone) (THIS_ UINT16 uPrefetchTrackId, HX_RESULT status) PURE;

uPrefetchTrackId
The identification of the track whose prefetch has finished.

status
The status of the prefetch. A value of HXR_OK indicates the process completed successfully.

IHXGroup2::RemovePrefetchSink

Removes the advise sink that was monitoring the prefetch status.
STDMETHOD(RemovePrefetchSink) (THIS_ IHXPrefetchSink* pSink) PURE;

pSink
Pointer to the IHXPrefetchSink interface to be removed.

IHXGroup2::RemovePrefetchTrack

Removes the prefetched track.
STDMETHOD(RemovePrefetchTrack) (THIS_ UINT16 uPrefetchTrackId) PURE;

uPrefetchTrackId
The identification of the prefetch track to be removed.

IHXGroup2::RemoveTrackSink

Removes the advise sink that was monitoring the track status.
STDMETHOD(RemoveTrackSink) (THIS_ IHXTrackSink* pSink) PURE;

pSink
Pointer to the IHXTrackSink interface to be removed.
IHXGroupManager

Purpose: Manipulates groups and sinks.
Implemented by: IHXPlayer object (Helix client core)
Used by: Top-level client or rendering plug-ins
Header file: hxgroup.h

A top-level client or rendering plug-in uses this interface to add or remove sinks; create or add groups; get, set, or remove groups, or get the number of groups in a presentation.


The IHXGroupManager interface contains the following methods:
- IHXGroupManager::AddGroup
- IHXGroupManager::AddSink
- IHXGroupManager::CreateGroup
- IHXGroupManager::GetCurrentGroup
- IHXGroupManager::GetGroup
- IHXGroupManager::GetGroupCount
- IHXGroupManager::GetPresentationProperties
- IHXGroupManager::RemoveGroup
- IHXGroupManager::RemoveSink
- IHXGroupManager::SetCurrentGroup
- IHXGroupManager::SetPresentationProperties

As with all Component Object Model (COM) interfaces, the IHXGroupManager interface inherits the following IUnknown methods:
- IUnknown::AddRef
- IUnknown::QueryInterface
- IUnknown::Release

IHXGroupManager::AddGroup

Adds a group to the presentation.

STDMETHOD(AddGroup) (  
    THIS_  
    IHXGroup* pGroup  
) PURE;

pGroup
    Pointer to an IHXGroup interface that manages the group to be added.

IHXGroupManager::AddSink

Adds a sink to get notifications about any tracks or groups being added to the presentation.
STDMETHOD(AddSink) (  
    THIS_,  
    IHXGroupSink* pGroupSink  
) PURE;

pGroupSink  
Pointer to an IHXGroupSink interface that manages the notifications.

**IHXGroupManager::CreateGroup**

Creates a group.

STDMETHOD(CreateGroup) (  
    THIS_,  
    REF(IHXGroup*) pGroup  
) PURE;

pGroup  
Returns a pointer to an IHXGroup interface that manages the group.

**IHXGroupManager::GetCurrentGroup**

Gets the current group index.

STDMETHOD(GetCurrentGroup) (  
    THIS_,  
    REF(UINT16) uGroupIndex  
) PURE;

uGroupIndex  
Returns the index of the current group.

**IHXGroupManager::GetGroup**

 Gets the specified group in the presentation.

STDMETHOD(GetGroup) (  
    THIS_,  
    UINT16 uGroupIndex,  
    REF(IHXGroup*) pGroup  
) PURE;

uGroupIndex  
The index of the group to get.

pGroup  
Returns a pointer to an IHXGroup interface that manages the group.

**IHXGroupManager::GetGroupCount**

 Gets the total number of groups within the presentation.

STDMETHOD_(UINT16,GetGroupCount) (  
    THIS  
) PURE;
IHXGroupManager::GetPresentationProperties

Gets any presentation information. This information is returned in the indicated IHXValues interface. This method can return NULL.

```cpp
STDMETHOD_(IHXValues*, GetPresentationProperties) (THIS) PURE;
```

IHXGroupManager::RemoveGroup

Removes the specified group.

```cpp
STDMETHOD(RemoveGroup) (THIS UINT16 uGroupIndex) PURE;
```

- **uGroupIndex**
  The index of the group to remove.

IHXGroupManager::RemoveSink

Removes the specified group sink.

```cpp
STDMETHOD(RemoveSink) (THIS IHXGroupSink* pGroupSink) PURE;
```

- **pGroupSink**
  Pointer to the IHXGroupSink interface to be removed.

IHXGroupManager::setCurrentGroup

Plays the specified group in the presentation.

```cpp
STDMETHOD(SetCurrentGroup) (THIS UINT16 uGroupIndex) PURE;
```

- **uGroupIndex**
  The index of the group to play in the presentation.

IHXGroupManager::SetPresentationProperties

Sets any presentation information, such as title, author, copyright, and so on.

```cpp
STDMETHOD(SetPresentationProperties) (THIS IHXValues* pProperties) PURE;
```
pProperties

Pointer to an IHXValues interface that manages the presentation information.
IHXGroupSink

Purpose: Receives notifications of tracks and groups in a presentation.
Implemented by: Top-level client or rendering plug-ins
Used by: Top-level client
Header file: hxgroup.h

A top-level client can use this interface to receive notifications of tracks added to a group, tracks removed from a group, tracks started or stopped in a group, and groups added to or removed from a presentation.


The IHXGroupSink interface contains the following methods:
• IHXGroupSink::AllGroupsRemoved
• IHXGroupSink::CurrentGroupSet
• IHXGroupSink::GroupAdded
• IHXGroupSink::GroupRemoved
• IHXGroupSink::TrackAdded
• IHXGroupSink::TrackRemoved
• IHXGroupSink::TrackStarted
• IHXGroupSink::TrackStopped

As with all Component Object Model (COM) interfaces, the IHXGroupSink interface inherits the following IUnknown methods:
• IUnknown::AddRef
• IUnknown::QueryInterface
• IUnknown::Release

IHXGroupSink::AllGroupsRemoved

Indicates that all groups have been removed from the current presentation.

STDMETHOD(AllGroupsRemoved) (
    THIS
) PURE;

IHXGroupSink::CurrentGroupSet

Indicates that this group is being currently played in the presentation.

STDMETHOD(CurrentGroupSet) (THIS_
    UINT16 uGroupIndex,
    IHXGroup* pGroup
) PURE;
uGroupIndex
    The index of the current group.

pGroup
    Pointer to an IHXGroup interface that manages the group.

IHXGroupSink::GroupAdded

Indicates that a new group is being added to the presentation.

STDMETHOD(GroupAdded) (  
    THIS_,  
    UINT16 uGroupIndex,  
    IHXGroup* pGroup  
) PURE;

uGroupIndex
    The index of the group being added.

pGroup
    Pointer to an IHXGroup interface that manages the indicated group.

IHXGroupSink::GroupRemoved

Indicates that a group is being removed from the presentation.

STDMETHOD(GroupRemoved) (  
    THIS_,  
    UINT16 uGroupIndex,  
    IHXGroup* pGroup  
) PURE;

uGroupIndex
    The index of the group being removed.

pGroup
    Pointer to an IHXGroup interface that manages the group.

IHXGroupSink::TrackAdded

Indicates that a new track is being added to a group.

STDMETHOD(TrackAdded) (  
    THIS_,  
    UINT16 uGroupIndex,  
    UINT16 uTrackIndex,  
    IHXValues* pTrack  
) PURE;

uGroupIndex
    The index of the group to which the track will be added.

uTrackIndex
    The index of the track to be added.
pTrack
Pointer to an IHXValues interface that manages the track information.

**IHXGroupSink::TrackRemoved**
Indicates that a track is being removed from a group.

```cpp
STDMETHOD(TrackRemoved) (
    THIS_,
    UINT16 uGroupIndex,
    UINT16 uTrackIndex,
    IHXValues* pTrack
) PURE;
```

- **uGroupIndex**
  The index of the group from which the track is being removed.

- **uTrackIndex**
  The index of the track being removed.

- **pTrack**
  Pointer to the IHXValues interface that manages the track.

**IHXGroupSink::TrackStarted**
Indicates that a track is being started (so that you can get the duration, for example).

```cpp
STDMETHOD(TrackStarted) (
    THIS_,
    UINT16 uGroupIndex,
    UINT16 uTrackIndex,
    IHXValues* pTrack
) PURE;
```

- **uGroupIndex**
  The index of the group that contains the track being started.

- **uTrackIndex**
  The index of the track being started.

- **pTrack**
  Pointer to an IHXValues interface that manages the track being started.

**IHXGroupSink::TrackStopped**
Indicates that a track is being stopped.

```cpp
STDMETHOD(TrackStopped) (
    THIS_,
    UINT16 uGroupIndex,
    UINT16 uTrackIndex,
    IHXValues* pTrack
) PURE;
```
**uGroupIndex**
The index of the group that contains the track being stopped.

**uTrackIndex**
The index of the track being stopped.

**pTrack**
Pointer to an IHXValues interface that manages the track being stopped.
IHXGroupSink2

Purpose: Receives notification of group placement in a presentation.
Implemented by: Top-level client or rendering plug-ins
Used by: Top-level client
Header file: hxgroup.h

A top-level client can use this interface to receive notifications of groups being inserted at various points in a presentation, or of a group replacing an existing group in the presentation.

The IHXGroupSink2 interface contains the following methods:

• IHXGroupSink2::GroupInsertedAfter
• IHXGroupSink2::GroupInsertedBefore
• IHXGroupSink2::GroupReplaced

As with all Component Object Model (COM) interfaces, the IHXGroupSink2 interface inherits the following IUnknown methods:

• IUnknown::AddRef
• IUnknown::QueryInterface
• IUnknown::Release

IHXGroupSink2::GroupInsertedAfter

Provides notification of a new group being inserted after the specified group.

STDMETHOD(GroupInsertedAfter) (
    THIS_ 
    UINT16 uAfterGroupIndex,
    IHXGroup* pGroup
) PURE;

uAfterGroupIndex
The index number of the group after which the new group was inserted.

pGroup
Pointer to an IHXGroup interface that manages the new group that was inserted.

IHXGroupSink2::GroupInsertedBefore

Provides notification of a new group being inserted before the specified group.

STDMETHOD(GroupInsertedBefore) (
    THIS_ 
    UINT16 uBeforeGroupIndex,
    IHXGroup* pGroup
) PURE;

uBeforeGroupIndex
The index number of the group before which the new group was inserted.
pGroup
   Pointer to an IHXGroup interface that manages the new group that was inserted.

IHXGroupSink2::GroupReplaced
   Provides notification of a new group replacing the specified group in the current presentation.

   STDMETHOD(GroupReplaced) (THIS_
   UINT16 uGroupIndex,
   IHXGroup* pGroup
   ) PURE;

   uGroupIndex
   The index number of the group that was replaced.

   pGroup
   Pointer to an IHXGroup interface that manages the new group.
IHXGUIDDBManager

Purpose: Provides storage of player GUID data for player authentication.
Implemented by: Database plug-in
Used by: Database user
Header file: hxdb.h

This interface is used by the server core to retrieve information from the pay-per-view GUID database. The response interface is IHXGUIDDBManagerResponse.

The IHXGUIDDBManager interface contains the following methods:

• IHXGUIDDBManager::GetPrincipalIDFromGUID
• IHXGUIDDBManager::SetGUIDForPrincipalID

As with all Component Object Model (COM) interfaces, the IHXGUIDDBManager interface inherits the following IUnknown methods:

• IUnknown::AddRef
• IUnknown::QueryInterface
• IUnknown::Release

IHXGUIDDBManager::GetPrincipalIDFromGUID

Gets the associated player GUID from a user. If the GUID is set to all 0's (zeros) because GUID logging has been disabled by either the client or the server, IHXGUIDDBManagerResponse::GetPrincipalIDFromGUIDDone will not return a valid user ID.

STDMETHOD(GetPrincipalIDFromGUID) (THIS_ IHXGUIDDBManagerResponse* pGUIDDBManagerResponseNew, IHXBuffer* pBufferGUID) PURE;

pGUIDDBManagerResponseNew
Pointer to an IHXGUIDDBManagerResponse interface that manages the response to this method.

pBufferGUID
Pointer to an IHXBuffer interface that manages the player GUID.

IHXGUIDDBManager::SetGUIDForPrincipalID

Associates a player GUID with a user. If GUID logging has been disabled by the server or the client, the player GUID will be set to all 0's (zeros).

STDMETHOD(SetGUIDForPrincipalID) (THIS_ IHXGUIDDBManagerResponse* pGUIDDBManagerResponseNew, IHXBuffer* pBufferPrincipalID, IHXBuffer* pBufferGUID) PURE;
pGUIDDBManagerResponseNew
   Pointer to an IHXGUIDDBManagerResponse interface that manages the response to this method.

pBufferPrincipalID
   Pointer to an IHXBuffer interface that manages the ID of the user.

pBufferGUID
   Pointer to an IHXBuffer interface that manages the player GUID.
IHXGUIDDBManagerResponse

Purpose: Manages player GUID data.
Implemented by: Database user
Used by: Database plug-in
Header file: hxdb.h

This interface provides the GUID data from the database.
The IHXGUIDDBManagerResponse interface contains the following methods:
  • IHXGUIDDBManagerResponse::GetPrincipalIDFromGUIDDone
  • IHXGUIDDBManagerResponse::SetGUIDForPrincipalIDDone

As with all Component Object Model (COM) interfaces, the IHXGUIDDBManagerResponse interface
inherits the following IUnknown methods:
  • IUnknown::AddRef
  • IUnknown::QueryInterface
  • IUnknown::Release

IHXGUIDDBManagerResponse::GetPrincipalIDFromGUIDDone

Returns the user ID associated with the indicated player GUID.

STDMETHOD(GetPrincipalIDFromGUIDDone) (  
  THIS_
  HX_RESULT ResultStatus,
  IHXBuffer* pBufferGUID,
  IHXBuffer* pBufferPrincipalID
) PURE;

ResultStatus
  The status of the IHXGUIDDBManager::GetPrincipalIDFromGUID operation. A value of HXR_OK indicates
  that the operation was completed successfully.

pBufferGUID
  Pointer to an IHXBuffer interface that manages the player GUID.

pBufferPrincipalID
  Pointer to an IHXBuffer interface that manages the user ID.

IHXGUIDDBManagerResponse::SetGUIDForPrincipalIDDone

Reports the status of the call to IHXGUIDDBManager::SetGUIDForPrincipalID.

STDMETHOD(SetGUIDForPrincipalIDDone) (  
  THIS_
  HX_RESULT ResultStatus,
  IHXBuffer* pBufferPrincipalID
) PURE;
ResultStatus
The status of the IHXGUIDDBManager::SetGUIDForPrincipalID operation. A value of HXR_OK indicates that the operation was completed successfully.

pBufferPrincipalID
Pointer to an IHXBuffer interface that manages the user ID.
IHXHyperNavigate

Purpose: Opens a specified URL in the browser.
Implemented by: Helix client
Used by: Rendering plug-ins
Header file: hxhyper.h

A rendering plug-in can use this interface to display a URL at a specified time in the stream. When the plug-in issues a hypernavigation request, the default Web browser opens. If the browser is open already, the target URL displays in the current window. The plug-in can also specify that the URL display in a specific frame of the current browser window through the IHXHyperNavigate::GoToURL method.

For More Information: See “Hypernavigating” in Volume 1, on page 166.

The IHXHyperNavigate interface contains the IHXHyperNavigate::GoToURL method.
As with all Component Object Model (COM) interfaces, the IHXHyperNavigate interface inherits the following IUnknown methods:
• IUnknown::AddRef
• IUnknown::QueryInterface
• IUnknown::Release

IHXHyperNavigate::GoToURL

Opens the default Web browser and displays the target URL.

STDMETHOD(GoToURL) (  
    THIS_  
    const char* pURL,  
    const char* pTarget  
) PURE;

pURL
Pointer to a fully qualified URL, such as http://www.real.com.

pTarget
Pointer to the target frame. To not use a frame, set this parameter to NULL. Possible values include:
• _new or _blank for a new window
• _self for the last active window or frame
• _top for the last active window.
IHXInfoLogger

Purpose: Sends any logging information back to the server.
Implemented by: Helix client
Used by: Rendering plug-ins
Header file: hxcore.h

This interface is used primarily by renderers for a given source to send statistical information back to the server. The information sent back to the server appears in the server’s access log. You can query (IUnknown::QueryInterface) this interface from IHXStreamSource.

The IHXInfoLogger interface contains the IHXInfoLogger::LogInformation method.

As with all Component Object Model (COM) interfaces, the IHXInfoLogger interface inherits the following IUnknown methods:

- IUnknown::AddRef
- IUnknown::QueryInterface
- IUnknown::Release

IHXInfoLogger::LogInformation

Logs any user-defined information in the form of an action and associated data.

STDMETHOD(LogInformation) (THIS_ const char* pAction, const char* pData)

pAction
The action associated with this log (such as seek, pause, play, stop, and so on) or a user-defined action.

pData
Data associated with the specified action (such as time for seek, pause, play, stop, and so on) or user-defined data.
IHXInterruptSafe

Purpose: Notifies Helix of support for interrupt time execution.
Implemented by: Renderers, callbacks
Used by: Helix client
Header file: hxengin.h

A component such as a plug-in can implement this optional interface to notify Helix that it is safe for Helix to call into this plug-in from multiple threads. If this interface is not implemented, the default is for there to be no support of calling on multiple threads.

For More Information: For restrictions on what can be executed at interrupt time, see Apple’s Deferred Task Manager notes.

The IHXInterruptSafe interface contains the IHXInterruptSafe::IsInterruptSafe method.

As with all Component Object Model (COM) interfaces, the IHXInterruptSafe interface inherits the following IUnknown methods:

- IUnknown::AddRef
- IUnknown::QueryInterface
- IUnknown::Release

IHXInterruptSafe::IsInterruptSafe

Determines whether interrupt time execution is supported. The component returns TRUE if interrupt time execution is supported. Returns FALSE if interrupt time execution is not supported.

STDMETHOD_(BOOL,IsInterruptSafe) ( THIS
)
) PURE;
IHXInterruptState

Purpose: Notifies components when entering or leaving execution on the main application thread (the interrupt task on the Macintosh).

Implemented by: Helix client

Used by: Helix core, plug-in components

Header file: hxengine.h

A component such as a plug-in can call this interface to determine whether the Helix core is at interrupt time, or is executing on a thread other than the main application thread. Top-level clients can also use this interface for the client core to always execute on the main application thread.

For More Information: For more information about interrupt tasks, see Apple’s Deferred Task Manager notes.

The IHXInterruptState interface contains the following methods:

• IHXInterruptState::AtInterruptTime
• IHXInterruptState::EnableInterrupt
• IHXInterruptState::EnterInterruptState
• IHXInterruptState::IsInterruptEnabled
• IHXInterruptState::LeaveInterruptState

As with all Component Object Model (COM) interfaces, the IHXInterruptState interface inherits the following IUnknown methods:

• IUnknown::AddRef
• IUnknown::QueryInterface
• IUnknown::Release

IHXInterruptState::AtInterruptTime

Determines whether the core is currently at interrupt task time. This method returns TRUE if it is interrupt task time.

STDMETHOD_(BOOL,AtInterruptTime) (THIS)

IHXInterruptState::EnableInterrupt

Enables or disables interrupt time processing. Works only if all players are stopped.

STDMETHOD(EnableInterrupt) (THIS_ BOOL bEnable)

bEnable

If TRUE, this method enables interrupt time processing. If FALSE, it disables interrupt time processing.
**Note:** On the Macintosh, interrupt time processing is always on.

**IHXInterruptState::EnterInterruptState**

Determines whether a deferred/interrupt task is being started. Functions only on the Macintosh.

```cpp
STDMETHOD(EnterInterruptState) (THIS) PURE;
```

**IHXInterruptState::IsInterruptEnabled**

Determines whether the core is currently interrupt enabled. If this method returns TRUE, the core is currently interrupt enabled.

```cpp
STDMETHOD_(BOOL, IsInterruptEnabled) (THIS) PURE;
```

**IHXInterruptState::LeaveInterruptState**

Determines whether a deferred/interrupt task is being left. Functions only on the Macintosh.

```cpp
STDMETHOD(LeaveInterruptState) (THIS) PURE;
```
Helix Universal Server includes the standard SLTA application that simulates live broadcasts. This interface provides a means of designing custom applications that deviate from the standard simulated live transfer agent application. This interface replaces the IHXSLTA interface in all new programming.


The IHXiQSLTA interface contains the following methods:

- IHXiQSLTA::BeginTransmission
- IHXiQSLTA::Close
- IHXiQSLTA::Init—Advanced Mode
- IHXiQSLTA::Init—Basic Mode
- IHXiQSLTA::EndTransmission
- IHXiQSLTA::SetTAC
- IHXiQSLTA::SetTargetBandwidth

As with all Component Object Model (COM) interfaces, the IHXiQSLTA interface inherits the following IUnknown methods:

- IUnknown::AddRef
- IUnknown::QueryInterface
- IUnknown::Release

**IHXiQSLTA::BeginTransmission**

Starts transmitting the file to the server(s).

```cpp
STDMETHOD(BeginTransmission) (  
    THIS_,  
    IHXBuffer* pMediaName,  
    IHXBuffer* pSessionName  
) PURE;
```

- **pMediaName**
  Pointer to an IHXBuffer interface that manages the directory path to the pre-encoded media stored on disk.

- **pSessionName**
  Pointer to an IHXBuffer interface that manages the name of the session that will represent the live stream of the pre-encoded media on the recipient server.
**IHXiQSLTA::Close**

Cleans up the SLTA implementation's resources.

```cpp
STDMETHOD(Close) (THIS ) PURE;
```

**IHXiQSLTA::Init—Advanced Mode**

Initializes the remote broadcast library. In advanced mode initialization, a special configuration file in XML format is provided to perform the initialization process.

```cpp
STDMETHOD(Init) (THIS_, IHXBuffer* pConfigFilePath, IHXBuffer* pSessionName ) PURE;
```

- **pConfigFilePath**
  Pointer to an IHXBuffer interface that manages the directory path to the configuration file.

- **pSessionName**
  Pointer to an IHXBuffer interface that manages the session name.

**IHXiQSLTA::Init—Basic Mode**

Initializes the remote broadcast library. In basic mode initialization, the SLTA will configure itself according to a set of default parameters specified in this method. No XML configuration file needs to be provided.

```cpp
STDMETHOD(Init) (THIS_, const char* host, UINT16 httpPort, const char* user, const char* password, IHXBuffer* pSessionName, BOOL bTCP ) PURE;
```

- **host**
  Pointer to the host to which you want to connect. This parameter can either be in dotted decimal notation or a fully-qualified domain name.

- **httpPort**
  The HTTP port number to which you want to connect.

- **user**
  Pointer to the user name supplied to make the connection.

- **password**
  Pointer to the password supplied to make the connection.
pSessionName
  Pointer to an IHXBuffer interface that manages the session name.

bTCP
  If TRUE, the connection is made using TCP. If FALSE, the connection is made using UDP.

**IHXiQSLTA::EndTransmission**

End the file transmission to the server(s).

```cpp
STDMETHOD(EndTransmission) (THIS) PURE;
```

**IHXiQSLTA::SetTAC**

Sets the title, author, and copyright information for the stream. This method must be called before IHXiQSLTA::BeginTransmission to have any effect.

IHXiQSLTA::SetTAC overrides the displayed title, author, and copyright. This method does not need to be used unless, for example, you have several items to play and you want to override the given title, author, and copyright with your own information. If you do not use this method, the title, author, and copyright for each individual file will still be displayed.

```cpp
STDMETHOD(SetTAC) (THIS, const char* Title, const char* Author, const char* Copyright) PURE;
```

**Title**
  Pointer to the title information for the stream.

**Author**
  Pointer to the author information for the stream.

**Copyright**
  Pointer to the copyright information for the stream.

**IHXiQSLTA::SetTargetBandwidth**

Sets the target bandwidth for rule subscription. This method should not be used unless you want to purposely override the bandwidth of the stream. Normally, the simulated live transfer agent (SLTA) sends data at whatever bandwidth the connection between the server and client can handle.

```cpp
STDMETHOD(SetTargetBandwidth) (THIS, UINT32 ulTargetBW) PURE;
```

**ulTargetBW**
  The target bandwidth in bits per second.
IHXKeyboardFocus

Purpose: Provides keyboard focus for a specific site user.
Implemented by: Site object (client core)
Used by: Display renderer
Header file: hxwin.h

This interface either provides information or gets information about the site that will receive keyboard focus (that is, the site where the user can input data by means of the keyboard).

The IHXKeyboardFocus interface contains the following methods:

- IHXKeyboardFocus::GetKeyboardFocus
- IHXKeyboardFocus::SetKeyboardFocus

As with all Component Object Model (COM) interfaces, the IHXKeyboardFocus interface inherits the following IUnknown methods:

- IUnknown::AddRef
- IUnknown::QueryInterface
- IUnknown::Release

IHXKeyboardFocus::GetKeyboardFocus

Gets the keyboard focus for the specified site user.

STDMETHOD(GetKeyboardFocus) (THIS_ IHXSiteUser* &pSiteUser) PURE;

pSiteUser
Pointer to an IHXSiteUser interface that manages the location of the site user.

IHXKeyboardFocus::SetKeyboardFocus

Sets the keyboard focus for the specified site user.

STDMETHOD(SetKeyboardFocus) (THIS_ IHXSiteUser* pSiteUser) PURE;

pSiteUser
Pointer to an IHXSiteUser interface that manages the site user.
IHXKeyValueList

Purpose: Stores a non-uniquely keyed list of strings.
Implemented by: Server or client core
Used by: RTSP and HTTP protocols
Header file: hxvalue.h

This interface stores a list of name (or key) and value pairs in much the same way as IHXValues. However, while IHXValues stores these pairs as either unsigned long, buffer, or string data, this interface will only store string data. In addition, you can use this interface to store strings that are not necessarily identified by a unique key, that is, a series of string values can have the same key name. You can repeatedly use IHXKeyValueListIter::GetNextPair to retrieve the entire list of string key and value pairs (which works in much the same way as using IHXValues::GetFirstPropertyCString and IHXValues::GetNextPropertyCString), or IHXKeyValueListIterOneKey::GetNextString to retrieve all of the strings associated with a specific key.

The IHXKeyValueList interface contains the following methods:

- IHXKeyValueList::AddKeyValue
- IHXKeyValueList::AppendAllListItems
- IHXKeyValueList::CreateObject
- IHXKeyValueList::GetIter
- IHXKeyValueList::GetIterOneKey
- IHXKeyValueList::ImportValues
- IHXKeyValueList::KeyExists

As with all COM interfaces, the IHXKeyValueList interface inherits the following IUnknown methods:

- IUnknown::AddRef
- IUnknown::QueryInterface
- IUnknown::Release

IHXKeyValueList::AddKeyValue

Adds a new key and value pair to the list of strings. Because multiple strings can exist for the same key, the key does not have to be unique.

STDMETHOD(AddKeyValue) (  
    const char* pKey,  
    IHXBuffer* pStr  
) PURE;

pKey
    Pointer to the key that identifies the string the list.

pStr
    Pointer to an IHXBuffer interface that manages the new string to add to the list of strings.
**IHXKeyValueList::AppendAllListItems**

Appends all the key and value lists from another list to this list. You can have duplicate keys.

```cpp
STDMETHOD(AppendAllListItems) (THIS_IHXKeyValueList* pList)
```

*pList*

Pointer to an IHXKeyValueList interface that manages the list on which the other list will be appended.

**IHXKeyValueList::CreateObject**

Creates a new empty list that has the same properties as the current list.

```cpp
STDMETHOD(CreateObject) (THIS_REF(IHXKeyValueList*) pNewList)
```

*pNewList*

Returns a pointer to an IHXKeyValueList interface that manages the new, empty list.

**IHXKeyValueList::GetIter**

Returns an iterator with which you can iterate through all the key and value lists in the current list of strings.

```cpp
STDMETHOD(GetIter) (THIS_REF(IHXKeyValueListIter*) pIter)
```

*pIter*

Returns a pointer to an IHXKeyValueListIter interface that manages the retrieval of key and value pairs from the current list of strings.

**IHXKeyValueList::GetIterOneKey**

Returns an iterator with which you can iterate through all the strings for a particular key.

```cpp
STDMETHOD(GetIterOneKey) (THIS_const char* pKey, THIS_REF(IHXKeyValueListIterOneKey*) pIter)
```

*pKey*

Pointer to a key that identifies the strings.

*pIter*

Returns a pointer to an IHXKeyValueListIterOneKey interface that manages the retrieval of strings that match the specified key.
IHXKeyValueList::ImportValues

Imports all the strings from an IHXValues list into this list. If this object also supports IHXValues, it should also import the ULONG32 and buffer lists supported by IHXValues. You can have duplicate keys, and old data is left untouched.

```cpp
STDMETHOD(ImportValues) (  
    IHXValues* pValues
) PURE;
```

`pValues`
- Pointer to an IHXValues interface that manages the strings to be imported.

IHXKeyValueList::KeyExists

Indicates whether any strings exist for a specified key. Returns TRUE if the string exists for the key. Returns FALSE if no strings exist for the key.

```cpp
STDMETHOD_(BOOL,KeyExists) (  
    const char* pKey
) PURE;
```

`pKey`
- Pointer to the key to check for strings.
**IHXKeyValueListIter**

Purpose: Retrieves key and value pairs from a list of strings.

Implemented by: Server or client core

Used by: RTSP and HTTP protocols

Header file: hxvalue.h

This interface gets individual key and value pairs from a list of strings. Optionally, this interface can replace the value in the pair with a new value. This interface is instantiated using IHXKeyValueList::GetIter.

The IHXKeyValueListIter interface contains the following methods:

- IHXKeyValueListIter::GetNextPair
- IHXKeyValueListIter::ReplaceCurr

As with all COM interfaces, the IHXKeyValueListIter interface inherits the following IUnknown methods:

- IUnknown::AddRef
- IUnknown::QueryInterface
- IUnknown::Release

**IHXKeyValueListIter::GetNextPair**

Retrieves one key and value pair from the list of strings. Strings are retrieved from the list of strings in the same order in which they were inserted in the list.

```
STDMETHOD(GetNextPair) (THIS_, REF(const char*) pKey, REF(IHXBuffer*) pStr) PURE;
```

`pKey`

Returns a pointer to the key that identifies the string.

`pStr`

Returns a pointer to an IHXBuffer interface that manages the string associated with the key.

**IHXKeyValueListIter::ReplaceCurr**

Replaces the value in the key and value pair that was returned in the last call to IHXKeyValueListIter::GetNextPair with a new string.

```
STDMETHOD(ReplaceCurr) (THIS_, IHXBuffer* pStr) PURE;
```

`pStr`

Pointer to an IHXBuffer interface that manages the new string.
IHXKeyValueListIterOneKey

Purpose: Retrieves items from a list of strings that match a particular key.
Implemented by: Server or client core
Used by: RTSP and HTTP protocols
Header file: hxvalue.h

This interface gets individual key and value pairs from a list of strings that match the key specified when this interface was instantiated using IHXKeyValueList::GetIterOneKey. Optionally, this interface can replace the value in the pair with a new value. This is especially useful when you need to replace several strings that use the same key.

The IHXKeyValueListIterOneKey interface contains the following methods:
  • IHXKeyValueListIterOneKey::GetNextString
  • IHXKeyValueListIterOneKey::ReplaceCurr

As with all COM interfaces, the IHXKeyValueListIterOneKey interface inherits the following IUnknown methods:
  • IUnknown::AddRef
  • IUnknown::QueryInterface
  • IUnknown::Release

IHXKeyValueListIterOneKey::GetNextString

Retrieves one string that matches the key specified in the IHXKeyValueList::GetIterOneKey method. Strings are retrieved from the list of strings in the same order in which they were inserted in the list.

STDMETHOD(GetNextString) (
    THIS_ 
    REF(IHXBuffer*) pStr 
) PURE;

pStr Returns a pointer to an IHXBuffer interface that manages the string that matches the key.

IHXKeyValueListIterOneKey::ReplaceCurr

Replaces the value in the key and value pair that was referenced in the last call to IHXKeyValueListIterOneKey::GetNextString with a new string.

STDMETHOD(ReplaceCurr) (
    THIS_ 
    IHXBuffer* pStr 
) PURE;

pStr Pointer to an IHXBuffer interface that manages the new string.
**IHXListenResponse**

Purpose: Receives notice of TCP connections on a port.

Implemented by: Any component

Used by: TCP listen object (Network Services)

Header file: hxengin.h

Used with Network Services, this is the response interface to IHXListenSocket. After the component creates a listen object with IHXNetworkServices and initializes the object for a specific port with IHXListenSocket, the listen object notifies the component through this interface when a TCP connection is made on that port.


The IHXListenResponse interface contains the IHXListenResponse::NewConnection method.

As with all Component Object Model (COM) interfaces, the IHXListenResponse interface inherits the following IUnknown methods:

- IUnknown::AddRef
- IUnknown::QueryInterface
- IUnknown::Release

**IHXListenResponse::NewConnection**

Indicates whether a TCP connection has been made to the specified port.

```c
STDRESULT(NewConnection) ( 
    THIS_ 
    HX_RESULT status, 
    IHXTCPSocket* pTCPSocket 
) PURE;
```

status

Contains the response status to the IHXListenSocket::Init operation. If this parameter is HXR_OK, the call completed successfully.

pTCPSocket

Pointer to an IHXTCPSocket interface that manages the read and write operations to the TCP socket.
**IHXListenSocket**

**Purpose:** Sets up a component to listen for TCP connections on a port.

**Implemented by:** TCP listen object (Network Services)

**Used by:** Any component

**Header file:** hxengin.h

This interface is part of Network Services. Any component that needs to listen for TCP connections on a Helix Universal Server port can use this interface. The component first creates a listen object with IHXNetworkServices, then initializes the object for a specific port with this interface. The listen object then notifies the component through IHXListenResponse when a TCP connection is made on that port.

**For More Information:** See “Chapter 6: Network Services” beginning in Volume 1, on page 81. See IHXTCPObject in Volume 2, on page 427.

The IHXListenSocket interface contains the IHXListenSocket::Init method.

As with all Component Object Model (COM) interfaces, the IHXListenSocket interface inherits the following IUnknown methods:

- IUnknown::AddRef
- IUnknown::QueryInterface
- IUnknown::Release

**IHXListenSocket::Init**

Initializes the listen object for a specific port.

```c
STDMETHOD(Init) (
    THIS_ UINT32 ulLocalAddr,
    UINT16 port,
    IHXListenResponse* pListenResponse
) PURE;
```

**ulLocalAddr**

The network interface to which you want to bind.

**port**

The port to which you want to bind.

**pListenResponse**

Pointer to an IHXListenResponse interface that notifies the component when a TCP connection is made on that port.
The `IHXLiveFileFormatInfo` interface implements various functionality used in retransmitting static files as if they were live content.

The `IHXLiveFileFormatInfo` interface contains the following methods:

- `IHXLiveFileFormatInfo::FormResendPacket`
- `IHXLiveFileFormatInfo::GetResendBitrate`
- `IHXLiveFileFormatInfo::GetResendDuration`
- `IHXLiveFileFormatInfo::IsLiveResendRequired`
- `IHXLiveFileFormatInfo::VerifyFileCompatibility`
- `IHXLiveFileFormatInfo::VerifyStreamCompatibility`

As with all COM interfaces, the `IHXLiveFileFormatInfo` interface inherits the following `IUnknown` methods:

- `IUnknown::AddRef`
- `IUnknown::QueryInterface`
- `IUnknown::Release`

### `IHXLiveFileFormatInfo::FormResendPacket`

Forms a live resend packet based on the original packet passed as the first parameter. This enables the file format plug-in to make resend packets distinguishable from original packets.

```cpp
STDMETHOD(FormResendPacket) ( 
    THIS_ 
    IHXPacket* pOriginalPacket, 
    REF(IHXPacket*) pResendPacket 
) PURE;
```

- `pOriginalPacket`  
  Pointer to an IHXPacket interface that manages the original packet.

- `pResendPacket`  
  Returns a pointer to an IHXPacket interface that manages the live resend packet.

### `IHXLiveFileFormatInfo::GetResendBitrate`

Gets the rate at which packets should be resent if periodic live resends are required for this stream.
STDMETHOD(GetResendBitrate) (
    THIS_,
    UINT16 unStreamNumber,
    REF(UINT32) ulBitrate
) PURE;

unStreamNumber
The number of the stream.

ulBitrate
Returns the bit rate (in bits per second, or bps) at which packets should be resent.

IHXLiveFileFormatInfo::GetResendDuration

Gets the amount of time elapsed between the time a packet is sent and the time it is resent, if periodic live resends are required for this stream.

STDMETHOD(GetResendDuration) (
    THIS_,
    IHXPacket* pPacket,
    REF(UINT32) ulDuration
) PURE;

pPacket
Pointer to an IHXPacket interface that manages the packet.

ulDuration
Returns the amount of time that will elapse (in milliseconds) before this packet is resent.

IHXLiveFileFormatInfo::IsLiveResendRequired

Returns TRUE if the indicated stream requires the latest packet to be resent periodically in a live presentation.

STDMETHOD_(BOOL,IsLiveResendRequired) (
    THIS_,
    UINT16 unStreamNumber
) PURE;

unStreamNumber
The number of the stream.

IHXLiveFileFormatInfo::VerifyFileCompatibility

Compares two file headers and returns HXR_OK if these two files can be transmitted sequentially in a single live presentation.

STDMETHOD(VerifyFileCompatibility) (
    THIS_,
    IHXValues* pFileHeader1,
    IHXValues* pFileHeader2
) PURE;

pFileHeader1
Pointer to an IHXValues interface that manages the header for the first file.
pFileHeader2
  Pointer to an IHXValues interface that manages the header for the second file.

IHXLiveFileFormatInfo::VerifyStreamCompatibility

  Compares two stream headers and returns HXR_OK if these two streams can be transmitted sequentially in a single live presentation.

STDMETHOD(VerifyStreamCompatibility) (  
  THIS_  
  IHXValues* pStreamHeader1,  
  IHXValues* pStreamHeader2  
) PURE;

pStreamHeader1
  Pointer to an IHXValues interface that manages the header for the first stream.

pStreamHeader2
  Pointer to an IHXValues interface that manages the header for the second stream.
IHXLiveRealPix

**Purpose:** Sends LiveRealPix data and markup to the LiveRealPix Broadcast Library.

**Implemented by:** LiveRealPix Broadcast Library

**Used by:** LiveRealPix broadcast application

**Header file:** hxlvpix.h

This interface is used for live broadcasting of RealPix, as with successive jpeg images from a camera. A LiveRealPix broadcast application uses this interface to send LiveRealPix data and markup to the LiveRealPix Broadcast Library, which then broadcasts the images through Helix Universal Server. The response interface is IHXLiveRealPixResponse.

For More Information: See “RealPix Broadcast” in Volume 1, on page 110.

The IHXLiveRealPix interface contains the following methods:

- IHXLiveRealPix::GetTime
- IHXLiveRealPix::InitImage
- IHXLiveRealPix::Process
- IHXLiveRealPix::SendEffect
- IHXLiveRealPix::SendImage
- IHXLiveRealPix::StartEncoder
- IHXLiveRealPix::StopEncoder

As with all COM interfaces, the IHXLiveRealPix interface inherits the following IUnknown methods:

- IUnknown::AddRef
- IUnknown::QueryInterface
- IUnknown::Release

**IHXLiveRealPix::GetTime**

Returns the current time, in milliseconds, from the encoder.

```c++
 STDMETHODCALLTYPE (UINT32, GetTime) ( 
    THIS
 ) PURE;
```

**IHXLiveRealPix::InitImage**

 Prepares an image for being sent. This method simply breaks the image up into packets; nothing has yet been sent to the server. This is not an asynchronous call; all processing has finished by the time this call returns.

```c++
 STDMETHODCALLTYPE (InitImage) ( 
    THIS_
    PixImageInfo *pImageInfo
 ) PURE;
```
pImageInfo
Pointer to a PixImageInfo structure that contains all the information for images that should be sent down the stream.

IHXLiveRealPix::Process
Gives the library time to send packets to the server. This method should be called very often in between IHXLiveRealPix::SendImage and IHXLiveRealPixResponse::ImageSent, as well as between IHXLiveRealPix::SendEffect and IHXLiveRealPixResponse::EffectSent. Other than these times, it should be called every 3 to 5 seconds.

STDMETHOD(Process) (THIS) PURE;

IHXLiveRealPix::SendEffect
Creates an effect packet and immediately adds this packet to the packet send queue. Further calls to IHXLiveRealPix::Process result in this packet being sent to the server. A handle is returned in the PixEffectInfo structure by which this effect can later be identified. When the effect has been sent to the server, the encoder responds with IHXLiveRealPixResponse::EffectSent.

STDMETHOD(SendEffect) (THIS_ PixEffectInfo *pEffectInfo) PURE;

pEffectInfo
Pointer to a PixEffectInfo structure that contains all the information about the effect that the RealPix renderer should perform.

IHXLiveRealPix::SendImage
Transfers all the packets for the image into the packet send queue. Further calls to IHXLiveRealPix::Process result in these packets being sent to the server. When all the packets for this image have been sent, the encoder responds with IHXLiveRealPixResponse::EffectSent.

STDMETHOD(SendImage) (THIS_ UINT32 ulImageHandle) PURE;

ulImageHandle
The image handle for the packets being sent.

IHXLiveRealPix::StartEncoder
Starts the encoder. The encoder responds to this call asynchronously with IHXLiveRealPixResponse::EncoderStarted.
STDMETHOD(StartEncoder) (
    THIS_
    PixInitInfo *pInitInfo,
    IHXLiveRealPixResponse *pResponse
) PURE;

pInitInfo
    Pointer to a PixInitInfo structure that contains the initialization information.

pResponse
    Pointer to an IHXLiveRealPixResponse interface that manages the response to various methods in this interface.

IHXLiveRealPix::StopEncoder

Informs the encoder that no more images of effects are going to be sent to the encoder. The encoder shuts down the connection to the server and responds with IHXLiveRealPixResponse::EncoderStopped. You can call this method at any time after calling IHXLiveRealPix::StartEncoder. Also, because the encoder can call back with IHXLiveRealPixResponse::EncoderStopped at any time, your application should be prepared to handle that situation.

STDMETHOD(StopEncoder) (
    THIS
) PURE;
IHXLiveRealPixResend

Purpose: Resends and releases images.
Implemented by: LiveRealPix Broadcast Library
Used by: LiveRealPix broadcast application
Header file: hxlvpix.h

Normally, when you call IHXLiveRealPix::SendImage for a particular image, it puts the image packets for that image on the send queue, and discards the image. If you called IHXLiveRealPix::SendImage for the same image again, you get a HXR_UNKNOWN_IMAGE error. However, if you have queried (IUnknown::QueryInterface) for IHXLiveRealPixResend and called IHXLiveRealPixResend::InitResend, from that point on whenever you call IHXLiveRealPix::SendImage for that particular image, it puts the image packets on the send queue as it did before, but it does not delete the image. That way, you can call IHXLiveRealPix::SendImage for that particular image multiple times. When the application no longer needs to send this image again, it can call IHXLiveRealPixResend::ReleaseImage and this image will be deleted. From then on, if you IHXLiveRealPix::SendImage for that image, you get the HXR_UNKNOWN_IMAGE error.

The IHXLiveRealPixResend interface contains the following methods:

- IHXLiveRealPixResend::DumpAllQueuedPackets
- IHXLiveRealPixResend::InitResend
- IHXLiveRealPixResend::ReleaseImage

As with all COM interfaces, the IHXLiveRealPixResend interface inherits the following IUnknown methods:

- IUnknown::AddRef
- IUnknown::QueryInterface
- IUnknown::Release

IHXLiveRealPixResend::DumpAllQueuedPackets
Clears any packets currently waiting to be sent to the server. These packets were put on the send queue by either IHXLiveRealPix::SendImage or IHXLiveRealPix::SendEffect.

STDMETHOD(DumpAllQueuedPackets) (THIS ) PURE;

IHXLiveRealPixResend::InitResend
Inform the encoder that from this point on, whenever an image is sent with IHXLiveRealPix::SendImage, it should not be discarded. Instead they should be held in the encoder until IHXLiveRealPixResend::ReleaseImage is called.

STDMETHOD(InitResend) (THIS_ IHXValues* pOptions ) PURE;
pOptions
This parameter is for future use, and it should pass only an empty IHXValues or NULL value.

IHXLiveRealPixResend::ReleaseImage

Informs the encoder that the application no longer intends to call IHXLiveRealPix::SendImage on the specified image and that the encoder can discard the image. After the call to this method, any further calls to IHXLiveRealPix::SendImage for the released image returns HXR_UNKNOWN_IMAGE. If the image handle specified in this method is an unknown, this method returns HXR_UNKNOWN_IMAGE.

STDMETHOD(ReleaseImage) ( 
    THIS_,
    UINT32 ulImageHandle
) PURE;

ullImageHandle
The image handle for the packets.
IHXLiveRealPixResponse

Purpose: Provides responses from IHXLiveRealPix.

Implemented by: LiveRealPix broadcast application

Used by: LiveRealPix Broadcast Library

Header file: hxlvpix.h

This interface provides responses to many of the methods implemented by the IHXLiveRealPix interface.

**For More Information:** See “RealPix Broadcast” in Volume 1, on page 110.

The IHXLiveRealPixResponse interface contains the following methods:

- IHXLiveRealPixResponse::EffectSent
- IHXLiveRealPixResponse::EncoderStarted
- IHXLiveRealPixResponse::EncoderStopped
- IHXLiveRealPixResponse::ErrorOccurred
- IHXLiveRealPixResponse::ImageSent

As with all COM interfaces, the IHXLiveRealPixResponse interface inherits the following IUnknown methods:

- IUnknown::AddRef
- IUnknown::QueryInterface
- IUnknown::Release

**IHXLiveRealPixResponse::EffectSent**

Indicates whether the effect was sent successfully.

```cpp
STDMETHOD(EffectSent) (  
    THIS_  
    HX_RESULT status,  
    UINT32 ulEffectHandle  
) PURE;
```

- **status**
  Contains the response status to the IHXLiveRealPix::SendEffect operation. If this parameter is HXR_OK, the call completed successfully.

- **ulEffectHandle**
  Identifies the effect that was just sent to the server. This is the handle that was set in the m_ulHandle member of the PixEffectInfo structure by IHXLiveRealPix::SendEffect.

**IHXLiveRealPixResponse::EncoderStarted**

Indicates whether the encoder started successfully.
STDMETHOD(EncoderStarted) (  
    THIS_,
    HX_RESULT status,
    const char *pszHeaderString  
) PURE;

status  
Contains the response status to the IHXLiveRealPix::StartEncoder operation. If this parameter is HXR_OK, the call completed successfully.

pszHeaderString  
Pointer to the text string returned by the server (for information only).

IHXLiveRealPixResponse::EncoderStopped  
Indicates whether the encoder stopped successfully.

STDMETHOD(EncoderStopped) (  
    THIS_,
    HX_RESULT status  
) PURE;

status  
Contains the response status to the IHXLiveRealPix::StopEncoder operation. If this parameter is HXR_OK, the call completed successfully. At this point, the application can then shut down or call IHXLiveRealPix::StartEncoder again.

IHXLiveRealPixResponse::ErrorOccurred  
Indicates that the encoder has received an error from the server. Depending upon the severity of the error, the server might shut down the encoder and an IHXLiveRealPixResponse::EncoderStopped call would be made. Therefore, the application should be ready to handle an IHXLiveRealPixResponse::EncoderStopped call at any time. If an error occurs, you should call IHXLiveRealPix::StopEncoder and shut down the encoder from the application side.

STDMETHOD(ErrorOccurred) (  
    THIS_,
    const UINT8 unSeverity,
    const ULONG32 ulHXCode,
    const ULONG32 ulUserCode,
    const char *pszUserString,
    const char *pszMoreInfoURL  
) PURE;

unSeverity  
Severity of the error. This value impacts how the server reacts to the error. Depending on the error type, an error message with the Helix Architecture (HX) code and a string translation of that code is displayed. The error dialog box includes a “more info” section that displays the user code and string, and a link to the more info URL. In the server these messages are logged to the log file. One of the following values:

• HXLOG_EMERG
Indicates a panic condition. The server or client will halt or restart.

- **HXLOG_ALERT**
  Indicates a condition that should be corrected immediately and needs user intervention to prevent problems.

- **HXLOG_CRIT**
  Indicates critical conditions.

- **HXLOG_ERR**
  Indicates errors. Currently, this is the only error value passed.

- **HXLOG_WARNING**
  Indicates warning messages.

- **HXLOG_NOTICE**
  Indicates conditions that are not error conditions, but should possibly be handled specially.

- **HXLOG_INFO**
  Indicates informational messages.

- **HXLOG_DEBUG**
  Indicates messages that contain information normally of use only when debugging a program.

**ulHXCode**
A Helix Architecture (HX) error code, as defined in hxresult.h. This is translated to a text representation for display in an error dialog box or log file.

**ulUserCode**
User-specific error code. This is not translated to a text representation. This can be any value the caller wants; it will be logged or displayed but not interpreted.

**pUserString**
Pointer to a user-specific error string. This is not translated or modified. This can be any value the caller wants; it will be logged or displayed but not interpreted.

**pMoreInfoURL**
Pointer to a user-specific “more information” URL string.

### IHXLiveRealPixResponse::ImageSent

Indicates whether the send image operation completed successfully.

```c
STDMETHOD(ImageSent) ( 
    THIS_ 
    HX_RESULT status, 
    UINT32 ulImageHandle 
) PURE;
```

**status**
Contains the response status to the IHXLiveRealPix::SendImage operation. If this parameter is HXR_OK, the call completed successfully.
ullImageHandle

Identifies the image that was just sent to the server.
**IHXLiveText**

**Purpose:** Sends RealText data and markup to the RealText Broadcast Library.

**Implemented by:** RealText Broadcast Library

**Used by:** RealText broadcast application

**Header file:** hxlvtxt.h

This interface is used for live broadcasting of RealText, as with a live stock ticker feed. A RealText broadcast application uses this interface to send RealText data and markup to the RealText Broadcast Library, which then broadcasts the text through Helix Universal Server. There is no response interface.

**For More Information:** See “RealText Broadcast” in Volume 1, on page 105.

The IHXLiveText interface contains the following methods:

- `IHXLiveText::AddData`
- `IHXLiveText::AddTickerItem`
- `IHXLiveText::EncoderIsDone`
- `IHXLiveText::EncoderIsInitialized`
- `IHXLiveText::flush`
- `IHXLiveText::GetTime`
- `IHXLiveText::InitLiveText`
- `IHXLiveText::PacketsHaveStarted`
- `IHXLiveText::Process`
- `IHXLiveText::SetBackgroundColor`
- `IHXLiveText::SetDoLooping`
- `IHXLiveText::SetEncoderDone`
- `IHXLiveText::SetHyperlinkInfo`
- `IHXLiveText::SetTextMotion`
- `IHXLiveText::SetType`
- `IHXLiveText::SetWindowDimensions`
- `IHXLiveText::UseWordwrap`

As with all COM interfaces, the IHXLiveText interface inherits the following `IUnknown` methods:

- `IUnknown::AddRef`
- `IUnknown::QueryInterface`
- `IUnknown::Release`

**IHXLiveText::AddData**

Adds text to the stream.
STDMETHOD(AddData) ( 
    THIS_,
    char* szMoreData,
    BOOL bSendImmediately
) PURE;

szMoreData
Pointer to the text to add to the stream. The text can include any RealText markup. If the text does not include <time begin> tags, the broadcast library time stamps the text as it is received. So if a broadcast application sends text string B to the library one second after it sends string A, the clients’ RealText renderers display string B one second after displaying string A.

bSendImmediately
If TRUE, this method forces the encoder to send all text that has been added (and not yet sent) immediately. If FALSE, the encoder decides when to send the text it has buffered, based on the length of the text in the buffer and the amount of time that has elapsed since the last packet was sent.

IHXLiveText::AddTickerItem

Adds “tickertape” text to the stream. This method prepends “<TU>” to the characters in the szTickerUpperData parameter, and “<TL>” to the characters in the szTickerLowerData parameter. Therefore, calling AddTickerItem("ABCD", "5 1/2", TRUE) is the same as calling AddData("<TU>ABCD", FALSE) followed by AddData("<TL>5 1/2", TRUE), except that the former guarantees that the “upper” and “lower” items are sent in the same packet.

STDMETHOD(AddTickerItem) ( 
    THIS_,
    char* szTickerUpperData,
    char* szTickerLowerData,
    BOOL bSendImmediately
) PURE;

szTickerUpperData
A NULL-terminated string that contains the text to be displayed in the upper line of the tickertape.

szTickerLowerData
A NULL-terminated string that contains the text to be displayed in the lower line of the tickertape.

bSendImmediately
If TRUE, this string forces the encoder to send all text that has been added (and not yet sent) immediately. If FALSE, the encoder decides when to send the text it has buffered based on the length of the text in the buffer and the amount of time that has elapsed since the last packet was sent.

IHXLiveText::EncoderIsDone

Indicates whether the library has broadcast all the text it has received. A return value of TRUE indicates that it has finished broadcasting all the text.
IHXLiveText::EncoderIsDone

Indicates whether the library has received the initialization parameters that enable it to connect to the Helix Universal Server broadcast plug-in. After initialization, the library is ready to receive text. A return value of TRUE indicates that the library has received the initialization parameters.

IHXLiveText::flush

Tells the encoder to send everything that is in the buffer, then clear the buffer.

IHXLiveText::GetTime

Returns the current system time, in milliseconds.

IHXLiveText::InitLiveText

Sets up the encoder so it can stream data through the server. The user name, password, and port are all values set up on the server (located at pAddress). This method must be called before any of the other IHXLiveText methods.

pAddress
Pointer to the host name or IP address of the server.

port
The port number defined in the configuration file where the FSMount option has the following entry:

```
{ /encoder/, hx-encoder, Port=#### }
```
The value #### is the port number that encoders will use.

pUsername
Pointer to a NULL-terminated character buffer that contains the user name set up in the encoder realm of the server. This is the same user name used by RealAudio encoders.

pPassword
Pointer to a NULL-terminated character buffer that contains the password associated with the user name specified by pUsername. This is the same password used by RealAudio encoders.

pFilename
Pointer to the “file” name that clients use to view the stream. For example, if your server is called “srv1” and the RTSP port in the configuration file is 8080, and the file name from the encoder is “foo.rtx,” then a client could view the stream by opening “rtsp://srv1:8080/encoder/foo.rtx”.

IHXLiveText::PacketsHaveStarted
Indicates whether packets have started. A return value of TRUE indicates that the packets have started.

STDMETHOD_(BOOL, PacketsHaveStarted) (THIS)
PURE;

IHXLiveText::Process
Allows the library to perform necessary processing. This method should be called after calling IHXLiveText::AddData, IHXLiveText::AddTickerItem, or IHXLiveText::flush. When not sending text, the application should also call this method approximately every 5 seconds. This method is required due to the asynchronous nature of Helix.

STDMETHOD(Process) (THIS)
PURE;

IHXLiveText::SetBackgroundColor
Controls the window’s background color. If the application does not use this method, the property's default value is used.

STDMETHOD(SetBackgroundColor) (THIS_
char* szBackgroundColor
) PURE;

szBackgroundColor
Pointer to the background color to be set. This background color can be any valid color string specified in the SMIL 1.0 specification. These color values are also listed in the RealNetworks Production Guide at http://service.real.com/help/library/encoders.htm.
**IHXLiveText::SetDoLooping**

Determines whether looping occurs. This method for “TickerTape”-type windows only; it is ignored by all other types. If the application does not use this method, the property’s default value is used.

```cpp
STDMETHOD(SetDoLooping) (  
    THIS_  
    BOOL bDoLooping  
) PURE;
```

*bDoLooping*

If TRUE, looping will occur.

**IHXLiveText::SetEncoderDone**

Notifies the library that the application has finished sending text.

```cpp
STDMETHOD(SetEncoderDone) (  
    THIS  
) PURE;
```

**IHXLiveText::SetHyperlinkInfo**

Specifies whether hyperlinks are underlined and sets the hyperlink color. If the application does not use this method, the property’s default value is used.

```cpp
STDMETHOD(SetHyperlinkInfo) (  
    THIS_  
    BOOL underlineHyperlinks,  
    char* szLinkColor  
) PURE;
```

*underlineHyperlinks*

If TRUE, the hyperlinks are underlined.

*szLinkColor*

Pointer to a NULL-terminated buffer that contains a color value for the hyperlink. This color can be any valid color string specified in the SMIL 1.0 specification. These color values are also listed in the *RealNetworks Production Guide* at [http://service.real.com/help/library/encoders.htm](http://service.real.com/help/library/encoders.htm).

**IHXLiveText::SetTextMotion**

Sets the window’s scroll rate and crawl rate. If the application does not use this method, the property’s default value is used.

```cpp
STDMETHOD(SetTextMotion) (  
    THIS_  
    LONG32 scrollrate,  
    LONG32 crawlrate  
) PURE;
```

*scrollrate*

The window’s scroll rate (a value from 0 to 8192).
crawlrate
The window’s crawl rate (a value from 0 to 8192).

IHXLiveText::SetType
Determines the window type (generic, tickertape, and so on). If the application does not use this method, the property’s default value is used.

STDMETHOD(SetType) (THIS_,
    char* szType
) PURE;

szType
Points to the window type. Can be any of the following:

- generic
  A generic window with no preset parameters.

- scrollingnews
  A scrollingnews window is preset to have text scroll from the bottom of the window to the top at a set rate for the entire presentation.

- tickertape
  Text in a tickertape window crawls from the right side of the window to the left. It can also loop back around to the right. Text displays next to the window’s top or bottom edge.

- marquee
  The marquee window is like the tickertape in that text crawls from right to left and can loop. It is different in that text is centered vertically within the window.

- teleprompter
  A teleprompter window behaves like a generic window except that text arriving at the bottom edge of the window causes the text above it to move up just enough to display the new line.

IHXLiveText::SetWindowDimensions
Controls the window’s width and height. If the application does not use this method, the property’s default value is used.

STDMETHOD(SetWindowDimensions) (THIS_,
    ULONG32 width,
    ULONG32 height
) PURE;

width
The width of the window, in pixels.

height
The height of the window, in pixels.
**IHXLiveText::UseWordwrap**

Sets word wrap on or off. If the application does not use this method, the property's default value is used.

```cpp
STDMETHOD(UseWordwrap) ( 
    BOOL useWordwrap 
) PURE;
```

*useWordwrap*

- If TRUE, sets word wrap on.
The interface is an addition to the IHXLiveText interface, which is used for live broadcasting of RealText, as with a live stock ticker feed. A RealText broadcast application uses this interface to send RealText data and markup to the RealText Broadcast Library, which then broadcasts the text through Helix Universal Server. There is no response interface.

The IHXLiveText2 interface contains the following methods:

- IHXLiveText2::DeclareAverageBitrate
- IHXLiveText2::DeclareMaximumBitrate
- IHXLiveText2::GetLastSentTextEndTime
- IHXLiveText2::IgnoreExtraSpaces
- IHXLiveText2::SetFlags
- IHXLiveText2::SetTimeBetweenDryStreamResends

As with all COM interfaces, the IHXLiveText2 interface inherits the following IUnknown methods:

- IUnknown::AddRef
- IUnknown::QueryInterface
- IUnknown::Release

### IHXLiveText2::DeclareAverageBitrate

Gives an estimate of the bit rate needed by a live text stream so the server can determine whether or not a player can view the stream based on its available bandwidth. This method should be called before sending any data.

```c
STDMETHOD(DeclareAverageBitrate) (
    THIS_,
    ULONG32 ulAvgBitsPerSecond
) PURE;
```

`ulAvgBitsPerSecond`

The estimated bit rate, per second, needed by a live text stream.

### IHXLiveText2::DeclareMaximumBitrate

Gives the maximum bit rate needed by a live text stream so that the server can determine whether or not a player can view the stream based on its available bandwidth. This method should be called before sending any data.

```c
STDMETHOD(DeclareMaximumBitrate) (
    THIS_,
    ULONG32 ulMaxBitsPerSecond
) PURE;
```
ulMaxBitsPerSecond
The maximum bit rate, per second, needed by a live text stream.

IHXLiveText2::GetLastSentTextEndTime
Tells when the last-sent text will end. This enables you to determine when it is all right to send more text. For example, if you are streaming a presentation that is scrolling at a rate of 20 pixels per second in a window that is 200 pixels high, and you call IHXLiveText::AddData with 100 lines of 20-point text, it will take 100 seconds for all that text to scroll up and out of the window. This method would return the current time plus 100 if you called it right after calling IHXLiveText::AddData.

You can call this method at any time while the encoder is running.

STDMETHOD(GetLastSentTextEndTime) (
    THIS_,
    ULONG32& ulLastEndTime
) PURE;

ulLastEndTime
Address of the time when the last text sent will end.

Note: If looping (see IHXLiveText::SetDoLooping) is set to TRUE in tickertape and marquee (and other windows with horizontal motion), then this method returns a value that is the start time of the last-added text. This is necessary because looping text has no defined end time.

IHXLiveText2::IgnoreExtraSpaces
Determines how multiple, contiguous space characters are presented. This method should be called before sending any data.

STDMETHOD(IgnoreExtraSpaces) (
    THIS_,
    BOOL bIgnoreExtraSpaces
) PURE;

bIgnoreExtraSpaces
If FALSE (default), multiple spaces in non-tag text are rendered. If TRUE, multiple, contiguous spaces are treated as a single space.

IHXLiveText2::SetFlags
This method is for internal testing use only and does not affect the presentation of a live text stream.

IHXLiveText2::SetTimeBetweenDryStreamResends
Adjusts the amount of time (since the last data was sent) that the encoder waits before resending “heartbeat” packets. These heartbeat packets let the server know that the encoder is still alive and to let newly connected players know what is currently visible in the window, in case they connected during a “dry” period in the stream, after the last new text was sent.

You can call this method at any time while the encoder is running.
STDMETHOD(SetTimeBetweenDryStreamResends) (  
    THIS,  
    ULONG32 ulMillisecBetweenResends  
) PURE;

ulMillisecBetweenResends
The number of milliseconds between resends. The minimum value for this parameter is 500 milliseconds. The default value for this parameter is 3000 milliseconds. Any higher value would result in longer connection times for players coming in during a “dry” spell in the stream. Any sufficiently higher value might cause the server to disconnect, because it assumes that the encoder has stopped if it does not periodically send data.
The `IHXLoadBalancedListen` interface is queried off of `IHXListenSocket`. This interface enables a plug-in to specify that it wants the server to load balance multiple instances of itself. The server will instantiate multiple instances of the plug-in as needed based on socket and descriptor limits. Each plug-in instance should attempt to listen on the same port as other instances (because they will share the port).

The `IHXLoadBalancedListen` interface contains the following methods:

- `IHXLoadBalancedListen::SetID`
- `IHXLoadBalancedListen::SetReserveLimit`

As with all COM interfaces, the `IHXLoadBalancedListen` interface inherits the following `IUnknown` methods:

- `IUnknown::AddRef`
- `IUnknown::QueryInterface`
- `IUnknown::Release`

**IHXLoadBalancedListen::SetID**

Sets the unique ID for this listen socket. This ID determines whether or not different instances of a plug-in trying to listen on a single port are actually the same plug-in. Without this method, it would be possible for two completely different plug-ins to listen on the same port using the load balanced listener.

```cpp
STDMETHOD(SetID) ( IUnknown::AddRef

ID

The unique ID for this listen socket.

**IHXLoadBalancedListen::SetReserveLimit**

Sets the reserve limit for descriptors and sockets. If less than the reserve limit of descriptors and sockets are left, then a new instance of the plug-in is created.

```cpp
STDMETHOD(SetReserveLimit) ( UINT32 ulDescriptors,

ulDescriptors

The reserve limit for descriptors.
**ulSockets**

The reserve limit for sockets.
**IHXMultiInstanceSiteUserSupplier**

**Purpose:** Attaches a Multi-Instance Site User Supplier (MISUS) object to a site.

**Implemented by:** MISUS object (client core)

**Used by:** Display rendering plug-in

**Header file:** hxwin.h

This is the interface to a MISUS object instantiated by the Helix client core. In normal Helix windowing, a display rendering plug-in that supports multiple sites renders data to this object, which in turn renders data to the multiple sites. The preferred implementation is to use IHXCommonClassFactory to create this interface.

*For More Information:* See “Sites (Windowing)” in Volume 1, on page 183. See also IHXSiteUserSupplier in Volume 2, on page 399.

The **IHXMultiInstanceSiteUserSupplier** interface contains the following methods:

- **IHXMultiInstanceSiteUserSupplier::ReleaseSingleSiteUser**
- **IHXMultiInstanceSiteUserSupplier::SetSingleSiteUser**

As with all COM interfaces, the **IHXMultiInstanceSiteUserSupplier** interface inherits the following **IUnknown** methods:

- **IUnknown::AddRef**
- **IUnknown::QueryInterface**
- **IUnknown::Release**

**IHXMultiInstanceSiteUserSupplier::ReleaseSingleSiteUser**

Releases the renderer from the site.

```cpp
STDMETHOD(ReleaseSingleSiteUser) (THIS) PURE;
```

**IHXMultiInstanceSiteUserSupplier::SetSingleSiteUser**

Sets a single renderer that could be used for multiple sites.

```cpp
STDMETHOD(SetSingleSiteUser) (THIS_,
   IUnknown* pUnknown)
) PURE;
```

**pUnknown**

Pointer to the **IUnknown** interface that identifies the instance of the MISUS object (IHXSiteUser).
IHXMutex

Purpose: Provides thread synchronization.
Implemented by: Server core
Used by: Server plug-ins and server core
Header file: hxengin.h

This interface provides thread-exclusive access to any piece of code.
The IHXMutex interface contains the following methods:

- IHXMutex::Lock
- IHXMutex::TryLock
- IHXMutex::Unlock

As with all COM interfaces, the IHXMutex interface inherits the following IUnknown methods:

- IUnknown::AddRef
- IUnknown::QueryInterface
- IUnknown::Release

IHXMutex::Lock

Locks the code from being used by any other process.

```cpp
STDMETHOD(Lock) ( 
    THIS
 ) PURE;
```

IHXMutex::TryLock

Returns with the status of a single lock attempt. If successful (returns HXR_OK), the code is locked.

```cpp
STDMETHOD(TryLock) ( 
    THIS
 ) PURE;
```

IHXMutex::Unlock

Unlocks the code.

```cpp
STDMETHOD(Unlock) ( 
    THIS
 ) PURE;
```
IHXNetworkInterfaceEnumerator

Purpose: Enumerates a list of local interfaces.
Implemented by: Network services
Used by: Client core
Header file: hxengin.h

The IHXNetworkInterfaceEnumerator interface contains the IHXNetworkInterfaceEnumerator::EnumerateInterfaces method.

As with all Component Object Model (COM) interfaces, the IHXNetworkInteraceEnumerator interface inherits the following IUnknown methods:

- IUnknown::AddRef
- IUnknown::QueryInterface
- IUnknown::Release

IHXNetworkInterfaceEnumerator::EnumerateInterfaces

Gets a list of local interfaces.

STDMETHOD(EnumerateInterfaces) (THIS_ 
    REF(UINT32*) pulInterfaces,
    REF(UINT32) ulNumInterfaces
) PURE;

pulInterfaces
Returns a pointer to the list of local interfaces. If the buffer passed in is too small, this method returns HXR_BUFFERTOOSMALL with the ulNumInterfaces parameter updated to the proper number of interfaces.

ulNumInterfaces
Returns the number of local interfaces in the list.
**IHXNetworkServices**

**Purpose:** Creates networking objects.

**Implemented by:** Network Services

**Used by:** Any component

**Header file:** hxengin.h

Helix architecture’s Network Services implements this interface, which any Helix component can use to generate network socket objects. Depending on the object type, the component can then use IHXListenSocket, IHXResolver, IHXTCPSocket, or IHXUDPSocket. Broadcast and monitor plug-ins typically use this interface. For example, a broadcast plug-in uses Network Services to get a TCP socket to connect to its live source over a TCP/IP network.

**For More Information:** See “Broadcast Plug-in” in Volume 1, on page 91 or “Creating a Monitor Plug-in” in Volume 1, on page 124.

The IHXNetworkServices interface contains the following methods:

- IHXNetworkServices::CreateListenSocket
- IHXNetworkServices::CreateResolver
- IHXNetworkServices::CreateTCPSocket
- IHXNetworkServices::CreateUDPSocket

As with all COM interfaces, the IHXNetworkServices interface inherits the following IUnknown methods:

- IUnknown::AddRef
- IUnknown::QueryInterface
- IUnknown::Release

**IHXNetworkServices::CreateListenSocket**

Creates a new TCP socket that listens for connections on a particular port.

```cpp
STDMETHOD(CreateListenSocket) ( 
   THIS_ 
   IHXListenSocket** ppListenSocket 
) PURE;

ppListenSocket
   Pointer to a pointer to an IHXListenSocket interface that manages the new TCP socket.
```

**IHXNetworkServices::CreateResolver**

Creates a new resolver that can look up host names.

```cpp
STDMETHOD(CreateResolver) ( 
   THIS_ 
   IHXResolver** ppResolver 
) PURE;
```
ppResolver
  Pointer to a pointer to an IHXResolver interface that resolves a DNS host name into an IP address.

IHXNetworkServices::CreateTCPSocket
Create a new TCP socket.
STDMETHOD(CreateTCPSocket) (  
  THIS_  
  IHXTCPSocket** ppTCPocket
) PURE;

ppTCPocket
Pointer to pointer to an IHXTCPSocket interface that manages the new TCP socket.

IHXNetworkServices::CreateUDPSocket
Create a new UDP socket.
STDMETHOD(CreateUDPSocket) (  
  THIS_  
  IHXUDPSocket** ppUDPSocket
) PURE;

ppUDPSocket
Pointer to a pointer to an IHXUDPSocket interface that manages the new UDP socket.
IHXNetworkServices2

Purpose: Creates additional networking objects.
Implemented by: Network Services
Used by: Any component
Header file: hxengin.h

Like the IHXNetworkServices interface, this interface generates network socket objects. Depending on the object type, the component can then use IHXListenSocket, IHXResolver, IHXTCPsocket, or IHXUDPSocket. In addition, this interface also creates new, locally-bound TCP sockets.

The IHXNetworkServices2 interface inherits the methods of the IHXNetworkServices interface, and in addition contains the IHXNetworkServices2::CreateLBoundTCPSocket method.

As with all Component Object Model (COM) interfaces, the IHXNetworkServices2 interface also inherits the following IUnknown methods:

- IUnknown::AddRef
- IUnknown::QueryInterface
- IUnknown::Release

IHXNetworkServices2::CreateLBoundTCPSocket

Creates a new, locally-bound TCP socket.

STDMETHOD(CreateLBoundTCPSocket) (THIS_
    IHXTCPsocket** ppTCPsocket
) PURE;

ppTCPsocket
Address of a pointer to an IHXTCPsocket interface that manages the new, locally-bound TCP socket.
**IHXObjectConfiguration**

Purpose: Initializes and configures plug-ins.

Implemented by: Plug-ins

Used by: Server core

Header file: hxplgns.h

This interface contains alternative methods for initializing and configuring plug-ins. You can use the IHXObjectConfiguration::SetContext method instead of IHXPlugin::InitPlugin for setting the context of the plug-in. This is especially useful if you only need to set the configuration of the plug-in after the plug-in has been initialized, which you can accomplish by using the IHXObjectConfiguration::SetConfiguration method.

For More Information: See “Additional Plug-in Initialization” in Volume 1, on page 41.

The IHXObjectConfiguration interface contains the following methods:

- IHXObjectConfiguration::SetConfiguration
- IHXObjectConfiguration::SetContext

As with all COM interfaces, the IHXObjectConfiguration interface inherits the following IUnknown methods:

- IUnknown::AddRef
- IUnknown::QueryInterface
- IUnknown::Release

**IHXObjectConfiguration::SetConfiguration**

Supplies configuration information from the plug-in. This configuration information is often a set of CString properties extracted from a list in the configuration file. This enables each plug-in within a specific class (that is, an authorization plug-in, a database plug-in, and so on) to require a different set of parameters. Typical configuration information handled by this method are realm, database ID, and so on. For example, you might use this method to configure an authentication object. The server core reads the server configuration file to determine the realm and database ID for authenticating a particular audio clip. If the clip’s realm uses basic authentication, the server core will create an object that supports basic authentication. It will then call this method on that object so the object knows which realm and database ID to user for that clip.

```cpp
STDMETHOD(SetConfiguration) ( 
    THIS_ 
    IHXValues* pIHXValuesConfiguration
) PURE;

pIHXValuesConfiguration
    Pointer to an IHXValues interface that manages the configuration information.
```
**IHXObjectConfiguration::SetContext**

Passes in an object to a plug-in so the plug-in can use services from the server core. Both IHXPlugin::InitPlugin and this method must be called before calling any other method (except IHXPlugin::GetPluginInfo) on the plug-in. The server can use this method instead of IHXPlugin::InitPlugin if the server does not need to assume that the object being dealt with is a plug-in.

```cpp
STDMETHOD(SetContext) (
    THIS_
    IUnknown* pIUnknownContext
) PURE;
```

*pIUnknownContext*  
Pointer to the IUnknown interface that identifies the instance of this object.
The IHXOptimizedScheduler interface provides the user with a way of scheduling callbacks that will be executed at some time in the future. This interface should only be used if you need accurately timed callbacks. These callbacks should be efficient and should not consume much time/CPU. This is not a thread-safe interface; therefore, you must take care of synchronization in your callbacks.

The IHXOptimizedScheduler interface contains the following methods:

- IHXOptimizedScheduler::AbsoluteEnter
- IHXOptimizedScheduler::GetCurrentSchedulerTime
- IHXOptimizedScheduler::RelativeEnter
- IHXOptimizedScheduler::Remove

As with all COM interfaces, the IHXOptimizedScheduler interface inherits the following IUnknown methods:

- IUnknown::AddRef
- IUnknown::QueryInterface
- IUnknown::Release

### IHXOptimizedScheduler::AbsoluteEnter
Schedules a callback to be executed at a specified time. Returns a handle to the callback.

```cpp
STDMETHOD_(CallbackHandle, AbsoluteEnter) (THIS_ IHXCallback* pCallback, HXTimeval tVal) PURE;
```

- **pCallback**: Pointer to an IHXCallback interface that manages the scheduled callback.
- **tVal**: Pointer to an HXTimeval structure that specifies the time for the callback to be executed.

### IHXOptimizedScheduler::GetCurrentSchedulerTime
Returns a pointer to an HXTimeval structure that contains the current time in the timeline of the scheduler.

```cpp
STDMETHOD_(HXTimeval, GetCurrentSchedulerTime) (THIS) PURE;
```
**IHXOptimizedScheduler::RelativeEnter**

Schedules a callback to be executed a specified number of milliseconds from now. This method is less precise than IHXOptimizedScheduler::AbsoluteEnter and should only be used when accurate timing is not critical. Returns the handle to the callback.

```cpp
STDMETHOD_(CallbackHandle, RelativeEnter) (THIS_ IHXCallback* pCallback, UINT32 ms) PURE;
```

- **pCallback**
  - Pointer to an IHXCallback interface that manages the scheduled callback.
- **ms**
  - The number of milliseconds before the callback is executed.

**IHXOptimizedScheduler::Remove**

Removes a callback from the scheduler.

```cpp
STDMETHOD(Remove) (THIS_ CallbackHandle Handle) PURE;
```

- **Handle**
  - The handle to the callback to be removed.
IHXOptions

Purpose: Sets options on any component.
Implemented by: Any component
Used by: Any other component
Header file: hxvalue.h

If a component implements this interface, the caller can find out all of the options (and their types and current settings) by calling the IHXOptions::GetOptions method. Then the caller can set any individual option by calling the appropriate IHXOptions method, depending on the option type.

The IHXOptions interface contains the following methods:

- IHXOptions::GetOptions
- IHXOptions::SetOptionBuffer
- IHXOptions::SetOptionCString
- IHXOptions::SetOptionULONG32

As with all COM interfaces, the IHXOptions interface inherits the following IUnknown methods:

- IUnknown::AddRef
- IUnknown::QueryInterface
- IUnknown::Release

IHXOptions::GetOptions

Returns a list of the options supported by this particular object, along with the value currently set for each option. Enumerate the members of the returned IHXValues object to discover what options a component supports and the type of each of those options. The value for each name-value pair is the current setting for that option.

STDMETHOD(GetOptions) (THIS_ REF(IHXValues*) pOptions) PURE;

pOptions
Returns a pointer to an IHXValues interface that manages the list of options.

IHXOptions::SetOptionBuffer

Sets the value of a buffer option.

STDMETHOD(SetOptionBuffer) (THIS_ const char* pName,
IHXBuffer* pValue) PURE;

pName
Pointer to the name of the option being set.
pValue
  Pointer to an IHXBuffer interface that manages the buffer value of the option being set.

IHXOptions::SetOptionCString

Sets the value of a CString option.
STDMETHOD(SetOptionCString) (
  THIS_  
  const char* pName,  
  IHXBuffer* pValue
) PURE;

pName
  Pointer to the name of the option being set.

pValue
  Pointer to an IHXBuffer interface that manages the CString value of the option being set.

IHXOptions::SetOptionULONG32

Sets the value of a ULONG32 option.
STDMETHOD(SetOptionULONG32) (
  THIS_  
  const char* pName,  
  ULONG32 ulValue
) PURE;

pName
  Pointer to the name of the option being set.

ulValue
  The value of the option being set.
IHXOverrideDefaultServices

Purpose: Overrides any default services provided by the system.

Implemented by: Helix client

Used by: Plug-ins

Header file: hxengin.h

This interface is queried off of the context. Currently, this interface is supported only on the client side. Using this interface, you can currently override:

- IHXNetworkServices
- IHXPlugin2Handler
- IHXPreferences

You can use the same interface to later restore the overridden services by calling the IHXOverrideDefaultServices::OverrideServices method with the original service that was queried (IUnknown::QueryInterface) before the initial override.

The IHXOverrideDefaultServices interface contains the IHXOverrideDefaultServices::OverrideServices method.

As with all COM interfaces, the IHXOverrideDefaultServices interface inherits the following IUnknown methods:

- IUnknown::AddRef
- IUnknown::QueryInterface
- IUnknown::Release

IHXOverrideDefaultServices::OverrideServices

Overrides the default services provided by the system.

STDMETHOD(OverrideServices) (THIS_ IUnknown* pContext) PURE;

pContext

Pointer to the IUnknown interface that identifies the instance of the object for which services are being overridden.
The IHXPacket interface contains the following methods:

- IHXPacket::Get
- IHXPacket::GetASMFlags
- IHXPacket::GetASMRuleNumber
- IHXPacket::GetBuffer
- IHXPacket::GetStreamNumber
- IHXPacket::GetTime
- IHXPacket::IsLost
- IHXPacket::Set
- IHXPacket::SetAsLost

As with all COM interfaces, the IHXPacket interface inherits the following IUnknown methods:

- IUnknown::AddRef
- IUnknown::QueryInterface
- IUnknown::Release

**IHXPacket::Get**

Retrieves the values from the packet all at one time.

```cpp
STDMETHOD(Get) (  
    THIS_  
    REF(IHXBuffer*) pBuffer,  
    REF(UINT32) ulTime,  
    REF(UINT16) unStreamNumber,  
    REF(UINT8) unASMFlags,  
    REF(UINT16) unASMRuleNumber  
) PURE;
```
pBuffer
   Returns a pointer to the IHXBuffer interface that manages the packet data.

ulTime
   Returns the time stamp for the packet.

unStreamNumber
   Returns the stream number to which the packet belongs.

unASMFlags
   Returns the adaptive stream management (ASM) flags associated with the packet.

unASMRuleNumber
   Returns the ASM rule number of the packet.

   For More Information: See “Chapter 11: Adaptive Stream Management” beginning in
   Volume 1, on page 141.

IHXPacket::GetASMFlags
   Returns the ASM flags. The ASM flags can be either HX_ASM_SWITCH_ON or
   HX_ASM_SWITCH_OFF.
   STDMETHOD_(UINT8,GetASMFlags) ( 
      THIS
   ) PURE;

IHXPacket::GetASMRuleNumber
   Returns the ASM rule number.
   STDMETHOD_(UINT16,GetASMRuleNumber) ( 
      THIS
   ) PURE;

IHXPacket::GetBuffer
   Returns an IHXBuffer interface that manages the packet data.
   STDMETHOD_(IHXBuffer*,GetBuffer) ( 
      THIS
   ) PURE;

IHXPacket::GetStreamNumber
   Returns the stream number.
   STDMETHOD_(UINT16,GetStreamNumber) ( 
      THIS
   ) PURE;

IHXPacket::GetTime
   Returns the time, in milliseconds.
STDMETHOD_(ULONG32, GetTime) (  
  THIS 
) PURE;

IHXPacket::IsLost

Indicates whether the packet has been lost. If the return value is TRUE, the packet has been lost. If the packet is lost, none of the values of the packet has any meaning.

STDMETHOD_(BOOL, IsLost) (  
  THIS 
) PURE;

IHXPacket::Set

Sets the values of the packet. This method succeeds only if there is only one reference to the packet. That is, if more than one component has a reference to the packet, you cannot use this method to change the packet’s values.

STDMETHOD(Set) (  
  THIS 
  IHXBuffer* pBuffer,  
  UINT32 ulTime,  
  UINT16 uStreamNumber,  
  UINT8 unASMFlags,  
  UINT16 unASMRuleNumber  
) PURE;

pBuffer
  Pointer to an IHXBuffer interface that manages the packet data.

ulTime
  The time stamp for the packet.

uStreamNumber
  The stream number to which the packet belongs.

unASMFlags
  The ASM flags associated with the packet.

unASMRuleNumber
  The ASM rule number of the packet.

IHXPacket::SetAsLost

Marks a packet as lost. If the packet is lost, this method is called instead of IHXPacket::Set. This method fails if the IHXBuffer of the packet is not NULL.

STDMETHOD(SetAsLost) (  
  THIS 
) PURE;
**IHXPacketFormat**

Purpose: Sets the packet format.
Implemented by: File format plug-ins
Used by: Helix Universal Server
Header file: hxformt.h

A file format plug-in implements this interface to support packet formats in addition to Real Data Transport (RDT) and PNA. Helix Universal Server uses this interface to determine which packet formats (such as RTP) the plug-in supports. It then tells the plug-in which format to use. If the plug-in does not implement IHXPacketFormat, Helix Universal Server assumes that the plug-in supports only the RDT and PNA packet formats.

For More Information: See “Supporting Multiple Packet Formats” in Volume 1, on page 63.

The IHXPacketFormat interface contains the following methods:
- IHXPacketFormat::GetSupportedPacketFormats
- IHXPacketFormat::SetPacketFormat

As with all COM interfaces, the IHXPacketFormat interface inherits the following IUnknown methods:
- IUnknown::AddRef
- IUnknown::QueryInterface
- IUnknown::Release

**IHXPacketFormat::GetSupportedPacketFormats**

Determines the supported packet formats.

```c
STDMETHOD(GetSupportedPacketFormats) ( 
    THIS_
    REF(const char**) pFormats
) PURE;
```

`pFormats`
Returns a pointer to a pointer to the formats (such as RTP) that the plug-in supports.

**IHXPacketFormat::SetPacketFormat**

Sets the packet format for the plug-in to use.

```c
STDMETHOD(SetPacketFormat) ( 
    THIS_
    const char* pFormat
) PURE;
```

`pFormats`
A pointer to the formats, such as RTP, the plug-in should use.
IHXPacketHook

This interface is obsolete and should not be used in any new programming.

IHXPacketHook::OnEnd

This method is obsolete and should not be used in any new programming.

IHXPacketHook::OnFileHeader

This method is obsolete and should not be used in any new programming.

IHXPacketHook::OnPacket

This method is obsolete and should not be used in any new programming.

IHXPacketHook::OnStart

This method is obsolete and should not be used in any new programming.

IHXPacketHook::OnStreamHeader

This method is obsolete and should not be used in any new programming.
IHXPacketHookHelper

This interface is obsolete and should not be used in any new programming.

IHXPacketHookHelper::StartHook

This method is obsolete and should not be used in any new programming.

IHXPacketHookHelper::StopHook

This method is obsolete and should not be used in any new programming.
IHXPacketHookHelperResponse

This interface is obsolete and should not be used in any new programming.

IHXPacketHookHelperResponse::OnEndOfPackets

This method is obsolete and should not be used in any new programming.

IHXPacketHookHelperResponse::OnPacket

This method is obsolete and should not be used in any new programming.
IHXPacketHookManager

This interface is obsolete and should not be used in any new programming.

IHXPacketHookManager::CloseHook

This method is obsolete and should not be used in any new programming.

IHXPacketHookManager::InitHook

This method is obsolete and should not be used in any new programming.

IHXPacketHookManager::StartHook

This method is obsolete and should not be used in any new programming.

IHXPacketHookManager::StopHook

This method is obsolete and should not be used in any new programming.
IHXPacketHookSink

This interface is obsolete and should not be used in any new programming.

IHXPacketHookSink::StartSink

This method is obsolete and should not be used in any new programming.

IHXPacketHookSink::StopHook

This method is obsolete and should not be used in any new programming.
IHXPacketTimeOffsetHandler

Purpose: Changes a packet’s time stamp.
Implemented by: File format plug-ins, renderers
Used by: Server, SLTA
Header file: hxformt.h

There are times when an IHXPacket is received from a file format plug-in and the time stamp needs to be modified. You can query this interface from any IHXFileFormatObject interface that represents data types whose internal data is dependent on the time stamp. This interface is typically used by applications that need to shift a stream in time, like a simulated live transfer agent (SLTA) that causes multiple static files look like a single line stream. The response interface is IHXPacketTimeOffsetHandlerResponse.

The IHXPacketTimeOffsetHandler interface contains the following methods:

- IHXPacketTimeOffsetHandler::HandlePacket
- IHXPacketTimeOffsetHandler::Init
- IHXPacketTimeOffsetHandler::SetTimeOffset

As with all COM interfaces, the IHXPacketTimeOffsetHandler interface inherits the following IUnknown methods:

- IUnknown::AddRef
- IUnknown::QueryInterface
- IUnknown::Release

IHXPacketTimeOffsetHandler::HandlePacket

Provides a packet to modify for the time offset.

```c
STDMETHOD(HandlePacket) ( 
  THIS_,
  IHXPacket* pPacket
) PURE;
```

pPacket
Pointer to an IHXPacket interface that manages the packet to modify for the time offset.

IHXPacketTimeOffsetHandler::Init

Initializes the packet time offset handler and sets the response. If the plug-in supports multiple data types, implementors should look up the MIME type in the pHeader to determine what data type is being used here.

```c
STDMETHOD(Init) ( 
  THIS_,
  IHXPacketTimeOffsetHandlerResponse* pResponse,
  IHXValues* pHeader,
  IUnknown* pContext
) PURE;
```
**pResponse**
Pointer to an IHXPacketTimeOffsetHandlerResponse interface that manages the response to this interface.

**pHeader**
Pointer to an IHXValues interface that manages the stream header for the streams to be offset.

**pContext**
Pointer to the IUnknown interface that provides access to other interfaces the plug-in might need.

**IHXPacketTimeOffsetHandler::SetTimeOffset**
Sets the time offset.

```cpp
STDMETHOD(SetTimeOffset) (  
    THIS_  
    UINT32 ulTimeOffset,  
    BOOL bPlus  
) PURE;
```

**ulTimeOffset**
The time offset, in milliseconds.

**bPlus**
If TRUE, adds the amount of time specified in ulTimeOffset to the packet time stamp. If FALSE, subtracts the amount of time specified in ulTimeOffset from the packet time stamp.
IHXPacketTimeOffsetHandlerResponse

Purpose: Returns packets passed in the IHXPacketTimeOffsetHandler interface.

Implemented by: Server, SLTA

Used by: File format plug-ins, renderers

Header file: hxformt.h

The return interface through which the time offset handler returns packets with offset time stamps.

The IHXPacketTimeOffsetHandlerResponse interface contains the
IHXPacketTimeOffsetHandlerResponse::PacketReady method.

As with all COM interfaces, the IHXPacketTimeOffsetHandlerResponse interface inherits the following IUnknown methods:
• IUnknown::AddRef
• IUnknown::QueryInterface
• IUnknown::Release

IHXPacketTimeOffsetHandlerResponse::PacketReady

Passes back the packet after the packet’s time stamp has been offset correctly.

STDMETHOD(TimeOffsetPacketReady) (THIS_)

IHXPacket* pPacket
)
PURE;

pPacket

Pointer to an IHXPacket interface that manages the packet whose time stamp is being adjusted.
IHXPassiveSiteWatcher

Purpose: Provides more than one site watcher.
Implemented by: Plug-ins
Used by: IHXSite implementations
Header file: hxsite2.h

Usually the top-level client is the site watcher and manages site position and size. If a plug-in needs to know about the when the site’s size and position changes, it can watch for the change using this interface. However, the plug-in cannot override the changes; it can only watch them passively.

The IHXPassiveSiteWatcher interface contains the following methods:

- IHXPassiveSiteWatcher::PositionChanged
- IHXPassiveSiteWatcher::SizeChanged

As with all COM interfaces, the IHXPassiveSiteWatcher interface inherits the following IUnknown methods:

- IUnknown::AddRef
- IUnknown::QueryInterface
- IUnknown::Release

IHXPassiveSiteWatcher::PositionChanged

Indicates that a position change has occurred.

STDMETHOD(PositionChanged) {
    THIS_,
    HXxPoint* pPoint
} PURE;

pPoint
    Pointer to a HXxPoint structure that describes the new position.

IHXPassiveSiteWatcher::SizeChanged

Indicates that a size change has occurred.

STDMETHOD(SizeChanged) {
    THIS_,
    HXxSize* pSize
} PURE;

pSize
    Pointer to a HXxSize structure that describes the new size.
IHXPassword

Purpose: Encrypts and verifies passwords.
Implemented by: Helix architecture
Used by: Clients
Header file: hxauth.h

Used with authentication, this interface gives the client the means to encrypt passwords for transmission to Helix Universal Server.

For More Information: See “Chapter 14: Client Authentication” beginning in Volume 1, on page 189.

The IHXPassword interface contains the following methods:
• IHXPassword::AsString
• IHXPassword::AsValues
• IHXPassword::CreateBuffer
• IHXPassword::CreateValues
• IHXPassword::Crypt
• IHXPassword::Verify

As with all COM interfaces, the IHXPassword interface inherits the following IUnknown methods:
• IUnknown::AddRef
• IUnknown::QueryInterface
• IUnknown::Release

IHXPassword::AsString

Returns a string that summarizes the authentication information. The contents of the returned string depend on the value of the AuthType key that is provided in the pAuth parameter.

STDMETHOD(AsString) (THIS_ IHXValues* pAuth, REF(IHXBuffer*) pBuffer) PURE;

pAuth
Pointer to an IHXValues interface that manages the authorization type. This parameter contains a set of key/value pairs from which the authorization type is retrieved. The value of the authorization type can be:
• HX_AUTH_BASIC
• HX_AUTH_DIGEST
• HX_AUTH_NTLM
• HX_AUTH_RN5
If the authorization type is HX_AUTH_BASIC, this method will use the Response and Realm keys to determine the content of the authorization summary. If the authorization type is HX_AUTH_DIGEST, this method will use the UserName, Realm, Nonce, Uri, Response, and Opaque keys to determine the content of the authorization summary. If the authorization type is HX_AUTH_NTLM, this method will use the Response key to determine the content of the authorization summary. If the authorization type is HX_AUTH_RN5, this method will use the UserName, GUID, Realm, Nonce, and Response keys to determine the content of the authorization summary.

```
pBuffer
        Returns a pointer to an IHXBuffer interface that manages the string that contains the authorization summary.

IHXPassword::AsValues

        Takes a string and parses the string into an IHXValues map, decrypting fields if necessary.
        STDMETHOD(AsValues) ( 
            THIS_ 
            const char* str, 
            IHXValues* pValues
        ) PURE;

        str
        Pointer to the string to be parsed. In most cases, this is an RTSP header.

        pValues
        Pointer to an IHXValues interface to which the indicated string is parsed.

IHXPassword::CreateBuffer

        Returns a buffer that can be passed into IHXPassword::AsString. This method is provided for the convenience of external users, who would otherwise have to get a context and common class factory just to create the IHXBuffer interfaces. This method can be used instead, but is not advisable if other means are available.
        STDMETHOD(CreateBuffer) ( 
            THIS_ 
            REF(IHXBuffer*) pBuffer
        ) PURE;

        pBuffer
        Returns a pointer to an IHXBuffer interface that manages the string information.

IHXPassword::CreateValues

        Returns a set of values that can be passed into IHXPassword::AsValues. This method is provided for the convenience of external users, who would otherwise have to get a context and common class factory just to create IHXValues interfaces. This method can be used instead, but is not advisable if other means are available.
STDMETHOD(CreateValues) (
    THIS_
    REF(IHXValues*) pValues
) PURE;

pValues
    Returns a pointer to an IHXValues interface that manages the values.

IHXPassword::Crypt

Encrypts the user name, password and any other required authentication values using a base-64 encoding format.

STDMETHOD(Crypt) (
    THIS_
    IHXValues* pAuthentication
) PURE;

pAuthentication
    Pointer to an IHXValues interface that manages the required authentication name/value pairs.

IHXPassword::Verify

Compare two sets of authentication name/value pairs to determine if they are identical.

STDMETHOD(Verify) (
    THIS_
    IRMAValues* pAuth1,
    IRMAValues* pAuth2
) PURE;

pAuth1
    Pointer to an IHXValues interface that manages the first set of authentication name/value pairs.

pAuth2
    Pointer to an IHXValues interface that manages the second set of authentication name/value pairs to compare against the first set.
IHXPendingStatus

Purpose: Reports the status of pending operations to the Helix client.
Implemented by: File format and file system plug-ins
Used by: Helix Client
Header file: hxpends.h

File format and file system plug-ins can implement this interface to inform the Helix client of its pending operations. Operations include initializing, buffering data, contacting host, and ready. To learn the component’s status, the client calls IHXPendingStatus::GetStatus.

For More Information: See “Reporting Pending Status” in Volume 1, on page 66.

The IHXPendingStatus interface contains the IHXPendingStatus::GetStatus method.

As with all COM interfaces, the IHXPendingStatus interface inherits the following IUnknown methods:
  • IUnknown::AddRef
  • IUnknown::QueryInterface
  • IUnknown::Release

IHXPendingStatus::GetStatus

GETMETHOD(GetStatus) (  
  THIS_,
  REF(UINT16) uStatusCode,
  REF(IHXBuffer*) pStatusDesc,
  REF(UINT16) ulPercentDone
) PURE;

uStatusCode
Indicates the plug-in’s current status, which can be any of the following:

• HX_STATUS_BUFFERING
  The file system or file format plug-in is buffering data. RealPlayer uses buffering statistics from all components to calculate the buffering “percent complete” displayed in its user interface.

• HX_STATUS_CONTACTING
  The file system plug-in is contacting the host. For example, the system’s HTTP file system plug-in reports this status when contacting the HTTP host.

• HX_STATUS_INITIALIZING
  The file system or file format plug-in is initializing.

• HX_STATUS_READY
  The file system or file format plug-in ready for operation.
pStatusDesc
Returns an optional pointer to an IHXBuffer interface that manages a text description of the current status. This buffer might contain, for example, the address of a host being contacted.

ulPercentDone
Returns an integer value that represents the percentage of the task complete. This is used primarily for buffering. If the plug-in is contacting a host or initializing, the returned value is 0 when the operation is underway, 100 when complete. A percentage completed value for the HX_STATUS_READY state is ignored.
IHXPlayer

Purpose: Controls presentation playback.
Implemented by: Player object (client core)
Used by: Top-level client and rendering plug-ins
Header file: hxcore.h

This interface for the player object enables the top-level client or rendering plug-in to control the presentation playback by beginning, pausing, stopping, or seeking in a presentation timeline. Through this interface, the client or rendering plug-in can also gain access to the client engine, stream objects, and stream source objects.


The IHXPlayer interface contains the following methods:
- IHXPlayer::AddAdviseSink
- IHXPlayer::Begin
- IHXPlayer::GetClientContext
- IHXPlayer::GetClientEngine
- IHXPlayer::GetCurrentPlayTime
- IHXPlayer::GetSource
- IHXPlayer::GetSourceCount
- IHXPlayer::IsDone
- IHXPlayer::IsLive
- IHXPlayer::OpenURL
- IHXPlayer::Pause
- IHXPlayer::RemoveAdviseSink
- IHXPlayer::Seek
- IHXPlayer::SetClientContext
- IHXPlayer::Stop

As with all COM interfaces, the IHXPlayer interface inherits the following IUnknown methods:
- IUnknown::AddRef
- IUnknown::QueryInterface
- IUnknown::Release

IHXPlayer::AddAdviseSink

Adds a client advise sink.
STDMETHOD(AddAdviseSink) (  
    THIS_,  
    IHXClientAdviseSink* pAdviseSink  
) PURE;

pAdviseSink
    Pointer to an IHXClientAdviseSink interface that is being added.

IHXPlayer::Begin
    Tells the player to begin playback of all of its sources.
    STDMETHOD(Begin) (  
        THIS  
    ) PURE;

IHXPlayer::GetClientContext
    Gets the client context for this player.
    STDMETHOD(GetClientContext) (  
        THIS_,  
        REF(IUnknown*) pUnknown  
    ) PURE;

pUnknown
    Returns a pointer to a context from which you can query other interfaces.

IHXPlayer::GetClientEngine
   Gets the client engine interface of which the player is a part.
    STDMETHOD(GetClientEngine) (  
        THIS_,  
        REF(IHXClientEngine*) pEngine  
    ) PURE;

pEngine
    Returns the IHXClientEngine interface of which the player is a part.

IHXPlayer::GetCurrentPlayTime
    Gets the current time on the player time line, in milliseconds.
    STDMETHOD_(ULONG32,GetCurrentPlayTime) (  
        THIS  
    ) PURE;

IHXPlayer::GetSource
    Returns the indicated stream source instance (IHXStreamSource) used by this player.
STDMETHOD(GetSource) ( 
    THIS_ 
    UINT16 nIndex, 
    REF(IUnknown*) pUnknown 
) PURE;

nIndex
    The index number of the source instance.

pUnknown
    Returns a pointer to a context from which you can query other interfaces.

IHXPlayer::GetSourceCount

Returns the number of stream source instances currently used by this player instance.

STDMETHOD_(UINT16, GetSourceCount) ( 
    THIS 
) PURE;

IHXPlayer::IsDone

Indicates whether the player is finished playing the current presentation. If this method returns TRUE, the player has finished.

STDMETHOD_(BOOL, IsDone) ( 
    THIS 
) PURE;

IHXPlayer::IsLive

Indicates whether the player contains the live source. If this method returns TRUE, the player contains the live source.

STDMETHOD_(BOOL, IsLive) ( 
    THIS 
) PURE;

IHXPlayer::OpenURL

Opens the specified URL.

STDMETHOD(OpenURL) ( 
    THIS_ 
    const char* pURL 
) PURE;

pURL
    Pointer to the URL to be opened.

IHXPlayer::Pause

Tells the player to pause playback of all of its sources.
IHXPlayer::Pause

Removes a client advise sink.

```cpp
STDMETHOD(Pause) (THIS) PURE;
```

IHXPlayer::RemoveAdviseSink

Removes a client advise sink.

```cpp
STDMETHOD(RemoveAdviseSink) (THIS_, IHXClientAdviseSink* pAdviseSink) PURE;
```

- **pAdviseSink**: Pointer to the IHXClientAdviseSink interface to be removed.

IHXPlayer::Seek

Tells the player to seek to the specified point in the timeline of all of its sources.

```cpp
STDMETHOD(Seek) (THIS_, ULONG32 ulTime) PURE;
```

- **ulTime**: The point in the time line you want to seek to.

IHXPlayer::SetClientContext

Installs the client as the provider of client services to the core. This method is traditionally called by the top-level client application.

```cpp
STDMETHOD(SetClientContext) (THIS_, IUnknown* pUnknown) PURE;
```

- **pUnknown**: Pointer to the new context, provided by the top-level client, that overrides the player’s own context.

IHXPlayer::Stop

Tells the player to stop playback of all of its sources.

```cpp
STDMETHOD(Stop) (THIS) PURE;
```
IHXPlayer2

Purpose: Controls presentation playback.
Implemented by: Player object (client core)
Used by: Top-level client and rendering plug-ins
Header file: hxcore.h

This interface for the player object enables the top-level client or rendering plug-in to control the presentation playback by managing specified request interfaces and the minimum preroll. The methods in this interface are provided as an addition to the IHXPlayer interface.

For More Information: See “Chapter 13: Rendering Plug-In” and “Chapter 12: Top-Level Client”. See also IHXPlayer in Volume 2, on page 262.

The IHXPlayer2 interface contains the following methods:
- IHXPlayer2::GetMinimumPreroll
- IHXPlayer2::GetRequest
- IHXPlayer2::OpenRequest
- IHXPlayer2::SetMinimumPreroll

As with all COM interfaces, the IHXPlayer2 interface inherits the following IUnknown methods:
- IUnknown::AddRef
- IUnknown::QueryInterface
- IUnknown::Release

IHXPlayer2::GetMinimumPreroll

Gets the minimum preroll of this clip.

STDMETHOD(GetMinimumPreroll) (  
    THIS__
    REF(UINT32) ulMinPreroll
) PURE;

ulMinPreroll
    Returns the minimum preroll.

IHXPlayer2::GetRequest

Gets the specified IHXRequest interface so the user can get information such as MIME type, a requested URL, and so on.

STDMETHOD(GetRequest) (  
    THIS__
    REF(IHXRequest*) pRequest
) PURE;

pRequest
    Returns a pointer to the IHXRequest interface.
**IHXPlayer2::OpenRequest**

Opens the specified IHXRequest interface to start playback.

```cpp
STDMETHOD(OpenRequest) ( 
    IHXRequest* pRequest 
) PURE;

pRequest
    Pointer to the IHXRequest interface to be opened.
```

**IHXPlayer2::SetMinimumPreroll**

Sets the minimum preroll of this clip.

```cpp
STDMETHOD(SetMinimumPreroll) ( 
    UINT32 ulMinPreroll 
) PURE;

ulMinPreroll
    The minimum preroll, in milliseconds.
```
IHXPlayerCreationSink

Purpose: Provides notification of player creation and destruction.
Implemented by: Plug-in components
Used by: Plug-in components
Header file: hxpsink.h

A plug-in implements this interface to get notifications of player creation and destruction. It should use IHXPlayerSinkControl, implemented by the Helix client, to add itself as a sink.

The IHXPlayerCreationSink interface contains the following methods:
- IHXPlayerCreationSink::PlayerClosed
- IHXPlayerCreationSink::PlayerCreated

As with all COM interfaces, the IHXPlayerCreationSink interface inherits the following IUnknown methods:
- IUnknown::AddRef
- IUnknown::QueryInterface
- IUnknown::Release

IHXPlayerCreationSink::PlayerClosed

Indicates that an existing player was closed.

STDMETHOD(PlayerClosed) ( 
    THIS_,
    IHXPlayer* pPlayer
) PURE;

pPlayer
    Pointer to an IHXPlayer interface that managed the player that was closed.

IHXPlayerCreationSink::PlayerCreated

Indicates that a new player was created.

STDMETHOD(PlayerCreated) ( 
    THIS_,
    IHXPlayer* pPlayer
) PURE;

pPlayer
    Pointer to an IHXPlayer interface that manages the new player.
IHXPlayerConnectionAdviseSink

Purpose: Responds to events in the life of a player.
Implemented by: Allowance plug-ins
Used by: Server core
Header file: hxallow.h

This interface’s methods are called in response to events in the life of a player. For example, the IHXPlayerConnectionAdviseSink::OnConnection method is called when a new player connects to the server.


The IHXPlayerConnectionAdviseSink interface contains the following methods:

• IHXPlayerConnectionAdviseSink::OnBegin
• IHXPlayerConnectionAdviseSink::OnConnection
• IHXPlayerConnectionAdviseSink::OnDone
• IHXPlayerConnectionAdviseSink::OnPause
• IHXPlayerConnectionAdviseSink::OnStop
• IHXPlayerConnectionAdviseSink::OnURL
• IHXPlayerConnectionAdviseSink::SetPlayerController
• IHXPlayerConnectionAdviseSink::SetRegistryID

As with all COM interfaces, the IHXPlayerConnectionAdviseSink interface inherits the following IUnknown methods:

• IUnknown::AddRef
• IUnknown::QueryInterface
• IUnknown::Release

IHXPlayerConnectionAdviseSink::OnBegin

Indicates the client has started or resumed (that is, an RTSP PLAY method has been received). This method is called while the client is connected to the server.

STDMETHOD(OnBegin) (THIS) PURE;

IHXPlayerConnectionAdviseSink::OnConnection

Indicates that a new player has connected to the server (that is, an RTSP OPTIONS method has been received). If the result is HXR_OK, then the plug-in is notified when certain events occur in the player’s life cycle. This method is called when a client first connects to the server.

STDMETHOD(OnConnection) (THIS,
           IHXPlayerConnectionResponse* pResponse
    ) PURE;
pResponse

Pointer to an IHXPlayerConnectionResponse interface that manages the response to various methods
in this interface.

IHXPlayerConnectionAdviseSink::OnDone

Indicates that the client has disconnected from the server (that is, an RTSP TEARDOWN method
request has been received).

STDMETHOD(OnDone) (THIS

) PURE;

IHXPlayerConnectionAdviseSink::OnPause

Indicates that playback has paused (that is, an RTSP PAUSE method has been received). This method is
called while the client is connected to the server.

STDMETHOD(OnPause) (THIS

) PURE;

IHXPlayerConnectionAdviseSink::OnStop

Indicates that playback has stopped (that is, an RTSP TEARDOWN method request has been received).
This method is called while the client is connected to the server.

STDMETHOD(OnStop) (THIS

) PURE;

IHXPlayerConnectionAdviseSink::OnURL

Indicates that the client is requesting a URL (that is, an RTSP DESCRIBE method has been received).
This method is called when a client first connects to the server.

STDMETHOD(OnURL) (THIS_

IHXRequest* pRequest

) PURE;

pRequest

Pointer to an IHXRequest interface that manages the URL name and headers.

IHXPlayerConnectionAdviseSink::SetPlayerController

Provides an interface that can stop, alert, redirect, or otherwise control the client. This method is called
when the client first connects to the server.

STDMETHOD(SetPlayerController) (THIS_

IHXPlayerController* pPlayerController

) PURE;
pPlayerController
Pointer to an IHXPlayerController interface that controls the client.

IHXPlayerConnectionAdviseSink::SetRegistryID
Provides the plug-in with the identification of this client in the server registry. The plug-in can then use this identification to find out information about the connected client. This method is called when a client first connects to the server.

STDMETHOD(SetRegistryID) (
    THIS_ 
    UINT32 ulPlayerRegistryID 
) PURE;

ulPlayerRegistryID
The registry ID of this client.
IHXPlayerConnectionAdviseSinkManager

Purpose: Manages the creation of IHXPlayerConnectionAdviseSink objects.

Implemented by: Server core

Used by: Allowance plug-ins

Header file: hxallow.h

This interface manages the creation of IHXPlayerConnectionAdviseSink objects.


The IHXPlayerConnectionAdviseSinkManager interface contains the
IHXPlayerConnectionAdviseSinkManager::CreatePlayerConnectionAdviseSink method.

As with all COM interfaces, the IHXPlayerConnectionAdviseSinkManager interface inherits the following IUnknown methods:

• IUnknown::AddRef
• IUnknown::QueryInterface
• IUnknown::Release

IHXPlayerConnectionAdviseSinkManager::CreatePlayerConnectionAdviseSink

Creates a player connection advise sink interface.

STDMETHOD(CreatePlayerConnectionAdviseSink) (THIS_ REF(IHXPlayerConnectionAdviseSink*) pPCAdviseSink ) PURE;

pPCAdviseSink

Returns a pointer to the IHXPlayerConnectionAdviseSink interface that was created.
**IHXPlayerConnectionResponse**

**Purpose:** Permits an allowance plug-in to communicate with the server when the plug-in is finished with a notification.

**Implemented by:** Server core

**Used by:** Allowance plug-ins

**Header file:** hxallow.h

This interface permits an allowance plug-in to tell the server when the plug-in is finished with a particular IHXPlayerConnectionAdviseSink notification.


The IHXPlayerConnectionResponse interface contains the following methods:

- IHXPlayerConnectionResponse::OnBeginDone
- IHXPlayerConnectionResponse::OnConnectionDone
- IHXPlayerConnectionResponse::OnPauseDone
- IHXPlayerConnectionResponse::OnStopDone
- IHXPlayerConnectionResponse::OnURLDone

As with all COM interfaces, the IHXPlayerConnectionResponse interface inherits the following IUnknown methods:

- IUnknown::AddRef
- IUnknown::QueryInterface
- IUnknown::Release

**IHXPlayerConnectionResponse::OnBeginDone**

Contains the response to the IHXPlayerConnectionAdviseSink::OnBegin method and indicates that the original call was completed.

```cpp
STDMETHOD(OnBeginDone) (THIS_ HX_RESULT status) PURE;
```

**status**

Contains the response status. If this parameter is HXR_OK, the call to IHXPlayerConnectionAdviseSink::OnBegin completed successfully.

**IHXPlayerConnectionResponse::OnConnectionDone**

Contains the response to the IHXPlayerConnectionAdviseSink::OnConnection method and indicates that the original call was completed.

```cpp
STDMETHOD(OnConnectionDone) (THIS_ HX_RESULT status) PURE;
```
status
Contains the response status. If this parameter is HXR_OK, the call to
IHXPlayerConnectionAdviseSink::OnConnection completed successfully.

IHXPlayerConnectionResponse::OnPauseDone
Contains the response to the IHXPlayerConnectionAdviseSink::OnPause method and indicates that the
original call was completed.
STDMETHOD(OnPauseDone) ( 
    THIS_ 
    HX_RESULT status 
) PURE;

status
Contains the response status. If this parameter is HXR_OK, the call to
IHXPlayerConnectionAdviseSink::OnPause completed successfully.

IHXPlayerConnectionResponse::OnStopDone
Contains the response to the IHXPlayerConnectionAdviseSink::OnStop method and indicates that the
original call was completed.
STDMETHOD(OnStopDone) ( 
    THIS_ 
    HX_RESULT status 
) PURE;

status
Contains the response status. If this parameter is HXR_OK, the call to
IHXPlayerConnectionAdviseSink::OnStop completed successfully.

IHXPlayerConnectionResponse::OnURLDone
Contains the response to the IHXPlayerConnectionAdviseSink::OnURL method and indicates that the
original call was completed.
STDMETHOD(OnURLDone) ( 
    THIS_ 
    HX_RESULT status 
) PURE;

status
Contains the response status. If this parameter is HXR_OK, the call to
IHXPlayerConnectionAdviseSink::OnURL completed successfully.
IHXPlayerController

Purpose: Controls a player.
Implemented by: Server core
Used by: Allowance plug-ins
Header file: hxallow.h

This interface permits an allowance plug-in to control a player by directing the server to alert, redirect, or disconnect the player.


The IHXPlayerController interface contains the following methods:

- IHXPlayerController::AlertAndDisconnect
- IHXPlayerController::Disconnect
- IHXPlayerController::HostRedirect
- IHXPlayerController::NetworkRedirect
- IHXPlayerController::Pause
- IHXPlayerController::Redirect
- IHXPlayerController::Resume

As with all COM interfaces, the IHXPlayerController interface inherits the following IUUnknown methods:

- IUUnknown::AddRef
- IUUnknown::QueryInterface
- IUUnknown::Release

IHXPlayerController::AlertAndDisconnect

Displays a message on the player and then disconnects the player.

STDMETHOD(AlertAndDisconnect) (THIS_ IHXBuffer* pAlert) PURE;

pAlert
    Pointer to an IHXBuffer interface that manages the message to display on the player.

IHXPlayerController::Disconnect

Disconnects the player.

STDMETHOD(Disconnect) (THIS) PURE;
**IHXPlayerController::HostRedirect**

Redirects this player to another host and/or port, for the same URL. This method works with both RTSP and PNA protocols.

```c
STDMETHOD(HostRedirect) (  
    THIS_  
    IHXBuffer* pHost,  
    UINT16 nPort  
) PURE;
```

- `pHost`  
  Pointer to an IHXBuffer interface that manages the host name to which this player is redirected.

- `nPort`  
  The port number to which this player is redirected.

**IHXPlayerController::NetworkRedirect**

Redirects this player to another URL. This method is only redirects an RTSP player connection to another RTSP URL.

```c
STDMETHOD(NetworkRedirect) (  
    THIS_  
    IHXBuffer* pURL,  
    UINT32 ulSecsFromNow  
) PURE;
```

- `pURL`  
  Pointer to an IHXBuffer interface that manages the URL to which this player is redirected.

- `ulSecsFromNow`  
  The number of seconds that must elapse before the redirection occurs.

**IHXPlayerController::Pause**

Alerts the player to pause.

```c
STDMETHOD(Pause) (  
    THIS  
) PURE;
```

**IHXPlayerController::Redirect**

Redirects this player to another URL on the same server. For example, if `pPartialURL` were set to “welcome.rm”, the player would be redirected to “current_protocol://current_host:current_port/welcome.rm”. This method works with both the RTSP and PNA protocols.

```c
STDMETHOD(Redirect) (  
    THIS_  
    IHXBuffer* pPartialURL  
) PURE;
```
pPartialURL
Pointer to an IHXBuffer interface that manages the partial URL.

IHXPlayerController::Resume
Alerts the player to resume.

STDMETHOD(Resume) ( 
  THIS 
) PURE;
**IHXPlayerControllerProxyRedirect**

**Purpose:** Provides proxy redirection.

**Implemented by:** Server core

**Used by:** Server allowance plug-in

**Header file:** hxallow.h

This interface makes it possible for “305” proxy redirects to be issued, as documented in the RTSP specification. This interface is queried from IHXPlayerController.

The IHXPlayerControllerProxyRedirect interface contains the IHXPlayerControllerProxyRedirect::NetworkProxyRedirect method.

As with all COM interfaces, the IHXPlayerControllerProxyRedirect interface inherits the following IUnknown methods:

- IUnknown::AddRef
- IUnknown::QueryInterface
- IUnknown::Release

**IHXPlayerControllerProxyRedirect::NetworkProxyRedirect**

Issues a proxy redirect that specifies the proxy to be used. You can use this method only between the IHXPlayerConnectionAdviseSink::OnURL and IHXPlayerConnectionResponse::OnURLDone methods. Note that this method works only with RTSP connections.

```cpp
STDMETHOD(NetworkProxyRedirect) ( 
    THIS_ 
    IHXBuffer* pURL 
) PURE;
```

**pURL**

Pointer to an IHXBuffer interface that manages the URL. This URL is just a hostname:port pair—for example, “rtsp://audio.real.com:554/’’.
IHXPlayerNavigator

Purpose: Navigates through player objects.
Implemented by: Client core
Used by: Top-level client
Header file: hxcore.h

The IHXPlayerNavigator interface contains the following methods:

- IHXPlayerNavigator::AddChildPlayer
- IHXPlayerNavigator::GetChildPlayer
- IHXPlayerNavigator::GetNumChildPlayer
- IHXPlayerNavigator::GetParentPlayer
- IHXPlayerNavigator::RemoveChildPlayer
- IHXPlayerNavigator::RemoveParentPlayer
- IHXPlayerNavigator::SetParentPlayer

As with all Component Object Model (COM) interfaces, the IHXPlayerNavigator interface inherits the following IUnknown methods:

- IUnknown::AddRef
- IUnknown::QueryInterface
- IUnknown::Release

IHXPlayerNavigator::AddChildPlayer

Adds a child player to the current player.

HRESULT AddChildPlayer(IHXPlayer* pPlayer)

pPlayer Pointer to an IHXPlayer interface that manages the child player.

IHXPlayerNavigator::GetChildPlayer

Gets the indicated child player.

HRESULT GetChildPlayer(UINT16 uPlayerIndex, IHXPlayer** pPlayer)

uPlayerIndex The index number of the child player.

pPlayer Returns a pointer to the IHXPlayer interface that manages the specified child player.
IHXPlayerNavigator::GetNumChildPlayer

Returns the total number of child players.

```c
STDMETHOD_(UINT16, GetNumChildPlayer) (THIS) PURE;
```

IHXPlayerNavigator::GetParentPlayer

Gets the parent player.

```c
STDMETHOD(GetParentPlayer) (THIS, REF(IHXPlayer*) pPlayer) PURE;
pPlayer
  Returns a pointer to the IHXPlayer interface that manages the parent player.
```

IHXPlayerNavigator::RemoveChildPlayer

Removes the specified child player from the current player.

```c
STDMETHOD(RemoveChildPlayer) (THIS, IHXPlayer* pPlayer) PURE;
pPlayer
  Pointer to the IHXPlayer interface that manages the child player to be removed.
```

IHXPlayerNavigator::RemoveParentPlayer

Removes the parent player.

```c
STDMETHOD(RemoveParentPlayer) (THIS, IHXPlayer* pPlayer) PURE;
pPlayer
  Pointer to an IHXPlayer interface that identifies the parent player to be removed.
```

IHXPlayerNavigator::SetParentPlayer

Sets the parent player.

```c
STDMETHOD(SetParentPlayer) (THIS, IHXPlayer* pPlayer) PURE;
pPlayer
  Pointer to the IHXPlayer interface that manages the parent player.
```
IHXPlayerSinkControl

Purpose: Adds sinks to get notifications of player creation and destruction.
Implemented by: Helix client
Used by: Plug-in components
Header file: hxpsink.h

A plug-in that needs to be notified of player creation and destruction should query for this interface from the context passed in IHXPlugin::InitPlugin and add themselves as an IHXPlayerCreationSink.

The IHXPlayerSinkControl interface contains the following methods:
  • IHXPlayerSinkControl::AddSink
  • IHXPlayerSinkControl::RemoveSink

As with all COM interfaces, the IHXPlayerSinkControl interface inherits the following IUnknown methods:
  • IUnknown::AddRef
  • IUnknown::QueryInterface
  • IUnknown::Release

IHXPlayerSinkControl::AddSink

Adds a new player creation sink for notification of player creation and destruction.

STDMETHOD(AddSink) (THIS_ IHXPlayerCreationSink* pSink) PURE;

pSink
  Pointer to an IHXPlayerCreationSink interface that manages the new sink.

IHXPlayerSinkControl::RemoveSink

Removes an existing player creation sink.

STDMETHOD(RemoveSink) (THIS_ IHXPlayerCreationSink* pSink) PURE;

pSink
  Pointer to an IHXPlayerCreationSink interface to remove.
**IHXPlugin**

**Purpose:** Provides plug-in characteristics.

**Implemented by:** All plug-ins

**Used by:** Helix Universal Server and Helix client

**Header file:** hxplugn.h

Every plug-in must implement IHXPlugin. On startup, Helix Universal Server or the client uses the IHXPlugin::GetPluginInfo method to determine each of its plug-in’s characteristics. A “C-style” entry point in a plug-in, HXCreateInstance, exposes this interface.

**For More Information:** For more on this entry point, see “Creating a Plug-in Instance” in Volume 1, on page 36.

The IHXPlugin interface contains the following methods:

- IHXPlugin::GetPluginInfo
- IHXPlugin::InitPlugin

As with all COM interfaces, the IHXPlugin interface inherits the following IUnknown methods:

- IUnknown::AddRef
- IUnknown::QueryInterface
- IUnknown::Release

**IHXPlugin::GetPluginInfo**

Returns the basic information about this plug-in. Most of this information is used in the About box in the user interface. This method is called when the Helix Architecture (HX) core application is started.

```idl
STDMETHOD(GetPluginInfo) ( 
    THIS, 
    REF(BOOL) bMultipleLoad, 
    REF(const char*) pDescription, 
    REF(const char*) pCopyright, 
    REF(const char*) pMoreInfoURL, 
    REF(ULONG32) ulVersionNumber 
) PURE;
```

**bMultipleLoad**

Indicates whether or not this plug-in can be instantiated multiple times. All file formats must set this parameter to TRUE. The only other type of plug-in that can specify this parameter as TRUE is a file system plug-in. Any plug-in that sets this parameter to TRUE must not use global variables of any type.

Setting this parameter to TRUE implies that you accept that your plug-in may be instantiated multiple times (possibly in different address spaces). Plug-ins are instantiated multiple times only in the server (for performance reasons).

An example of a plug-in that must set this parameter to FALSE is a file system plug-in that uses a single TCP connection to communicate with a database.
pDescription
Returns a pointer to a text description of the plug-in. This parameter can be NULL.

pCopyright
Returns a pointer to a text copyright notice. This parameter can be NULL.

pMoreInfoURL
Returns a pointer to a text URL location where more information can be obtained. This parameter can be NULL.

ulVersionNumber
Returns the version of this plug-in.

**IHXPlugin::InitPlugin**

Initializes the plug-in for use. This method must always be called before any other method is called. This method is primarily needed so the plug-in can have access to the context for creation of IHXBuffer interfaces and IMalloc.

```cpp
STDMETHOD(InitPlugin) ( 
    THIS_ 
    IUknown* pContext 
) PURE;
```

pContext
Pointer to an IUknown interface that provides the plug-in with access to other useful interfaces.
IHXPlugin2Handler

Purpose: Provides for the discovery of plug-ins.
Implemented by: Client core
Used by: Client core, file format plug-ins, renderers
Header file: hxplugn.h

This interface enables you to find plug-ins based on a description of functionality. Upon calling IHXPlugin2Handler::Init, the plug-in handler should scan a given directory for all DLLs that export an HXCreateInstance (this particular implementation scans the directory specified in DT_Plugins). The DLLs that export this interface are termed “plugins.” Those plug-ins will then be queried for a number of parameters, and those parameters will be stored. Later, components of the system can then ask the plug-in handler for plug-ins that fit a particular description (for example, find a file system that supports the “file:” protocol).

The IHXPlugin2Handler interface contains the following methods:

- IHXPlugin2Handler::Close
- IHXPlugin2Handler::FindImplementationFromClassID
- IHXPlugin2Handler::FindIndexUsingStrings
- IHXPlugin2Handler::FindIndexUsingValues
- IHXPlugin2Handler::FindPluginUsingStrings
- IHXPlugin2Handler::FindPluginUsingValues
- IHXPlugin2Handler::FlushCache
- IHXPlugin2Handler::GetInstance
- IHXPlugin2Handler::GetNumPlugins2
- IHXPlugin2Handler::GetPluginInfo
- IHXPlugin2Handler::Init
- IHXPlugin2Handler::SetCacheSize
- IHXPlugin2Handler::SetRequiredPlugins

As with all COM interfaces, the IHXPlugin2Handler interface inherits the following IUnknown methods:

- IUnknown::AddRef
- IUnknown::QueryInterface
- IUnknown::Release

IHXPlugin2Handler::Close

Unloads all DLLs that are not being used. Returns HXR_FAIL if there are still plug-ins active in the system.

STDMETHOD(Close) (THIS ) PURE;

Note: This method is not currently implemented.
**IHXPlugin2Handler::FindImplementationFromClassID**

Scans all plug-ins that support the IHXCommonClassFactory interface; determines which, if any, of these plug-ins support the creation of the GUID specified in GUIDClassID; and returns an instance of this class.

```cpp
STDMETHOD(FindImplementationFromClassID) (
    THIS_ 
    REFGUID GUIDClassID,
    REF(IUnknown*) pIUnknownInstance
) PURE;
```

**GUIDClassID**

The class ID the plug-in must support.

**pIUnknownInstance**

Returns a pointer to a plug-in instance that supports the given class ID.

---

**IHXPlugin2Handler::FindIndexUsingStrings**

Finds a plug-in that matches the given list of criteria. An index is returned that you can use to either get more information about the plug-in (using IHXPlugin2Handler::GetPluginInfo) or create an instance using IHXPlugin2Handler::GetInstance. For example:

```cpp
UINT32 index
pPluginhandler->FindIndexUsingStrings(“PluginType”, “PLUGIN_FILE_SYSTEM”,
    “FileProtocol”, “file”, NULL, NULL, index);
```

does the system to find a plug-in that is a file system that supports the file protocol. The index of that plug-in (probably smplfsys) is returned in the index.

**Note:** A maximum of three values can be given.

```cpp
STDMETHOD(FindIndexUsingStrings) ( 
    THIS_ 
    char* PropName1,
    char* PropVal1,
    char* PropName2,
    char* PropVal2,
    char* PropName3,
    char* PropVal3,
    REF(UINT32) unIndex
) PURE;
```

**PropName1**

Pointer to the first property name.

**PropVal1**

Pointer to the value of the first property.

**PropName2**

Pointer to the second property name.

**PropVal2**

Pointer to the value of the second property.
PropName3
  Pointer to the third property name.

PropVal3
  Pointer to the value of the third property.

unIndex
  Returns the index number of the plug-in that matches the given values.

IHXPlugin2Handler::FindIndexUsingValues

Finds a plug-in that matches a given list of values. An index is returned that you can use to either get
the values (using IHXPlugin2Handler::GetPluginInfo) or create an instance using
IHXPlugin2Handler::GetInstance. The IHXPlugin2Handler::FindIndexUsingStrings method only provides slots
for three property strings. Using this method, you can include as many strings as you need by using an
IHXValues interface to facilitate more exacting searches.

STDMETHOD(FindIndexUsingValues) (n)
  THIS_
  IHXValues* pValue,
  REF(UINT32) unIndex
) PURE;

pValue
  Pointer to an IHXValues interface that manages the values of the plug-in you want to find.

unIndex
  Returns the index number of the plug-in that matches the given values.

IHXPlugin2Handler::FindPluginUsingStrings

Finds a plug-in that matches the given set of values.

Note: A maximum of three values can be given.

STDMETHOD(FindPluginUsingStrings) (n)
  THIS_
  char* PropName1,
  char* PropVal1,
  char* PropName2,
  char* PropVal2,
  char* PropName3,
  char* PropVal3,
  REF(IUnknown*) pUnk
) PURE;

PropName1
  Pointer to the first property name.

PropVal1
  Pointer to the value of the first property.

PropName2
  Pointer to the second property name.
PropVal2
Pointer to the value of the second property.

PropName3
Pointer to the third property name.

PropVal3
Pointer to the value of the third property.

pUnk
Returns a pointer to the plug-in instance that matches the given values.

IHXPlugin2Handler::FindPluginUsingValues
Finds a plug-in that matches the given set of values.

```cpp
STDMETHOD(FindPluginUsingValues) ( 
    THIS_IHXValues* pValue,
    REF(IUnknown*) pUnk
) PURE;
```

pValue
Pointer to an IHXValues interface that manages the values to match.

pUnk
Returns a pointer to the plug-in instance that matches the given values.

IHXPlugin2Handler::FlushCache
Flushes the Least Recently Used (LRU) cache, which unloads all DLLs from memory that currently have a reference count of 0 (zero).

```cpp
STDMETHOD(FlushCache) ( 
    THIS
) PURE;
```

IHXPlugin2Handler::GetInstance
Returns a plug-in instance given a plug-in index.

```cpp
STDMETHOD(GetInstance) ( 
    THIS_UINT32 index,
    REF(IUnknown*) pUnknown
) PURE;
```

index
The index number of the plug-in.

pUnknown
Returns a pointer to the new instance of the plug-in specified by the index.
IHXPlugin2Handler::GetNumPlugins2

Returns the number of plug-ins.

STDMETHOD_(ULONG32, GetNumOfPlugins2) (THIS) PURE;

IHXPlugin2Handler::GetPluginInfo

Gets the information about the specified plug-in.

STDMETHOD(GetPluginInfo) (THIS_ UINT32 unIndex, REF(IHXValues*) Values) PURE;

unIndex

The index number of the specified plug-in.

Values

Returns a pointer to an IHXValues interface that manages the plug-in information. The information is stored as name and value pairs. For plug-ins that implement the IHXGenericPlugin interface, this information is determined completely by the implementor.

IHXPlugin2Handler::Init

Initializes the plug-in handler.

STDMETHOD(Init) (THIS_ IUnknown* pContext) PURE;

pContext

Pointer to the context passed into plug-ins when they are initialized by the plug-in handler.

IHXPlugin2Handler::SetCacheSize

Sets the size of the cache. The cache is initially set to 1000 KB. To disable the cache, set the size to 0 (zero). If the cache is disabled, a DLL will be unloaded whenever its reference count becomes 0 (which might cause performance problems).

STDMETHOD(SetCacheSize) (THIS_ ULONG32 nSizeKB) PURE;

nSizeKB

The size, in KB, to set the cache.
**IHXPlugin2Handler::SetRequiredPlugins**

Sets the required plug-in list. This information is used by the server; those plug-ins specified as “required plug-ins” must be present in their original form for the server to function.

```c
STDMETHOD(SetRequiredPlugins) ( 
    THIS_
    const char** ppszRequiredPlugins 
) PURE;
```

*ppszRequiredPlugins*

Returns a pointer to the list of required plug-ins.
IHXPluginEnumerator

Purpose: Enumerates and returns plug-in instances.
Implemented by: Client and server core
Used by: Client and server applications
Header file: hxplugn.h

This interface finds all plug-ins supported by the client or server core. It uses IHXPluginGroupEnumerator to find plug-ins that expose a specific interface.


The IHXPluginEnumerator interface contains the following methods:
- IHXPluginEnumerator::GetNumOfPlugins
- IHXPluginEnumerator::GetPlugin

As with all COM interfaces, the IHXPluginEnumerator interface inherits the following IUnknown methods:
- IUnknown::AddRef
- IUnknown::QueryInterface
- IUnknown::Release

IHXPluginEnumerator::GetNumOfPlugins

Returns the total number of plug-ins available to the client or server core.

```cpp
STDMETHOD_(ULONG32, GetNumOfPlugins) (THIS)
```

IHXPluginEnumerator::GetPlugin

Returns an instance of the plug-in.

```cpp
STDMETHOD(GetPlugin) (THIS_
    ULONG32 ulIndex,
    REF(IUnknown*) pPlugin)
```

ulIndex

Index number of the plug-in.

pPlugin

Returns a pointer to the IUnknown interface that identifies the instance of the plug-in object, so calls can be made to that plug-in’s interfaces.
**IHXPluginFactory**

**Purpose:** Provides for more than one Helix architecture plug-in in a single DLL.

**Implemented by:** Plug-ins

**Used by:** Client core and server

**Header file:** hxplugn.h

This interface provides for more than one Helix architecture plug-in in a single DLL. For example, a plug-in author could use this interface to have three different file format plug-ins in a single DLL.

The IHXPluginFactory interface contains the following methods:

- IHXPluginFactory::GetNumPlugins
- IHXPluginFactory::GetPlugin

As with all COM interfaces, the IHXPluginFactory interface inherits the following IUnknown methods:

- IUnknown::AddRef
- IUnknown::QueryInterface
- IUnknown::Release

**IHXPluginFactory::GetNumPlugins**

Returns the number of plug-ins in the DLL.

```cpp
STDMETHOD_(UINT16, GetNumPlugins) (THIS)  PURE;
```

**IHXPluginFactory::GetPlugin**

Creates a new instance of the requested plug-in.

```cpp
STDMETHOD(GetPlugin) (THIS_ UINT16 uIndex, IUnknown** pPlugin ) PURE;
```

- **uIndex**
  The index number of the plug-in.

- **pPlugin**
  Returns a pointer to the IUnknown interface that identifies the instance of the plug-in object indicated by uIndex.
IHXPluginGroupEnumerator

Purpose: Enumerates and returns plug-ins supporting a particular interface.

Implemented by: Client and server core

Used by: Client and server applications

Header file: hxplugn.h

This interface finds all plug-ins to the client or server core that expose a specific interface. It uses IHXPluginEnumerator to find all plug-ins to the core.


The IHXPluginGroupEnumerator interface contains the following methods:
- IHXPluginGroupEnumerator::GetNumOfPlugins
- IHXPluginGroupEnumerator::GetPlugin
- IHXPluginGroupEnumerator::Init

As with all COM interfaces, the IHXPluginGroupEnumerator interface inherits the following IUnknown methods:
- IUnknown::AddRef
- IUnknown::QueryInterface
- IUnknown::Release

IHXPluginGroupEnumerator::GetNumOfPlugins

Returns the number of available plug-ins that support a particular interface.

STDMETHOD_(ULONG32, GetNumOfPlugins) (THIS ) PURE;

IHXPluginGroupEnumerator::GetPlugin

Creates a new instance of the requested plug-in.

STDMETHOD(GetPlugin) (THIS _
UINT32 ulIndex,
REF(IUnknown*) pPlugin ) PURE;

ulIndex
   Index number of the plug-in.

pPlugin
   Returns a pointer to the IUnknown interface that identifies the instance of the plug-in object indicated by ulIndex.
IHXPluginGroupEnumerator::Init

Builds a list of plug-ins that support the specified interface. You must call this method before you can call the other IHXPluginGroupEnumerator methods.

STDMETHOD(Init) (  
    THIS_  
    REFIID iid  
) PURE;

iid  
The interface into which the plug-ins are grouped.
This interface provides a means for plug-ins to return a list of properties that uniquely describes the plug-in to the plug-in handler.

The IHXPluginProperties interface contains the IHXPluginProperties::GetProperties method.

As with all COM interfaces, the IHXPluginProperties interface inherits the following IUnknown methods:

• IUnknown::AddRef
• IUnknown::QueryInterface
• IUnknown::Release

### IHXPluginProperties::GetProperties

Returns a list of properties that uniquely describe the specified plug-in.

```cpp
STDMETHOD(GetProperties) (THIS_ 
     REF(IHXValues*) pIHXValuesProperties
) PURE

pIHXValuesProperties
     Returns a pointer to an IHXValues interface that manages the property information.
```
This interface enables top-level clients to inform the client core to look for new plug-ins. Currently this interface is used by RealPlayer to implement auto-upgrade.

The IHXPluginReloader interface contains the IHXPluginReloader::ReloadPlugins method.

As with all COM interfaces, the IHXPluginReloader interface inherits the following IUnknown methods:

- IUnknown::AddRef
- IUnknown::QueryInterface
- IUnknown::Release

**IHXPluginReloader::ReloadPlugins**

Causes the client core to reload all plug-ins.

```cpp
STDMETHOD(ReloadPlugins) ( THIS ) PURE;
```
IHXXPVDatabase

Purpose: Provides access to a pay-per-view database.
Implemented by: Pay-per-view database plug-in
Used by: File system and pay-per-view allowance plug-ins
Header file: hxppv.h

You can use this interface to create a database plug-in to store pay-per-view data.

The IHXXPVDatabase interface contains the following methods:

- IHXXPVDatabase::DeductTime
- IHXXPVDatabase::GetPasswordFromUserid
- IHXXPVDatabase::GetPPVDInfo
- IHXXPVDatabase::GetRedirect
- IHXXPVDatabase::GetUserIdFromGUID
- IHXXPVDatabase::GrantPermission
- IHXXPVDatabase::GrantTime
- IHXXPVDatabase::InitPPVD
- IHXXPVDatabase::InsertUser
- IHXXPVDatabase::LogAccess
- IHXXPVDatabase::LogReg
- IHXXPVDatabase::PutRedirect
- IHXXPVDatabase::RegisterGUID
- IHXXPVDatabase::RemoveUser
- IHXXPVDatabase::RevokeAllPermissions
- IHXXPVDatabase::RevokePermission
- IHXXPVDatabase::SetPassword
- IHXXPVDatabase::ValidateUser

As with all COM interfaces, the IHXXPVDatabase interface inherits the following IUnknown methods:

- IUnknown::AddRef
- IUnknown::QueryInterface
- IUnknown::Release

IHXXPVDatabase::DeductTime

Deducts a user time to view. This method is obsolete and should not be used in any new applications.

STDMETHOD(DeductTime) (const char* pUserid, const PPVPermission* pPPVPermission, UINT32 ulDeduct) PURE;
pUserid
   Pointer to the user ID key of the record.

pPPVPermission
   Pointer to a PPVPermission structure that contains the permission information and URL.

ulDeduct
   The amount of time to deduct.

IHXXPPVDatabase::GetPasswordFromUserid
   Gets a user’s password.
   STDMETHOD(GetPasswordFromUserid) ( 
      THIS_ 
      const char* pUserid, 
      char* pCipherPassword, 
      UINT32 ulCipherPasswordLen 
   ) PURE;

   pUserid
      Pointer to the user ID key of the record.

   pCipherPassword
      Pointer to the password to retrieve.

   ulCipherPasswordLen
      The maximum length of the password in pCipherPassword.

IHXXPPVDatabase::GetPPVDBInfo
   Gets this database's short name. This unique identifier identifies this database plug-in, and should be unique enough that no other plug-in will ever have the same plug-in name.
   STDMETHOD(GetPPVDBInfo) ( 
      THIS_ 
      REF(const char*) pShortName 
   ) PURE;

   pShortName
      Returns a pointer to the short name of the database.

IHXXPPVDatabase::GetRedirect
   Gets the redirection URL corresponding to the requested URL from the database.
   STDMETHOD(GetRedirect) ( 
      THIS_ 
      const char* pURL, 
      char* pURLRedirect, 
      UINT32 ulURLRedirectLen 
   ) PURE;

   pURL
      Pointer to the URL key for the record to redirect.
pURLRedirect
    Pointer to the URL to which to redirect.

ulURLRedirectLen
    The maximum length of the URL in pURLRedirect.

**IHXPPVDatabase::GetUseridFromGUID**

Gets a user's ID. For this method to return a valid user ID, GUID logging must be enabled (that is, the GUID cannot be all 0's (zeros)).

```cpp
STDMETHOD(GetUseridFromGUID) ( 
    THIS_ 
    const char* pGUID, 
    char* pUserid, 
    UINT32 ulUseridLen 
) PURE;
```

pGUID
    Pointer to the GUID of an existing user. This GUID may be all 0's (zeros) if GUID logging has been disabled by the client or the server.

pUserid
    Pointer to the user ID key to retrieve.

ulUseridLen
    The maximum length of the user ID key in pUserid.

**IHXPPVDatabase::GrantPermission**

Adds a permission record.

```cpp
STDMETHOD(GrantPermission) ( 
    THIS_ 
    const char* pUserid, 
    const PPVPermission* pPPVPermission 
) PURE;
```

pUserid
    Pointer to the user ID key of the record.

pPPVPermission
    Pointer to a PPVPermission structure that contains the permission information and URL.

**IHXPPVDatabase::GrantTime**

Grants a user time to view. This method is obsolete and should not be used by any new applications.

```cpp
STDMETHOD(GrantTime) ( 
    THIS_ 
    const char* pUserid, 
    const PPVPermission* pPPVPermission, 
    UINT32 ulGrant 
) PURE;
```
pUserID
Point to the user ID key of the record.

pPPVPermission
Pointer to a PPVPermission structure that contains the permission information and URL.

ulGrant
The amount of time to grant.

IHXPVDatabase::InitPPVDB
Opens and initializes the pay-per-view database. This method is called once per instance before any other methods in this interface are called.

STDMETHOD(InitPPVDB) ( THIS_ const char* pDBName, const char* pUserID, const char* pPassword ) PURE;

pDBName
Pointer to the name of the database (if supported).

pUserID
Pointer to the user ID used to access the database (if supported).

cPassword
Pointer to the password used to access the database (if supported).

IHXPVDatabase::InsertUser
Inserts a user in the database (the user should not already exist).

STDMETHOD(InsertUser) ( THIS_ const char* pUserid, const char* pPasswordCipher ) PURE;

pUserid
Pointer to the user ID key of the record to insert.

cPasswordCipher
Pointer to an optional password to associate with this user. The password being passed in is already encrypted.

IHXPVDatabase::LogAccess
Inserts a new record into the user authentication database.

STDMETHOD(LogAccess) ( THIS_ const char* pPPVAccessLog ) PURE;
pPPVAccessLog
  Pointer to a PPVAccessLog structure that contains the logging information.

  **For More Information:** For information on user authentication, see the chapters on authenticating Helix Universal Server visitors and storing authentication data in the *Helix Universal Server Administration Guide*.

**IHXPPVDatabase::LogReg**

Inserts a new record into the database. This method uses information supplied by the IHXPPVDatabase::RegisterGUID method.

```cpp
STDMETHOD(LogReg) ( 
  THIS_
  PPVRegLog* pPPVRegLog
) PURE;
```

pPPVRegLog
  Pointer to a PPVRegLog structure that contains the logging information.

  **For More Information:** For information on user authentication, see the chapters on authenticating Helix Universal Server visitors and storing authentication data in the *Helix Universal Server Administration Guide*.

**IHXPPVDatabase::PutRedirect**

Adds a redirection URL to the database that corresponds with the URL key.

```cpp
STDMETHOD(PutRedirect) ( 
  THIS_
  const char* pURL,
  const char* pURLRedirect
) PURE;
```

pURL
  Pointer to the URL key.

pURLRedirect
  Pointer to the redirection URL to be added to the database.

**IHXPPVDatabase::RegisterGUID**

Registers a GUID to be associated with the given user ID. For the GUID to be properly associated with the user ID, GUID logging must be enabled (that is, the GUID cannot be all 0's (zeros)).

```cpp
STDMETHOD(RegisterGUID) ( 
  THIS_
  const char* pUserid,
  const char* pGUID,
  BOOL bForce
) PURE;
```

pUserid
  Pointer to the user ID key of the record.
pGUID
Pointer to the GUID to associate with the user record.

bForce
If TRUE, this forces the registration of the GUID, even if the uuid_writeable flag in the server's authentication database is set to read only. For more information on uuid_writeable, see the chapter on storing authentication data in the Helix Universal Server Administration Guide.

IHXPPVDatabase::RemoveUser
Removes a user from the database.

STDMETHOD(RemoveUser) ( THIS_ const char* pUserid ) PURE;

pUserid
Pointer to the user ID key of the record.

IHXPPVDatabase::RevokeAllPermissions
Removes all permission records for a user.

STDMETHOD(RevokeAllPermissions) ( THIS_ const char* pUserid ) PURE;

pUserid
Pointer to the user ID key of the record.

IHXPPVDatabase::RevokePermission
Removes a permission record.

STDMETHOD(RevokePermission) ( THIS_ const char* pUserid, const PPVPermission* pPPVPermission ) PURE;

pUserid
Pointer to the user ID key of the record.

pPPVPermission
Pointer to a PPVPermission structure that contains the permission information and URL.

IHXPPVDatabase::SetPassword
Sets the password for the user associated with the given user ID.
STDMETHOD(SetPassword) (  
    THIS_,  
    const char* pUserid,  
    const char* pCipherPassword  
) PURE;

pUserid  
    Pointer to the user ID key of the record.

pCipherPassword  
    Pointer to the user’s password.

IHXPVPDatabase::ValidateUser

Validates whether the user associated with the given user ID should be given permission to view a specific piece of content. If the user is granted permission to view the file, the database should fill in the other members of the PPVPPermission structure with information about what kind of permission the user should be granted (that is, should the user be granted a number of seconds of viewing time, or should they be able to watch until a specific expiration date, and so on).

STDMETHOD(ValidateUser) (  
    THIS_,  
    const char* pUserid,  
    PPVPPermission* pPPVPPermission  
) PURE;

pUserid  
    Pointer to the user ID key of the record.

pPPVPPermission  
    Pointer to a PPVPPermission structure that contains the URL of the content that requires validation.
**IHXPPVDBManager**

**Purpose:** Provides storage of pay-per-view permission data.

**Implemented by:** Database plug-in

**Used by:** File system and pay-per-view allowance plug-ins

**Header file:** hxdb.h

This interface administers and enforces permissions for pay-per-view content. The response interface is IHXPPVDBManagerResponse.

The IHXPPVDBManager interface contains the following methods:

- `IHXPPVDBManager::GetPermissions`
- `IHXPPVDBManager::LogAccessAttempt`
- `IHXPPVDBManager::RevokeAllPermissions`
- `IHXPPVDBManager::RevokePermissions`
- `IHXPPVDBManager::SetPermissions`

As with all COM interfaces, the IHXPPVDBManager interface inherits the following IUUnknown methods:

- `IUUnknown::AddRef`
- `IUUnknown::QueryInterface`
- `IUUnknown::Release`

**IHXPPVDBManager::GetPermissions**

Retrieves the pay-per-view permissions for the specified URL and user.

```cpp
HRESULT GetPermissions(IHXPPVDBManagerResponse* pPPVDBManagerResponseNew, IHXBuffer* pBufferPrincipalID, IHXBuffer* pBufferURL) PURE;
```

- `pPPVDBManagerResponseNew` Pointer to an IHXPPVDBManagerResponse interface that manages the response to this method.

- `pBufferPrincipalID` Pointer to an IHXBuffer interface that manages the specified user’s identification.

- `pBufferURL` Pointer to an IHXBuffer interface that manages the specified URL.

**IHXPPVDBManager::LogAccessAttempt**

Records the results of an attempt to access protected content.

```cpp
HRESULT LogAccessAttempt(IHXValues* pValuesAccess) PURE;
```

- `pValuesAccess` Pointer to an IHXValues interface that manages the values associated with the access attempt.
**pValuesAccess**

Pointer to an IHXValues interface that manages the result of an attempt to access protected content. The following keys are used:

- **PPV_ACCLOG_ACCESSGRANTED**
  Indicates whether access was granted or not.
- **PPV_ACCLOG_USERID**
  Information about the connection.
- **PPV_ACCLOG_GUID**
  Information about the connection.
- **PPV_ACCLOG_IPADDRESS**
  Information about the connection.
- **PPV_ACCLOG_URL**
  Information about the connection.
- **PPV_ACCLOG_PERMTYPE**
  The type of permission used.
- **PPV_ACCLOG_PERMON**
  Indicates whether permission was granted for a file or a directory.
- **PPV_ACCLOG_STARTTIME**
  The start time for which permission was granted.
- **PPV_ACCLOG_STOPTIME**
  The stop time for which permission was granted.
- **PPV_ACCLOG_TOTALTIME**
  The total time for which permission was granted
- **PPV_ACCLOG_WHYDISCONNECT**
  The reason for disconnecting.

**IHXPPVDBManager::RevokeAllPermissions**

Removes the pay-per-view permissions for all URLs to which the specified user has access.

```cpp
STDMETHOD(RevokeAllPermissions) (    THIS_    IHXPPVDBManagerResponse* pPPVDBManagerResponseNew,    IHXBuffer* pBufferPrincipalID
) PURE;
```

**pPPVDBManagerResponseNew**

Pointer to an IHXPPVDBManagerResponse interface that manages the response to this method.

**pBufferPrincipalID**

Pointer to an IHXBuffer interface that manages the specified user’s identification.
**IHXPPVDBManager::RevokePermissions**

Removes the pay-per-view permissions for the specified URL and user.

STD\textbf{METHOD}(RevokePermissions) ( 
  \textbf{THIS} 
  \hspace{2em} \textbf{IHXPPVDBManagerResponse* pPPVDBManagerResponseNew,} 
  \hspace{2em} \textbf{IHXBuffer* pBufferPrincipalID,} 
  \hspace{2em} \textbf{IHXBuffer* pBufferURL} 
) PURE;

\textbf{pPPVDBManagerResponseNew}

Pointer to an \textbf{IHXPPVDBManagerResponse} interface that manages the response to this method.

\textbf{pBufferPrincipalID}

Pointer to an \textbf{IHXBuffer} interface that manages the specified user’s identification.

\textbf{pBufferURL}

Pointer to an \textbf{IHXBuffer} interface that manages the specified URL.

**IHXPPVDBManager::SetPermissions**

Sets the pay-per-view permissions for the specified URL and user.

STD\textbf{METHOD}(SetPermissions) ( 
  \textbf{THIS} 
  \hspace{2em} \textbf{IHXPPVDBManagerResponse* pPPVDBManagerResponseNew,} 
  \hspace{2em} \textbf{IHXBuffer* pBufferPrincipalID,} 
  \hspace{2em} \textbf{IHXBuffer* pBufferURL,} 
  \hspace{2em} \textbf{IHXValues* pValuesPermissions} 
) PURE;

\textbf{pPPVDBManagerResponseNew}

Pointer to an \textbf{IHXPPVDBManagerResponse} interface that manages the response to this method.

\textbf{pBufferPrincipalID}

Pointer to an \textbf{IHXBuffer} interface that manages the specified user’s identification.

\textbf{pBufferURL}

Pointer to an \textbf{IHXBuffer} interface that manages the specified URL.

\textbf{pValuesPermissions}

Pointer to an \textbf{IHXValues} interface that manages the pay-per-view permissions. The following keys are used:

- **PPV\_PERM\_URL**
  The exact file or directory to which you are giving information.

- **PPV\_PERM\_URLTYPE**
  Indicates whether the URL is a file or a directory.

- **PPV\_PERM\_PERMTYPE**
  Indicates the permission type being used (if unlimited, the PPV\_PERM\_EXPIRES and PPV\_PERM\_DEBITTIME keys do not apply).

- **PPV\_PERM\_EXPIRES**
If you are using the expiration permission type, this key indicates the expiration date.

• PPV_PERM_DEBITTIME

If you are using the countdown permission type, this key indicates how many seconds are granted.
IHXPPVDBManagerResponse

Purpose: Manages pay-per-view permission data.
Implemented by: Database user
Used by: Database plug-in
Header file: hxdb.h

This interface provides an asynchronous response to IHXPPVDBManager.

The IHXPPVDBManagerResponse interface contains the following methods:
- IHXPPVDBManagerResponse::GetPermissionsDone
- IHXPPVDBManagerResponse::RevokeAllPermissionsDone
- IHXPPVDBManagerResponse::RevokePermissionsDone
- IHXPPVDBManagerResponse::SetPermissionsDone

As with all COM interfaces, the IHXPPVDBManagerResponse interface inherits the following IUnknown methods:
- IUnknown::AddRef
- IUnknown::QueryInterface
- IUnknown::Release

IHXPPVDBManagerResponse::GetPermissionsDone

Reports the status of the call to get the pay-per-view permissions, and returns the permissions for the specified user.

STDMETHOD(GetPermissionsDone) (THIS_ HX_RESULT ResultStatus, IHXBuffer* pBufferPrincipalID, IHXValues* pValuesPermissions ) PURE;

ResultStatus
The status of the IHXPPVDBManager::GetPermissions operation. A value of HXR_OK indicates that the operation was completed successfully.

pBufferPrincipalID
Pointer to an IHXBuffer interface that manages the user's identification.

pValuesPermissions
Pointer to an IHXValues interface that manages the pay-per-view permissions. The following keys are used:
- PPV_PERM_URL
- PPV_PERM_URLTYPE
- PPV_PERM_PERMTYPE
- PPV_PERM_EXPIRES
• PPV_PERM_DEBITTIME

**IHXPPVDBManagerResponse::RevokeAllPermissionsDone**

Reports the status of the call to revoke all the pay-per-view permissions for the specified user.

```cpp
STDMETHOD(RevokeAllPermissionsDone) (
    THIS_,
    HRESULT ResultStatus,
    IHXBuffer* pBufferPrincipalID
) PURE;
```

**ResultStatus**
The status of the IHXPPVDBManager::RevokeAllPermissions operation. A value of HXR_OK indicates that the operation was completed successfully.

**pBufferPrincipalID**
Pointer to an IHXBuffer interface that manages the user’s identification.

**IHXPPVDBManagerResponse::RevokePermissionsDone**

Reports the status of the call to revoke the pay-per-view permissions for the specified user.

```cpp
STDMETHOD(RevokePermissionsDone) (
    THIS_,
    HRESULT ResultStatus,
    IHXBuffer* pBufferPrincipalID
) PURE;
```

**ResultStatus**
The status of the IHXPPVDBManager::RevokePermissions operation. A value of HXR_OK indicates that the operation was completed successfully.

**pBufferPrincipalID**
Pointer to an IHXBuffer interface that manages the user’s identification.

**IHXPPVDBManagerResponse::SetPermissionsDone**

Reports the status of the call to set the pay-per-view permissions for the specified user.

```cpp
STDMETHOD(SetPermissionsDone) (
    THIS_,
    HRESULT ResultStatus,
    IHXBuffer* pBufferPrincipalID
) PURE;
```

**ResultStatus**
The status of the IHXPPVDBManager::SetPermissions operation. A value of HXR_OK indicates that the operation was completed successfully.

**pBufferPrincipalID**
Pointer to an IHXBuffer interface that manages the user’s identification.
IHXPreferenceEnumerator

Purpose: Enumerates the subkeys of a specified preference key.
Implemented by: Helix Architecture
Used by: Top-level client, plug-ins, client core
Header file: hxprefs.h

This interface provides a means of locating a specific preference subkey in the client or server preferences. You first initialize a pointer to the preference key you want to search in the registry using IHXPreferenceEnumerator::BeginSubPref. Then use the IHXPreferenceEnumerator::GetPrefKey method to get the specific preference subkey. After you have located the subkey, you can use the IHXPreferenceEnumerator::ReadPref method to obtain the value for that preference subkey.

The IHXPreferenceEnumerator interface contains the following methods:

- IHXPreferenceEnumerator::BeginSubPref
- IHXPreferenceEnumerator::EndSubPref
- IHXPreferenceEnumerator::GetPrefKey
- IHXPreferenceEnumerator::ReadPref

As with all COM interfaces, the IHXPreferenceEnumerator interface inherits the following IUnknown methods:

- IUnknown::AddRef
- IUnknown::QueryInterface
- IUnknown::Release

IHXPreferenceEnumerator::BeginSubPref

Initializes the preference key enumerator.

```cpp
STDMETHOD(BeginSubPref) ( 
    THIS_ 
    const char* szSubPref
) PURE;
```

szSubPref
Pointer to the preference key whose subkeys are to be enumerated.

IHXPreferenceEnumerator::EndSubPref

Ends the enumeration process.

```cpp
STDMETHOD(EndSubPref) ( 
    THIS
) PURE;
```

IHXPreferenceEnumerator::GetPrefKey

Gets the specified preference subkey for the enumeration.
STDMETHOD(GetPrefKey) (  
   THIS_,  
   UINT32 nIndex,  
   REF(IHXBuffer*) pBuffer  
) PURE;

nIndex
   Index number of the preference subkey.

pBuffer
   Returns a pointer to an IHXBuffer interface that manages the preference subkey name.

IHXPreferenceEnumerator::ReadPref

   Reads the value of the preference subkey returned by IHXPreferenceEnumerator::GetPrefKey.

STDMETHOD(ReadPref) (  
   THIS_,  
   const char* pPrefKey,  
   IHXBuffer*& pBuffer  
) PURE;

pPrefKey
   Pointer to the preference subkey.

pBuffer
   Returns a pointer to the IHXBuffer interface that manages the preference subkey value.
IHXPrefs

Purpose: Stores preferences in the Helix server or client registry or configuration file.

Implemented by: Helix architecture

Used by: Top-level clients, plug-ins, client core

Header file: hxprefs.h

Calls to this interface’s methods enable you to store a preference name/value pair in the server or client registry.

The IHXPrefs interface contains the following methods:

- IHXPrefs::ReadPref
- IHXPrefs::WritePref

As with all COM interfaces, the IHXPrefs interface inherits the following IUnknown methods:

- IUnknown::AddRef
- IUnknown::QueryInterface
- IUnknown::Release

IHXPrefs::ReadPref

Reads a preference from the registry or configuration file.

STDMETHOD(ReadPref) (const char* pPrekKey, IHXBuffer* pBuffer) PURE;

pPrekKey
Pointer to the name of the preference.

pBuffer
Returns a pointer to an IHXBuffer interface that manages the value of the preference.

IHXPrefs::WritePref

Writes a preference to the registry or configuration file.

STDMETHOD(WritePref) (const char* pPrekKey, IHXBuffer* pBuffer) PURE;

pPrekKey
Pointer to the name of the preference.

pBuffer
Returns a pointer to an IHXBuffer interface that manages the value of the preference.
IHXPreferences2

Purpose: Gives subpreference options abilities.
Implemented by: Helix architecture
Used by: Top-level clients
Header file: hxprefs.h

This interface is provided as an addition to the IHXPreferences interface. The methods provided by this interface enable you to enumerate various preference subkeys and to change the default root of the preferences in the server or client registry.

The IHXPreferences2 interface contains the following methods:
- IHXPreferences2::GetPreferenceEnumerator
- IHXPreferences2::ResetRoot

As with all COM interfaces, the IHXPreferences2 interface inherits the following IUnknown methods:
- IUnknown::AddRef
- IUnknown::QueryInterface
- IUnknown::Release

IHXPreferences2::GetPreferenceEnumerator
Provides the interface for reading a preference from the registry.

STDMETHOD(GetPreferenceEnumerator) (THIS_REF(IHXPreferenceEnumerator*) pEnum) PURE;

pEnum Returns a pointer to an IHXPreferenceEnumerator interface that manages the preference enumeration.

IHXPreferences2::ResetRoot
Resets the default root of the preferences. For example, in Windows you can use this to change the default root location of \HKEY_CLASSES_ROOT\Software\RealNetworks\RealMediaSDK\6.0 to another location of your choice.

STDMETHOD(ResetRoot) (THIS_ const char* pCompanyName, const char* pProductName, int nProdMajorVer, int nProdMinorVer) PURE;

pCompanyName The company name that is to replace the default location.
pProductName
The product name that is to replace the default SDK location.

nProdMajorVer
The product major version that is to replace the default “6” location.

nProdMinorVer
The product minor version that is to replace the default “0” location.
IHXPrefetch

Purpose: Reduces buffering time by prefetching packets.
Implemented by: Client core
Used by: SMIL
Header file: hxgroup.h

The IHXPrefetch interface contains the following methods:
• IHXPrefetch::AddPrefetchSink
• IHXPrefetch::AddPrefetchTrack
• IHXPrefetch::GetPrefetchTrack
• IHXPrefetch::GetPrefetchTrackCount
• IHXPrefetch::RemovePrefetchSink
• IHXPrefetch::RemovePrefetchTrack

As with all Component Object Model (COM) interfaces, the IHXPrefetch interface inherits the following IUnknown methods:
• IUnknown::AddRef
• IUnknown::QueryInterface
• IUnknown::Release

IHXPrefetch::AddPrefetchSink

Adds a prefetch sink that provides a callback so the caller can be notified when the prefetch is done.

STDMETHOD(AddPrefetchSink) (THIS_ IHXPrefetchSink* pSink) PURE;
pSink
Pointer to an IHXPrefetchSink interface that manages the status of the prefetch.

IHXPrefetch::AddPrefetchTrack

Adds a prefetch track by specifying the prefetch type (PrefetchType) and prefetch value (PrefetchValue).

STDMETHOD(AddPrefetchTrack) (THIS_ IRMAValues* pTrack) PURE;
pTrack
Pointer to an IHXValues interface that manages the prefetch track to be added.

IHXPrefetch::GetPrefetchTrack

Get the specified prefetch track.
STDMETHOD(GetPrefetchTrack) ( 
    THIS_,
    UINT16 uTrackIndex,
    REF(IRMAValues*) pTrack
) PURE;

uTrackIndex
    The index number of the prefetch track.

pTrack
    Returns a pointer to an IHXValues interface that manages the specified prefetch track.

IHXPrefetch::GetPrefetchTrackCount
    Returns the number of prefetch tracks that have been added.
    STDMETHOD_(UINT16,GetPrefetchTrackCount) ( 
        THIS
    ) PURE;

IHXPrefetch::RemovePrefetchSink
    Removes the prefetch sink.
    STDMETHOD(RemovePrefetchSink) ( 
        THIS_,
        IRMAPrefetchSink* pSink
    ) PURE;

pSink
    Pointer to the IHXPrefetchSink interface to be removed.

IHXPrefetch::RemovePrefetchTrack
    Removes the specified prefetch track.
    STDMETHOD(RemovePrefetchTrack) ( 
        THIS_,
        UINT16 uTrackIndex
    ) PURE;

uTrackIndex
    The index number of the track to be removed.
IHXPrefetchSink

Purpose: Prefetches a specified track.
Implemented by: Client core
Used by: Renderer plug-in
Header file: hxgroup.h

This interface is called by a specific group through IHXGroup2 to prefetch a track in that group. This interface is provided for SMIL 2.0 support.

The IHXPrefetchSink interface contains the following methods:

• IHXPrefetchSink::PrefetchDone
• IHXPrefetchSink::PrefetchTrackAdded
• IHXPrefetchSink::PrefetchTrackRemoved

As with all COM interfaces, the IHXPrefetchSink interface inherits the following IUnknown methods:

• IUnknown::AddRef
• IUnknown::QueryInterface
• IUnknown::Release

IHXPrefetchSink::PrefetchDone

Indicates the prefetch track is done.

STDMETHOD(PrefetchDone) ( 
  THIS_  
  UINT16 uGroupIndex, 
  UINT16 uPrefetchTrackId 
) PURE;

uGroupIndex
  Index of the group to which the track belongs.

uPrefetchTrackId
  Index of the track that was prefetched.

IHXPrefetchSink::PrefetchTrackAdded

Indicates the prefetch track has been added.

STDMETHOD(PrefetchTrackAdded) ( 
  THIS_  
  UINT16 uGroupIndex, 
  UINT16 uPrefetchTrackId, 
  IHXValues* pTrack 
) PURE;

uGroupIndex
  Index of the group to which the track belongs.
uPrefetchTrackId
Index of the track that was added.

pTrack
Pointer to an IHXValues interface that manages the track that was added.

**IHXPrefetchSink::PrefetchTrackRemoved**

Indicates the prefetch track has been removed.

```cpp
STDMETHOD(PrefetchTrackRemoved) ( 
    THIS_ 
    UINT16 uGroupIndex, 
    UINT16 uPrefetchTrackId, 
    IHXValues* pTrack
) PURE;
```

uGroupIndex
Index of the group to which the track belongs.

uPrefetchTrackId
Index of the track that was removed.

pTrack
Pointer to an IHXValues interface that manages the track that was removed.
IHXProcess

Purpose: Creates a new server process.
Implemented by: Server core
Used by: Server plug-ins
Header file: hxengin.h

This interface provides the ability to name the newly-created process and specify an entry point. This interface is queried off the context.

For More Information: See IHXScheduler in Volume 2, on page 373.

The IHXProcess interface contains the IHXProcess::Start method.
As with all COM interfaces, the IHXProcess interface inherits the following IUnknown methods:

• IUnknown::AddRef
• IUnknown::QueryInterface
• IUnknown::Release

IHXProcess::Start

Begins the new server process.

STDMETHOD(Start) ( 
    THIS_ 
    const char* pProcessName,  
    IHXProcessEntryPoint* pEntryPoint  
) PURE;

pProcessName
Pointer to the text string name to apply to the new server process. This parameter can be NULL.

pEntryPoint
Pointer to an IHXProcessEntryPoint interface that manages the entry point for the new server process.
IHXProcessEntryPoint

Purpose: Provides an entry point for a process.
Implemented by: Server plug-ins
Used by: Server core
Header file: hxengin.h

This interface is the callback for IHXProcess.
The IHXProcessEntryPoint interface contains the IHXProcessEntryPoint::Func method.
As with all Component Object Model (COM) interfaces, the IHXProcessEntryPoint interface inherits the following IUnknown methods:
• IUnknown::AddRef
• IUnknown::QueryInterface
• IUnknown::Release

IHXProcessEntryPoint::Func

Provides the callback for IHXProcess::Start. This method should never return—it should run continuously after being called.

STDMETHOD(Func) (THIS_ IUnknown* pContext) PURE;

pContext
Pointer to an IUnknown interface that manages the context for the process.
**IHXPropWatch**

Purpose: Sets watches on registry properties.

Implemented by: Registry object (Helix Architecture)

Used by: Monitor plug-ins

Header file: hxmon.h

After creating a registry object with IHXRegistry, you can use a monitor plug-in to set watches on server registry properties with this interface. Helix Universal Server then uses IHXPropWatchResponse to inform the plug-in of changes to the registry properties.


The IHXPropWatch interface contains the following methods:

- IHXPropWatch::ClearWatchById
- IHXPropWatch::ClearWatchByName
- IHXPropWatch::ClearWatchOnRoot
- IHXPropWatch::Init
- IHXPropWatch::SetWatchById
- IHXPropWatch::SetWatchByName
- IHXPropWatch::SetWatchOnRoot

As with all COM interfaces, the IHXPropWatch interface inherits the following IUnknown methods:

- IUnknown::AddRef
- IUnknown::QueryInterface
- IUnknown::Release

**IHXPropWatch::ClearWatchById**

Clears a watch-point based on the specified property's ID.

```cpp
STDMETHOD(ClearWatchById) (const UINT32 id) PURE;
```

id

The ID of the property from which to clear the watch-point.

**IHXPropWatch::ClearWatchByName**

Clears a watch-point based on the specified property's name.

```cpp
STDMETHOD(ClearWatchByName) (const char* pName) PURE;
```
pName
   Pointer to the name of the property from which to clear the watch-point.

**IHXPropWatch::ClearWatchOnRoot**

Clears the watch on the root of the registry.

```cpp
STDMETHOD(ClearWatchOnRoot) (THIS ) PURE;
```

**IHXPropWatch::Init**

Initializes the response interface so that the watch notifications can be sent back to their respective plug-ins.

```cpp
STDMETHOD(Init) (THIS _ IHXPropWatchResponse* pResponse ) PURE;
```

pResponse
   Pointer to an IHXPropWatchResponse interface that manages responses to various IHXPropWatch methods.

**IHXPropWatch::SetWatchById**

Sets a watch-point on the property whose ID is specified. A notification is returned to the plug-in that set the watch-point if that particular property gets modified or deleted.

```cpp
STDMETHOD_(UINT32, SetWatchById) (THIS _ const UINT32 id ) PURE;
```

id
   The ID of the property for which a watch-point is set.

**IHXPropWatch::SetWatchByName**

Sets a watch-point on the property whose name is specified. A notification is returned to the plug-in that set the watch-point if that particular property gets modified or deleted.

```cpp
STDMETHOD_(UINT32, SetWatchByName) (THIS _ const char* pName ) PURE;
```

pName
   Pointer to the name of the property for which a watch-point is set.
**IHXPropWatch::SetWatchOnRoot**

Puts a watch at the highest level of the registry hierarchy. This method returns a notification to the plug-in only if properties at this level get added, modified, or deleted.

```cpp
STDMETHOD_(UINT32, SetWatchOnRoot) (THIS
    
) PURE;
```
IHXPropWatchResponse

Purpose: Informs the monitor plug-in of server registry changes, and the top-level client of statistics changes.

Implemented by: Monitor plug-in, top-level client

Used by: Server registry object (Helix architecture)

Header file: hxmon.h

A monitor plug-in or a top-level client implements this response interface to IHXPropWatch to receive information about changes to monitored properties in the registry.

For More Information: See “Chapter 9: Managing the Server Registry”.

The IHXPropWatchResponse interface contains the following methods:

• IHXPropWatchResponse::AddedProp
• IHXPropWatchResponse::DeletedProp
• IHXPropWatchResponse::ModifiedProp

As with all COM interfaces, the IHXPropWatchResponse interface inherits the following IUnknown methods:

• IUnknown::AddRef
• IUnknown::QueryInterface
• IUnknown::Release

IHXPropWatchResponse::AddedProp

Responds when a new property gets added under the property on which the watch was set.

STDMETHOD(AddedProp) ( 
    THIS_ 
    const UINT32 id, 
    const HXPropType propType, 
    const UINT32 ulParentID 
) PURE;

id
    The ID of the property that was just added to the registry.

propType
    The data type of the property that was just added to the registry.

ulParentID
    The ID of the added property’s immediate parent COMPOSITE property.

IHXPropWatchResponse::DeletedProp

Responds when a watched property gets deleted. This method gets passed in the ID of the property just deleted and its immediate parent COMPOSITE property.
STDMETHOD(DeletedProp) (  
  THIS_  
  const UINT32 id,  
  const UINT32 ulParentID  
) PURE;

\textit{id}  
The ID of the property that was just deleted from the registry.

\textit{ulParentID}  
The ID of the deleted property's immediate parent COMPOSITE property.

\textbf{IHXPropWatchResponse::ModifiedProp}

Responds when a watched property gets modified.

STDMETHOD(ModifiedProp) (  
  THIS_  
  const UINT32 id,  
  const HXPropType propType,  
  const UINT32 ulParentID  
) PURE;

\textit{id}  
The ID of the property that was just modified in the registry.

\textit{propType}  
The data type of the property that was just modified in the registry.

\textit{ulParentID}  
The ID of the modified property's immediate parent COMPOSITE property.
**IHXRawSinkObject**

**Purpose:** Receives raw packets from a source.

**Implemented by:** Server's live packet sink plug-ins

**Used by:** Live source

**Header file:** hxsrc.h

This interface provides notifications about all packets that are served from a live source connected to the Helix Architecture (HX) server. It also retrieves the relevant file and stream headers.

The `IHXRawSinkObject` interface contains the following methods:

- `IHXRawSinkObject::FileHeaderReady`
- `IHXRawSinkObject::InitDone`
- `IHXRawSinkObject::PacketReady`
- `IHXRawSinkObject::StreamDone`
- `IHXRawSinkObject::StreamHeaderReady`

As with all COM interfaces, the `IHXRawSinkObject` interface inherits the following `IUnknown` methods:

- `IUnknown::AddRef`
- `IUnknown::QueryInterface`
- `IUnknown::Release`

### `IHXRawSinkObject::FileHeaderReady`

Indicates the status of the request for the file header.

```c
STDMETHOD(FileHeaderReady) (  
    THIS_  
    HX_RESULT status,  
    IHXValues* pHeader  
) PURE;
```

**status**

The status of the `IHXRawSourceObject::GetFileHeader` operation. A value of `HXR_OK` indicates that the operation was completed successfully.

**pHeader**

Pointer to an `IHXValues` interface that manages the file header information.

### `IHXRawSinkObject::InitDone`

Indicates the status after the source interface initializes the connection.

```c
STDMETHOD(InitDone) (  
    THIS_  
    HX_RESULT status  
) PURE;
```
status
The status of the IHXRawSourceObject::Init operation. A value of HXR_OK indicates that the operation was completed successfully.

**IHXRawSinkObject::PacketReady**
Indicates the status of the request for packets. This method is called for every packet received from the live source.

```cpp
STDMETHOD(PacketReady) ( 
    THIS_ 
    HX_RESULT status, 
    IHXPacket* pPacket
) PURE;
```

status
The status of the IHXRawSourceObject::StartPackets operation. A value of HXR_OK indicates that the operation was completed successfully.

**pPacket**
Pointer to an IHXPacket interface that manages the packets.

**IHXRawSinkObject::StreamDone**
Indicates that a live stream was completed.

```cpp
STDMETHOD(StreamDone) ( 
    THIS_ 
    UINT16 unStreamNumber
) PURE;
```

unStreamNumber
The stream number of the live stream that has completed.

**IHXRawSinkObject::StreamHeaderReady**
Indicates the status of the request for the stream header.

```cpp
STDMETHOD(StreamHeaderReady) ( 
    THIS_ 
    HX_RESULT status, 
    IHXValues* pHeader
) PURE;
```

**status**
The status of the IHXRawSourceObject::GetStreamHeader operation. A value of HXR_OK indicates that the operation was completed successfully.

**pHeader**
Pointer to an IHXValues interface that manages the stream header information.
**IHXRawSourceObject**

Purpose: Serves packets to sinks.
Implemented by: Server core
Used by: Live packet sink plug-in
Header file: hxsrc.h

This interface converses with the server to get the packets for a live stream.

The IHXRawSourceObject interface contains the following methods:

• IHXRawSourceObject::Done
• IHXRawSourceObject::GetFileHeader
• IHXRawSourceObject::GetStreamHeader
• IHXRawSourceObject::Init
• IHXRawSourceObject::StartPackets
• IHXRawSourceObject::StopPackets

As with all COM interfaces, the IHXRawSourceObject interface inherits the following IUnknown methods:

• IUnknown::AddRef
• IUnknown::QueryInterface
• IUnknown::Release

**IHXRawSourceObject::Done**

Finished receiving packets.

STDMETHOD(Done) (THIS) PURE;

**IHXRawSourceObject::GetFileHeader**

Retrieves the file header.

STDMETHOD(GetFileHeader) (THIS) PURE;

**IHXRawSourceObject::GetStreamHeader**

Retrieves the indicated stream header.

STDMETHOD(GetStreamHeader) (THIS,

  UINT16 unStreamNumber
) PURE;

unStreamNumber
  The stream number of the stream from which the header is to be retrieved.
**IHXRawSourceObject::Init**

Initializes the connection between the source and the sink interfaces.

```cpp
STDMETHOD(Init) ( 
    THIS_ 
    IUnknown* pUnknown 
) PURE;
```

*pUnknown*

Pointer to an instance of a IHXRawSinkObject interface.

**IHXRawSourceObject::StartPackets**

Starts sending packets to the indicated stream.

```cpp
STDMETHOD(StartPackets) ( 
    THIS_ 
    UINT16 unStreamNumber 
) PURE;
```

*unStreamNumber*

The stream number to which to send the packets.

**IHXRawSourceObject::StopPackets**

Stops sending packets to the indicated stream.

```cpp
STDMETHOD(StopPackets) ( 
    THIS_ 
    UINT16 unStreamNumber 
) PURE;
```

*unStreamNumber*

The stream number to which to stop sending the packets.
IHXReconfigServerResponse

Purpose: Indicates that the server reconfiguration was completed
Implemented by: Server plug-in
Used by: Server core
Header file: hxcomm.h

This is the response interface for the IHXServerControl2::ReconfigServer and
IHXWantServerReconfigNotification::ServerReconfig methods.
The IHXReconfigServerResponse interface contains the IHXReconfigServerResponse::ReconfigServerDone
method.
As with all COM interfaces, the IHXReconfigServerResponse interface inherits the following IUnknown
methods:
• IUnknown::AddRef
• IUnknown::QueryInterface
• IUnknown::Release

IHXReconfigServerResponse::ReconfigServerDone

Indicates that the server has been reconfigured.

STDMETHOD(ReconfigServerDone) (
    THIS_ 
    HX_RESULT res,
    IHXBuffer** pInfo,
    UINT32 ulNumInfo
) PURE;

res
   The status of the IHXServerControl2::ReconfigServer operation. A value of HXR_OK indicates that the
   operation was completed successfully.

pInfo
   Pointer to a pointer to an IHXBuffer interface that manages some of the active prop users affected by
   reparsing the configuration file.

ulNumInfo
   The number of active prop users in pInfo.
**IHXRecordTimeline**

This interface is obsolete and should not be used in any new programming.

**IHXRecordTimeline::AdjustTimeline**

This method is obsolete and should not be used in any new programming.

**IHXRecordTimeline::GetRecordPos**

This method is obsolete and should not be used in any new programming.

**IHXRecordTimeline::IsRecordStartPosSet**

This method is obsolete and should not be used in any new programming.

**IHXRecordTimeline::OnPlaybackPos**

This method is obsolete and should not be used in any new programming.

**IHXRecordTimeline::SetRecordStartPos**

This method is obsolete and should not be used in any new programming.
**IHXRedirectDBManager**

Purpose: Provides storage of URLs to be redirected.

Implemented by: Database plug-in, MSOL and ODBC plug-ins

Used by: Pay-per-view allowance plug-in

Header file: hxdb.h

This interface manages the URL redirection of database information. The response interface is IHXRedirectDBManagerResponse.

The IHXRedirectDBManager interface contains the following methods:

- IHXRedirectDBManager::AddRedirect
- IHXRedirectDBManager::GetRedirect
- IHXRedirectDBManager::RemoveRedirect

As with all COM interfaces, the IHXRedirectDBManager interface inherits the following IUnknown methods:

- IUnknown::AddRef
- IUnknown::QueryInterface
- IUnknown::Release

### IHXRedirectDBManager::AddRedirect

Sets the new URL to which the specified URL should be redirected.

```cpp
STDMETHOD(AddRedirect) (  
    THIS_,  
    IHXRedirectDBManagerResponse* pRedirectDBManagerResponseNew,  
    IHXBuffer* pBufferURL,  
    IHXBuffer* pBufferNewURL  
) PURE;
```

- **pRedirectDBManagerResponseNew**
  Pointer to an IHXRedirectDBManagerResponse interface that manages the response to this method.

- **pBufferURL**
  Pointer to an IHXBuffer interface that manages the specified URL.

- **pBufferNewURL**
  Pointer to an IHXBuffer interface that manages the new URL.

### IHXRedirectDBManager::GetRedirect

Retrieves the URL to which the specified URL should be redirected.

```cpp
STDMETHOD(GetRedirect) (  
    THIS_,  
    IHXRedirectDBManagerResponse* pRedirectDBManagerResponseNew,  
    IHXBuffer* pBufferURL  
) PURE;
```
pRedirectDBManagerResponseNew
    Pointer to an IHXRedirectDBManagerResponse interface that manages the response to this method.

pBufferURL
    Pointer to an IHXBuffer interface that manages the specified URL.

IHXRedirectDBManager::RemoveRedirect

    Stops redirecting the specified URL.

    STDMETHOD(RemoveRedirect) ( 
        THIS_,
        IHXRedirectDBManagerResponse* pRedirectDBManagerResponseNew,
        IHXBuffer* pBufferURL
    ) PURE;

pRedirectDBManagerResponseNew
    Pointer to an IHXRedirectDBManagerResponse interface that manages the response to this method.

pBufferURL
    Pointer to an IHXBuffer interface that manages the specified URL.
IHXRedirectDBManagerResponse

Purpose: Manages the URLs to be redirected.

Implemented by: Pay-per-view allowance plug-in

Used by: Database plug-in, Admin file system

Header file: hxdb.h

This interface provides asynchronous responses for the IHXRedirectDBManager interface.

The IHXRedirectDBManagerResponse interface contains the following methods:

- IHXRedirectDBManagerResponse::AddRedirectDone
- IHXRedirectDBManagerResponse::GetRedirectDone
- IHXRedirectDBManagerResponse::RemoveRedirectDone

As with all COM interfaces, the IHXRedirectDBManagerResponse interface inherits the following IUnknown methods:

- IUnknown::AddRef
- IUnknown::QueryInterface
- IUnknown::Release

IHXRedirectDBManagerResponse::AddRedirectDone

Reports the status of the call to set the new URL.

STDMETHOD(AddRedirectDone) (  
    THIS_  
    HX_RESULT ResultStatus,  
    IHXBuffer* pBufferURL  
) PURE;

ResultStatus
The status of the IHXRedirectManager::AddRedirect operation. A value of HXR_OK indicates that the operation was completed successfully.

pBufferURL
Pointer to an IHXBuffer interface that manages the specified URL (the original URL, not the redirected URL).

IHXRedirectDBManagerResponse::GetRedirectDone

Reports the status of the call to get the URL to which to redirect the specified URL.

STDMETHOD(GetRedirectDone) (  
    THIS_  
    HX_RESULT ResultStatus,  
    IHXBuffer* pBufferURL,  
    IHXBuffer* pBufferNewURL  
) PURE;
ResultStatus
The status of the IHXRedirectDBManager::GetRedirect operation. A value of HXR_OK indicates that the operation was completed successfully.

pBufferURL
Pointer to an IHXBuffer interface that manages the specified URL.

pBufferNewURL
Pointer to an IHXBuffer interface that manages the new URL.

IHXRedirectDBManagerResponse::RemoveRedirectDone
Reports the status of the call to remove the new URL.

STDMETHOD(RemoveRedirectDone) (  
   THIS_  
   HX_RESULT ResultStatus,  
   IHXBuffer* pBufferURL  
) PURE;

ResultStatus
The status of the IHXRedirectDBManager::RemoveRedirect operation. A value of HXR_OK indicates that the operation was completed successfully.

pBufferURL
Pointer to an IHXBuffer interface that manages the specified URL from which to remove the redirected URL.
**IHXRegConfig**

Use this interface to rewrite the configuration file after making a configuration change.

The `IHXRegConfig` interface contains the `IHXRegConfig::WriteKey` method.

As with all COM interfaces, the `IHXRegConfig` interface inherits the following `IUnknown` methods:

- `IUnknown::AddRef`
- `IUnknown::QueryInterface`
- `IUnknown::Release`

### `IHXRegConfig::WriteKey`

Writes out the registry from the passed-in key name to the currently active permanent configuration storage area (for example, the configuration file or the registry).

```c
STDMETHOD(WriteKey) (  
    THIS_  
    const char* pKeyName  
) PURE;
```

**pKeyName**

Pointer to the key name in the registry from which to write.
IHXRegistrationLogger

Purpose: Provides storage for player registration attempts.
Implemented by: Database plug-in
Used by: Pay-per-view allowance plug-in
Header file: hxdb.h

This interface logs the registration of a player’s GUID in the GUID-based authentication model. The IHXRegistrationLogger interface contains the IHXRegistrationLogger::LogRegistrationAttempt method. As with all COM interfaces, the IHXRegistrationLogger interface inherits the following IUnknown methods:
- IUnknown::AddRef
- IUnknown::QueryInterface
- IUnknown::Release

IHXRegistrationLogger::LogRegistrationAttempt

Records the results of an attempt to register a player’s GUID.

STDMETHOD(LogRegistrationAttempt) (THIS_ IHXValues* pValuesRegistration) PURE;

pValuesRegistration
Pointer to an IHXValues interface that manages a list of names for which values are expected. The names included in this list are as follows:
- Status
  The status of the operation. This can be any of the following:
  - PPV_GUID_REG_SUCCESS — The GUID was retrieved and successfully entered in the database.
  - PPV_GUID_REG_FAILED_LOCKED — The entry was already registered in the database and the database is locked.
  - PPV_GUID_REG_FAILED_COLLISION — Attempted to register a GUID that matches a GUID in the database with another user name.
  - PPV_GUID_REG_FAILED_OLD_PLAYER — The player was too old to participate in this pay-per-view process (for example, the player had no GUID).
  - PPV_GUID_REG_FAILED_NO_USER — No user by this name could be found.
  - PPV_GUID_REG_FAILED — An unspecified failure occurred.
- Userid
  The user ID associated with the GUID in the registration URL.
- GUID
The actual GUID sent.

• IPAddress
  The IP address of the client that accessed the content.

• RequestTime
  The start time requested, indicated by the amount of time that has elapsed since January 1, 1970.

• URLRedirect
  The URL to play if registration succeeds.
The Helix Universal Server registry stores information about Helix events, such as how many clients are currently connected. The Helix client registry stores statistics about media players. A monitor plug-in can use the IHXRegistry interface to gain access to a registry. This interface creates a registry object that the plug-in can then use to set watches on registry properties through IHXPropWatch.


The IHXRegistry interface contains the following methods:

- IHXRegistry::AddBuf
- IHXRegistry::AddComp
- IHXRegistry::AddInt
- IHXRegistry::AddIntRef
- IHXRegistry::AddStr
- IHXRegistry::CreatePropWatch
- IHXRegistry::DeleteById
- IHXRegistry::DeleteByName
- IHXRegistry::FindParentIdById
- IHXRegistry::FindParentIdByName
- IHXRegistry::GetBufById
- IHXRegistry::GetBufByName
- IHXRegistry::GetId
- IHXRegistry::GetIntById
- IHXRegistry::GetIntByName
- IHXRegistry::GetNumPropsAtRoot
- IHXRegistry::GetNumPropsById
- IHXRegistry::GetNumPropsByName
- IHXRegistry::GetPropListById
- IHXRegistry::GetPropListByName
- IHXRegistry::GetPropListOfRoot
- IHXRegistry::GetPropName
- IHXRegistry::GetStrById
- IHXRegistry::GetStrByName
- IHXRegistry::GetTypeById
- IHXRegistry::GetTypeByName
As with all COM interfaces, the IHXRegistry interface inherits the following IUnknown methods:

- IUnknown::AddRef
- IUnknown::QueryInterface
- IUnknown::Release

### IHXRegistry::AddBuf

 Adds a BUFFER property to the registry. This method returns the ID of the newly added property if successful or returns 0 (zero) if an error occurred during the operation.

```cpp
STDMETHOD_(UINT32, AddBuf) (THIS_ const char* pName, IHXBuffer* pValue) PURE;
```

- **pName**
  - Pointer to the name of the newly added BUFFER property.

- **pValue**
  - Pointer to an IHXBuffer interface that manages the BUFFER value.

### IHXRegistry::AddComp

 Adds a COMPOSITE property to the registry. This method returns the ID of the newly added property if successful or returns 0 (zero) if an error occurred during the operation.

```cpp
STDMETHOD_(UINT32, AddComp) (THIS_ const char* pName) PURE;
```

- **pName**
  - Pointer to the name of the newly added COMPOSITE property.

### IHXRegistry::AddInt

 Adds an INTEGER property to the registry. This method returns the ID of the newly added property if successful or returns 0 (zero) if an error occurred during the operation.

```cpp
STDMETHOD_(UINT32, AddInt) (THIS_ const char* pName, const INT32 iValue) PURE;
```
pName
   Pointer to the name of the newly added INTEGER property.

iValue
   The value of the newly added INTEGER property.

IHXRegistry::AddIntRef

Adds an INTEGER REFERENCE property to the registry. This property enables the user to modify the registry’s contents directly without having to go through the registry itself. This method returns the ID of the newly added property if successful or returns 0 (zero) if an error occurred during the operation.

STDMETHOD_(UINT32, AddIntRef) (  
   THIS_  
   const char* pName,  
   INT32* pValue  
) PURE;

pName
   Pointer to the name of the newly added INTEGER REFERENCE property.

pValue
   Pointer to the value of the newly added INTEGER REFERENCE property.

IHXRegistry::AddStr

Adds a STRING property to the registry. This method returns the ID of the newly added property if successful or returns 0 (zero) if an error occurred during the operation.

STDMETHOD_(UINT32, AddStr) (  
   THIS_  
   const char* pName,  
   IHXBuffer* pValue  
) PURE;

pName
   Pointer to the name of the newly added STRING property.

pValue
   Pointer to an IHXBuffer interface that manages the STRING value.

IHXRegistry::CreatePropWatch

Creates a new IHXPropWatch interface that can then be used to set watches on specific registry properties.

STDMETHOD(CreatePropWatch) (  
   THIS_  
   REF(IHXPropWatch*) pPropWatch  
) PURE;

pPropWatch
   Returns a pointer to an IHXPropWatch interface that manages the watches on registry properties.
IHXRegistry::DeleteById

Deletes a property from the registry using its ID.

```c
STDMETHOD_(UINT32, DeleteById) (THIS_ const UINT32 ulID) PURE;
```

`ulID`  
The ID of the property to delete from the registry.

IHXRegistry::DeleteByName

Deletes a property from the registry using its name.

```c
STDMETHOD_(UINT32, DeleteByName) (THIS_ const char* pcPropName) PURE;
```

`pcPropName`  
Pointer to the name of the property to delete from the registry.

IHXRegistry::FindParentIdById

Returns the ID value of the parent node of the property whose ID has been specified. If this method fails, a 0 (zero) value is returned.

```c
STDMETHOD_(UINT32, FindParentIdById) (THIS_ const UINT32 id) const PURE;
```

`id`  
The ID of the property from which to return the ID value of the parent node.

IHXRegistry::FindParentIdByName

Returns the ID value of the parent node of the property whose name has been specified. If this method fails, a 0 (zero) value is returned.

```c
STDMETHOD_(UINT32, FindParentIdByName) (THIS_ const char* pName) const PURE;
```

`pName`  
Pointer to the name of the property from which to return the ID value of the parent node.

IHXRegistry::GetBufById

Retrieves a BUFFER value from the registry given its property ID. If the property is found, this method returns HXR_OK, otherwise it returns HXR_FAIL.

```c
```
STDMETHOD(GetBufById) (  
    THIS_,  
    const UINT32 id,  
    REF(IHXBuffer*) ppValue  
) const PURE;

id  
The property ID of the BUFFER value to retrieve from the registry.

ppValue  
Returns a pointer to an IHXBuffer interface that manages the BUFFER value.

IHXRegistry::GetBufByName

Retrieves a BUFFER value from the registry given its property name. If the property is found, this  
method returns HXR_OK, otherwise it returns HXR_FAIL.

STDMETHOD(GetBufByName) (  
    THIS_,  
    const char* pName,  
    REF(IHXBuffer*) ppValue  
) const PURE;

pName  
Pointer to the property name of the BUFFER value to retrieve from the registry.

ppValue  
Returns a pointer to an IHXBuffer interface that manages the BUFFER value.

IHXRegistry::GetId

Returns the property's ID given the property name.

STDMETHOD_(UINT32, GetId) (  
    THIS_,  
    const char* pName  
) const PURE;

pName  
Pointer to the property name from which to return the property ID.

IHXRegistry::GetIntById

Retrieves an INTEGER value from the registry given its property ID. If the property is found, this  
method returns HXR_OK, otherwise it returns HXR_FAIL.

STDMETHOD(GetIntById) (  
    THIS_,  
    const UINT32 ulId,  
    REF(INT32) nValue  
) const PURE;

ulID  
The property ID of the INTEGER value to retrieve from the registry.
nValue
Returns the INTEGER value.

IHXRegistry::GetIntByName
Retrieves an INTEGER value from the registry given its property name. If the property is found, this method returns HXR_OK, otherwise it returns HXR_FAIL.

STDMETHOD(GetIntByName) (THIS_ const char* pName, REF(INT32) nValue) const PURE;

pName
Pointer to the property name of the INTEGER value to retrieve from the registry.

nValue
Returns the INTEGER value.

IHXRegistry::GetNumPropsAtRoot
Returns the number of properties at the root of the registry.

STDMETHOD_(INT32, GetNumPropsAtRoot) (THIS) const PURE;

IHXRegistry::GetNumPropsById
Returns the count of the number of properties in the registry. If a property's ID is specified, then it returns the number of properties under that ID.

STDMETHOD_(INT32, GetNumPropsById) (THIS_ const UINT32 id) const PURE;

id
The property ID that specifies a starting point for the count.

IHXRegistry::GetNumPropsByName
Returns the count of the number of properties in the registry. If a property name is specified, then it returns the number of properties under that name.

STDMETHOD_(INT32, GetNumPropsByName) (THIS_ const char* pName) const PURE;

pName
The property name that specifies a starting point for the count.
**IHXRegistry::GetPropListById**

Returns a list of properties immediately under the one with the specified ID.

```cpp
STDMETHOD(GetPropListById) (    THIS_    const UINT32 id,    REF(IHXValues*) pValues ) const PURE;
```

- **id**
  - The ID of the array of properties to return.

- **pValues**
  - Returns a pointer to an IHXValues interface that manages the array of properties.

**IHXRegistry::GetPropListByName**

Returns a list of properties immediately under the one with the specified name.

```cpp
STDMETHOD(GetPropListByName) (    THIS_    const char* pName,    REF(IHXValues*) pValues ) const PURE;
```

- **pName**
  - The name of the array of properties to return.

- **pValues**
  - Returns a pointer to an IHXValues interface that manages the array of properties.

**IHXRegistry::GetPropListOfRoot**

Returns a list of properties under the root level of the registry's hierarchy.

```cpp
STDMETHOD(GetPropListOfRoot) (    THIS_    REF(IHXValues*) pValues ) const PURE;
```

- **pValues**
  - Returns a pointer to an IHXValues interface that manages the array of properties.

**IHXRegistry::GetPropName**

Returns the property name given the property's ID.

```cpp
STDMETHOD(GetPropName) (    THIS_    const UINT32 ulId,    REF(IHXBuffer*) pName ) const PURE;
```

- **ulID**
  - The property ID of the property name to return.
pName
Returns a pointer to an IHXBuffer interface that manages the property name.

IHXRegistry::GetStrById
Retrieves a STRING value from the registry given its property ID. If the property is found, this method
returns HXR_OK, otherwise it returns HXR_FAIL.

STDMETHOD(GetStrById) (const UINT32 id, REF(IHXBuffer*) pValue) const PURE;

id
The property ID of the STRING value to retrieve from the registry.

pValue
Returns a pointer to an IHXBuffer interface that manages the STRING value.

IHXRegistry::GetStrByName
Retrieves a STRING value from the registry given its property name. If the property is found, this
method returns HXR_OK, otherwise it returns HXR_FAIL.

STDMETHOD(GetStrByName) (const char* pName, REF(IHXBuffer*) pValue) const PURE;

pName
Pointer to the property name of the STRING value to retrieve from the registry.

pValue
Returns a pointer to an IHXBuffer interface that manages the STRING value.

IHXRegistry::GetTypeById
Returns the data type of the property given its ID.

STDMETHOD_(HXPropType, GetTypeById) (const UINT32 id) const PURE;

id
The ID of the property from which to return the data type.

IHXRegistry::GetTypeByName
Returns the data type of the property given its name.

Returns the data type of the property given its name.
STDMETHOD_(HXPropType, GetTypeByName) (  
  THIS_  
  const char* pName  
) const PURE;

pName
  Pointer to the name of the property from which to return the data type.

IHXRegistry::SetBufById

Modifies a property’s BUFFER value in the registry given its property ID. If the value was set, this method returns HXR_OK, otherwise it returns HXR_FAIL.

STDMETHOD(SetBufById) (  
  THIS_  
  const UINT32 id,  
  IHXBuffer* pValue  
) PURE;

id
  The property ID of the BUFFER value to set in the registry.

pValue
  Pointer to an IHXBuffer interface that manages the new BUFFER value to be set.

IHXRegistry::SetBufByName

Modifies a property’s BUFFER value in the registry given its property name. If the value was set, this method returns HXR_OK, otherwise it returns HXR_FAIL.

STDMETHOD(SetBufByName) (  
  THIS_  
  const char* pName,  
  IHXBuffer* pValue  
) PURE;

pName
  Pointer to the property name of the BUFFER value to set in the registry.

pValue
  Pointer to an IHXBuffer interface that manages the new BUFFER value to be set.

IHXRegistry::SetIntById

Modifies a property’s INTEGER value in the registry given its property ID. If the value was set, this method returns HXR_OK, otherwise it returns HXR_FAIL.

STDMETHOD(SetIntById) (  
  THIS_  
  const UINT32 id,  
  const INT32 iValue  
) PURE;

id
  The property ID of the INTEGER value to set in the registry.
iValue
The new INTEGER value to be set.

IHXRegistry::SetIntByName
Modifies a property's INTEGER value in the registry given its property name. If the value was set, this
method returns HXR_OK, otherwise it returns HXR_FAIL.

STDMETHOD(SetIntByName) (  
    THIS_  
    const char* pName,  
    const INT32 iValue  
) PURE;

pName
  Pointer to the property name of the INTEGER value to set in the registry.

iValue
  The new INTEGER value to be set.

IHXRegistry::SetStrById
Modifies a property's STRING value in the registry given its property ID. If the value was set, this
method returns HXR_OK, otherwise it returns HXR_FAIL.

STDMETHOD(SetStrById) (  
    THIS_  
    const UINT32 id,  
    IHXBuffer* pValue  
) PURE;

id
  The property ID of the STRING value to set in the registry.

pValue
  Pointer to an IHXBuffer interface that manages the new INTEGER value to be set.

IHXRegistry::SetStrByName
Modifies a property's STRING value in the registry given its property name. If the value was set, this
method returns HXR_OK, otherwise it returns HXR_FAIL.

STDMETHOD(SetStrByName) (  
    THIS_  
    const char* pName,  
    IHXBuffer* pValue  
) PURE;

pName
  Pointer to the property name of the STRING value to set in the registry.

pValue
  Pointer to an IHXBuffer interface that manages the new INTEGER value to be set.
The registry treats strings and buffers as different types, even though at their core they are stored the same (using an IHXBuffer interface). Therefore, if you set a string, and then attempt to retrieve it using either IHXRegistry::GetBufById or IHXRegistry::GetBufBy Name, the call will fail and return HXR_PROP_TYPE_MISMATCH. However, if you query (IUnknown::QueryInterface) IHXRegistry for the IHXRegistryAltStringHandling interface, you can then specify properties where it is all right to treat a string as a buffer. You do this by passing the registry ID of the string property you want to permit into IHXRegistryAltStringHandling::SetStringAccessAsBufferById. You can then retrieve a string using either IHXRegistry::GetBufById or IHXRegistry::GetBufByName in addition to being able to retrieve it using IHXRegistry::GetStrById or IHXRegistry::GetStrByName.

The IHXRegistryAltStringHandling interface inherits the following IUnknown methods:

- IUnknown::AddRef
- IUnknown::QueryInterface
- IUnknown::Release

IHXRegistryAltStringHandling::SetStringAccessAsBufferById

Accesses a buffer as though it were a string. Create the property as a string, then pass this method the strings ID. The property can now be accessed or set as a buffer, but it will still be a string.

STDMETHOD (SetStringAccessAsBufferById) (
    THIS_
    UINT32 ulId
) PURE;

ulId

The ID of the string to be accessed or set as a buffer.
**IHXRegistryID**

Purpose: Gets the registry ID for an object.

Implemented by: Player, stream, and stream source objects (Helix architecture)

Used by: Top-level client

Header file: hxcomm.h

Through this interface, the top-level client can get the registry ID of an IHXPlayer, IHXStream, or IHXStreamSource object. It can then use IHXStatistics to pass this ID to the client core and tell the client to update the object statistics.


The IHXRegistryID interface contains the IHXRegistryID::GetID method.

As with all COM interfaces, the IHXRegistryID interface inherits the following IUnknown methods:

- IUnknown::AddRef
- IUnknown::QueryInterface
- IUnknown::Release

**IHXRegistryID::GetID**

Gets the registry ID of the player, source, or stream object.

```cpp
STDMETHOD(GetID) (
    THIS_
    REF(UINT32) ulRegistryID
) PURE;
```

ulRegistryID

Returns the registry ID.
**IHXRmoteBroadcastConfiguration**

**Purpose:** Provides a simple interface to the XML-based configuration.

**Implemented by:** Remote broadcast services

**Used by:** Remote broadcast services users (such as SLTA)

**Header file:** hxbrcst.h

This interface is called at any point before or during a remote broadcast session. It can be used either to add or remove a host to which remote broadcast services is being transmitted. The response interface is IHXRmoteBroadcastConfigurationResponse.

The IHXRmoteBroadcastConfiguration interface contains the following methods:

- IHXRmoteBroadcastConfiguration::AddDestination
- IHXRmoteBroadcastConfiguration::AddPullSplit
- IHXRmoteBroadcastConfiguration::Commit
- IHXRmoteBroadcastConfiguration::Init
- IHXRmoteBroadcastConfiguration::RemoveDestination
- IHXRmoteBroadcastConfiguration::RemovePullSplit
- IHXRmoteBroadcastConfiguration::RetrieveDestination
- IHXRmoteBroadcastConfiguration::RetrievePullSplit
- IHXRmoteBroadcastConfiguration::UpdateDestination
- IHXRmoteBroadcastConfiguration::UpdatePullSplit

As with all Component Object Model (COM) interfaces, the IHXRmoteBroadcastConfiguration interface inherits the following IUnknown methods:

- IUnknown::AddRef
- IUnknown::QueryInterface
- IUnknown::Release

**IHXRmoteBroadcastConfiguration::AddDestination**

Adds a new destination to which to broadcast.

```c
STDMETHOD(AddDestination) (  
    _THIS_,  
    IHXValues* pDestination,  
    BOOL bOverwrite  
) PURE;
```
**pDestination**
Pointer to an IHXValues interface that manages the new destination to which to broadcast. The IHXValues interface that defines the new destination consists of the following fields:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>&lt;string&gt;</td>
<td>Unique user-defined name of the destination</td>
</tr>
<tr>
<td>RelayMode</td>
<td>&lt;int&gt;</td>
<td>TRUE or FALSE; should always be FALSE for remote</td>
</tr>
<tr>
<td>Protocol</td>
<td>&lt;string&gt;</td>
<td>One of the following protocol strings: &lt;udp/unicast&gt; &lt;udp/multicast&gt; &lt;tcp&gt;</td>
</tr>
<tr>
<td>Password</td>
<td>&lt;string&gt;</td>
<td>User-defined password</td>
</tr>
<tr>
<td>SecurityType</td>
<td>&lt;string&gt;</td>
<td>“None” or “Basic”</td>
</tr>
<tr>
<td>Address</td>
<td>&lt;string&gt;</td>
<td>Host name or IP address of the remote receiver</td>
</tr>
<tr>
<td>PortRange</td>
<td>&lt;string/int&gt;</td>
<td>Outbound port range to use (should match receiver)</td>
</tr>
<tr>
<td>PathPrefix</td>
<td>&lt;string&gt;</td>
<td>Prefix of the media to broadcast (can be * for wildcard)</td>
</tr>
<tr>
<td>ResendSupported</td>
<td>&lt;int&gt;</td>
<td>TRUE or FALSE</td>
</tr>
<tr>
<td>FECLevel</td>
<td>&lt;int&gt;</td>
<td>Percent redundancy to use for FEC</td>
</tr>
<tr>
<td>TTL</td>
<td>&lt;int&gt;</td>
<td>Multicast packet time to live</td>
</tr>
<tr>
<td>TCPReconectTimeout</td>
<td>&lt;int&gt;</td>
<td></td>
</tr>
<tr>
<td>TCPWouldBlockedTimeout</td>
<td>&lt;int&gt;</td>
<td></td>
</tr>
<tr>
<td>RedundancySendInterval</td>
<td>&lt;int&gt;</td>
<td></td>
</tr>
<tr>
<td>AcquisitionDataInterval</td>
<td>&lt;int&gt;</td>
<td>Rate (in seconds) at which to send header blocks for stream (re)establishment</td>
</tr>
</tbody>
</table>

**bOverwrite**
If a registry entry with the same destination name exists and this parameter is TRUE, then the existing destination information will be overwritten by the destination information provided by this method. If FALSE, an error will be returned from this method and no changes will be made to the registry.

**IHXRemoteBroadcastConfiguration::AddPullSplit**

Adds a new pull split to which to broadcast.

```cpp
STDMETHOD(AddPullSplit) (    THIS_,    IHXValues* pPullSplit,    BOOL bOverwrite
) PURE;
```
pPullSplit

Pointer to an IHXValues interface that manages the pull split to which to broadcast. The IHXValues interface that defines the pull split consists of the following fields:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>&lt;string&gt;</td>
<td>Unique user-defined name of the destination</td>
</tr>
<tr>
<td>ListenPort</td>
<td>&lt;int&gt;</td>
<td>Port on which pull split connections are accepted</td>
</tr>
<tr>
<td>LocalAddress</td>
<td>&lt;string&gt;</td>
<td>Address of interface to which to bind</td>
</tr>
<tr>
<td>PathPrefix</td>
<td>&lt;string&gt;</td>
<td>Prefix of the media to broadcast (can be * for wildcard)</td>
</tr>
<tr>
<td>Password</td>
<td>&lt;string&gt;</td>
<td>User-defined password</td>
</tr>
<tr>
<td>SecurityType</td>
<td>&lt;string&gt;</td>
<td>“None” or “Basic”</td>
</tr>
<tr>
<td>TCPReconnectTimeout</td>
<td>&lt;int&gt;</td>
<td></td>
</tr>
<tr>
<td>TCPWouldBlockedTimeout</td>
<td>&lt;int&gt;</td>
<td></td>
</tr>
</tbody>
</table>

bOverwrite

If a registry entry with the same pull split name exists and this parameter is TRUE, then the existing pull split information will be overwritten by the pull split information provided by this method. If FALSE, an error will be returned from this method and no changes will be made to the registry.

IHXRemoteBroadcastConfiguration::Commit

Makes the remote broadcast session aware of the changes made to the configuration if the remote broadcast session is already active.

STDMETHOD(Commit) ( THIS ) PURE;

IHXRemoteBroadcastConfiguration::Init

Associates a remote broadcast configuration response interface to this remote broadcast configuration interface.

STDMETHOD(Init) ( THIS, IHXRemoteBroadcastConfigurationResponse* pResponse ) PURE;

pResponse

Pointer to an IHXRemoteBroadcastConfigurationResponse interface that manages the response to the IHXRemoteBroadcastConfiguration::Commit method.

IHXRemoteBroadcastConfiguration::RemoveDestination

Removes the specified destination.
STDMETHOD(RemoveDestination) ( THIS_,
    const char* pDestinationName
) PURE;

pDestinationName
    Pointer to the name of the destination to remove.

IHXRemoteBroadcastConfiguration::RemovePullSplit

Removes the specified pull split.
STDMETHOD(RemovePullSplit) ( THIS_,
    const char* pPullSplitName
) PURE;

pPullSplitName
    Pointer to the name of the pull split to remove.

IHXRemoteBroadcastConfiguration::RetrieveDestination

Gets the destination information from the specified destination.
STDMETHOD(RetrieveDestination) ( THIS_,
    const char* pDestinationName,
    REF(IHXValues*) pDestination
) PURE;

pDestinationName
    Pointer to the name of the destination from which to retrieve information.

pDestination
    Returns a pointer to an IHXValues interface that manages the specified destination’s information.

IHXRemoteBroadcastConfiguration::RetrievePullSplit

Gets the pull split information from the specified pull split.
STDMETHOD(RetrievePullSplit) ( THIS_,
    const char* pPullSplitName,
    REF(IHXValues*) pPullSplit
) PURE;

pPullSplitName
    Pointer to the name of the pull split.

pPullSplit
    Returns a pointer to an IHXValues interface that manages the specified pull split’s information.

IHXRemoteBroadcastConfiguration::UpdateDestination

Reconfigures an existing destination.
STDMETHOD(UpdateDestination) (THIS_ IHXValues* pDestination) PURE;

pDestination
Pointer to an IHXValues interface that manages the destination information with which to update the destination.

IHXRemoteBroadcastConfiguration::UpdatePullSplit
Reconfigures an existing pull split.
STDMETHOD(UpdatePullSplit) (THIS_ IHXValues* pPullSplit) PURE;

pPullSplit
Pointer to an IHXValues interface that manages the pull split information with which to update the pull split.
IHXRemoteBroadcastConfigurationResponse

Purpose: Provides the status of asynchronous configuration modifications.
Implemented by: Remote broadcast services users (such as SLTA)
Used by: Remote broadcast services
Header file: hxbrdcst.h

This interface provides notification that the committed change to remote broadcast services are complete. This is the response interface for IHXRemoteBroadcastConfiguration.

The IHXRemoteBroadcastConfigurationResponse interface contains the IHXRemoteBroadcastConfigurationResponse::CommitDone method.

As with all Component Object Model (COM) interfaces, the IHXRemoteBroadcastConfigurationResponse interface inherits the following IUnknown methods:

• IUnknown::AddRef
• IUnknown::QueryInterface
• IUnknown::Release

IHXRemoteBroadcastConfigurationResponse::CommitDone

Provides a response to IHXRemoteBroadcastConfiguration::Commit that the commit process has completed.

STDMETHOD(CommitDone) (  
    THIS_  
    HX_RESULT hResult  
) PURE;

hResult
The status of the IHXRemoteBroadcastConfiguration::Commit operation. A value of HXR_OK indicates that the operation was completed successfully. This parameter will return HXR_FAIL if for some reason the commit could not be completed at this time.
**IHXRemoteBroadcastServices**

Purpose: Controls transmission of media streams to Helix Universal Server.

Implemented by: Remote broadcast services

Used by: Live data broadcasters

Header file: hxbrdcst.h

This interface controls and transmits real-time data streams to Helix Universal Server from an application that is independent from the server, such as a simulated live transfer agent (SLTA) or Helix DNA Producer.

The IHXRemoteBroadcastServices interface contains the following methods:

- IHXRemoteBroadcastServices::Close
- IHXRemoteBroadcastServices::GetTime
- IHXRemoteBroadcastServices::InitRemoteBroadcast—Advanced
- IHXRemoteBroadcastServices::InitRemoteBroadcast—Basic
- IHXRemoteBroadcastServices::NewRemoteBroadcastSession
- IHXRemoteBroadcastServices::Process

As with all Component Object Model (COM) interfaces, the IHXRemoteBroadcastServices interface inherits the following IUnknown methods:

- IUnknown::AddRef
- IUnknown::QueryInterface
- IUnknown::Release

**IHXRemoteBroadcastServices::Close**

Cleans up and closes the remote broadcast services.

STDMETHOD(Close) (  
    THIS  
) PURE;

**IHXRemoteBroadcastServices::GetTime**

Returns the current scheduler time, in milliseconds.

STDMETHOD_(UINT32, GetTime) (  
    THIS  
) PURE;

**IHXRemoteBroadcastServices::InitRemoteBroadcast—Advanced**

Loads an initial configuration from a configuration file for the remote broadcast plug-in. This initialization method provides authentication for remote configuration of the receiving server.
STDMETHOD(InitRemoteBroadcast) (  
    THIS_,  
    IHXBuffer* pConfigFilePath,  
    IHXAuthResponse *pAuthResponse,  
    IHXBuffer* pSessionName  
) PURE;

pConfigFilePath  
Pointer to an IHXBuffer interface that manages the directory path to the configuration file.

pAuthResponse  
Pointer to an IHXAuthResponse interface that manages authentication for the receiving server.

pSessionName  
Pointer to an IHXBuffer interface that manages the name of the remote broadcast session.

**IHXRemoteBroadcastServices::InitRemoteBroadcast—Basic**

Loads an initial configuration from a configuration file for the remote broadcast plug-in.

STDMETHOD(InitRemoteBroadcast) (  
    THIS_,  
    IHXBuffer* pConfigFilePath  
) PURE;

pConfigFilePath  
Pointer to an IHXBuffer interface that manages the directory path to the configuration file.

**IHXRemoteBroadcastServices::NewRemoteBroadcastSession**

Initiates a new remote broadcast session.

STDMETHOD(NewRemoteBroadcastSession) (  
    THIS_,  
    IHXBuffer* pSessionName,  
    IUnknown* pSessionSource  
) PURE;

pSessionName  
Pointer to an IHXBuffer interface that manages the remote broadcast session name.

pSessionSource  
Pointer to the instance of the session source.

**IHXRemoteBroadcastServices::Process**

Yields time to remote broadcast services processing.

STDMETHOD(Process) (  
    THIS  
) PURE;
IHXRenderer

Purpose: Relays client instructions to the renderer.
Implemented by: Rendering plug-in
Used by: Helix Client
Header file: hxrendr.h

All rendering plug-ins implement this interface, which manages header and packet reception, as well as stream status information. The interface receives data packets for a particular stream of a presentation and “renders” that data synchronized with the playback time line. To fulfill this role, the renderer is given access to all status events of the time line, including the current playback time.


The IHXRenderer interface contains the following methods:

- IHXRenderer::EndStream
- IHXRenderer::GetDisplayType
- IHXRenderer::GetRendererInfo
- IHXRenderer::OnBegin
- IHXRenderer::OnBuffering
- IHXRenderer::OnEndOfPackets
- IHXRenderer::OnHeader
- IHXRenderer::OnPacket
- IHXRenderer::OnPause
- IHXRenderer::OnPostSeek
- IHXRenderer::OnPreSeek
- IHXRenderer::OnTimeSync
- IHXRenderer::StartStream

As with all COM interfaces, the IHXRenderer interface inherits the following IUnknown methods:

- IUnknown::AddRef
- IUnknown::QueryInterface
- IUnknown::Release

IHXRenderer::EndStream

Informs the renderer that the stream it was rendering is completed. In this method, the renderer should deallocate its resources.

STDMETHOD(EndStream) (THIS ) PURE;
**IHXRenderer::GetDisplayType**

Returns the preferred display type for the renderer. When layout information is not present, the renderer is asked for its preferred display type. Depending on the display type, a buffer of additional information may be needed to provide information about its preferred window size. This method is called when the plug-in is being initialized.

```cpp
HRESULT GetDisplayType(
    /* [in] */ HX_DISPLAY_TYPE ulFlags,
    /* [out] */ IHXBuffer** pBuffer)
```

- **ulFlags**
  - Returns the preferred display type, which can be either of the following:
    - **HX_DISPLAY_WINDOW**
      - If the renderer renders data on the screen.
    - **HX_DISPLAY_NONE**
      - If the renderer does not use a screen because it is, for example, an audio renderer.

- **pBuffer**
  - Returns an `IHXBuffer` interface that manages any required additional information. This parameter is reserved for future use, and should be `NULL`.

**IHXRenderer::GetRendererInfo**

Returns critical information to associate this rendering plug-in with a given stream MIME type. This information tells the Helix core which renderer to use to display a particular type of stream. This method is called when the Helix core application is started.

```cpp
HRESULT GetRendererInfo(
    /* [in] */ const char** pStreamMimeTypes,
    /* [in] */ UINT32 unInitialGranularity)
```

- **pStreamMimeTypes**
  - Returns a pointer to a pointer that indicates which stream MIME types the renderer handles. The client uses this information when determining which renderer to use for a stream.

- **unInitialGranularity**
  - Returns a value that indicates how often the renderer wants to receive time line synchronization information from the client. The minimum interval is 20 milliseconds.

**IHXRenderer::OnBegin**

Informs the renderer that a begin or resume has just occurred.

```cpp
HRESULT OnBegin(
    /* [in] */ ULONG32 ulTime)
```
ulTime
The first time for the stream's time line after the resume, in milliseconds. This value is 0 (zero) if the stream is just beginning.

IHXRenderer::OnBuffering
Informs the renderer that the core is buffering all rendered data instead of presenting it. The render is also informed of the reason for the buffering (startup of stream, seek has occurred, network congestion, and so on), as well as percentage of the buffering process completed.

STDMETHOD(OnBuffering) ( 
    THIS_,
    ULONG32 ulFlags,
    UINT16 unPercentComplete
) PURE;

ulFlags
The reason for buffering. This can be any of the following:
- BUFFERING_CONGESTION
- BUFFERING_LIVE_PAUSE
- BUFFERING_SEEK
- BUFFERING_START_UP

unPercentComplete
The percentage of the buffering process that has been completed.

IHXRenderer::OnEndOfPackets
Informs the renderer that it received all of the packets. However, if the user seeks before IHXRenderer::EndStream is called, the renderer may start receiving packets again, and the client engine will eventually call this function again.

STDMETHOD(OnEndofPackets) ( 
    THIS
) PURE;

IHXRenderer::OnHeader
Retrieves a pointer to the stream header that was created by the file format plug-in. The header arrives before any packets.

STDMETHOD(OnHeader) ( 
    THIS_
    IHXValues* pHeader
) PURE;

pHeader
Pointer to an IHXValues interface that manages the stream header data for this renderer.
IHXRenderer::OnPacket

Retrieves a packet streamed by the associated file format plug-in. The client engine calls this method when a packet for this renderer is ready (or should be ready, but is lost). The rendering plug-in can render the packets directly from this method, or queue the packets and render them from another method, such as IHXRenderer::OnTimeSync.

Tip: If rendering is CPU-intensive, rendering from IHXRenderer::OnPacket may hinder the client core’s ability to deliver packets on time.

```cpp
STDMETHOD(OnPacket) (
    THIS_ 
    IHXPacket* pPacket,
    LONG32 lTimeOffset 
) PURE;
```

pPacket
Pointer to an IHXPacket interface that manages the packet from the file format plug-in.

lTimeOffset
The packet’s time offset from the start of the stream.

IHXRenderer::OnPause

Informs the renderer that a pause has just occurred.

```cpp
STDMETHOD(OnPause) (
    THIS_ 
    ULONG32 ulTime 
) PURE;
```

ulTime
The last time for the stream’s time line before the pause.

IHXRenderer::OnPostSeek

Informs the renderer that a seek has just occurred.

```cpp
STDMETHOD(OnPostSeek) (
    THIS_ 
    ULONG32 ulOldTime,
    ULONG32 ulNewTime 
) PURE;
```

ulOldTime
The last time for the stream’s time line before the seek.

ulNewTime
The first new time for the stream’s time line after the seek.

IHXRenderer::OnPreSeek

Informs the renderer that a seek is about to occur.
STDMETHOD(OnPreSeek) (  
    THIS_,  
    ULONG32 ulOldTime,  
    ULONG32 ulNewTime  
) PURE;

ulOldTime  
The last time for the stream’s time line before the seek.

ulNewTime  
The first new time for the stream’s time line after the seek is completed.

IHXRenderer::OnTimeSync

Informs the renderer of the current time relative to the stream’s synchronized time-line. The renderer should use this time value to update its display or render its stream data accordingly.

STDMETHOD(OnTimeSync) (  
    THIS_,  
    ULONG32 ulTime  
) PURE;

ulTime  
The current time relative to the stream’s synchronized time line.

IHXRenderer::StartStream

Informs the renderer of the stream it will be rendering. The stream interface provides access to its source or player, as well as to the primary client controller interface. This method is called when the plug-in is being initialized.

STDMETHOD(StartStream) (  
    THIS_,  
    IHXStream* pStream,  
    IHXPlayer* pPlayer  
) PURE;

pStream  
Pointer to an IHXStream interface that manages the stream that is to be rendered.

pPlayer  
Pointer to an IHXPlayer interface that manages the plug-in’s access to the client.

For More Information: See “Initializing” in Volume 1, on page 163.
IHXRequest

Purpose: Controls the URL request object.
Implemented by: Request object (Helix architecture)
Used by: File format plug-ins, file objects, broadcast objects
Header file: hxfiles.h

The request object contains the URL for the client request, along with the request headers and the optional response headers. Plug-ins can use the IHXRequest::GetURL and IHXRequest::SetResponseHeaders methods to get the request's fully qualified path and associate response headers with the request object, respectively. Through the response headers a plug-in can send the client any information.

For More Information: See “Modifying the Response Headers” in Volume 1, on page 66.

The IHXRequest interface contains the following methods:

• IHXRequest::GetRequestHeaders
• IHXRequest::GetResponseHeaders
• IHXRequest::GetURL
• IHXRequest::SetRequestHeaders
• IHXRequest::SetResponseHeaders
• IHXRequest::SetURL

As with all COM interfaces, the IHXRequest interface inherits the following IUnknown methods:

• IUnknown::AddRef
• IUnknown::QueryInterface
• IUnknown::Release

IHXRequest::GetRequestHeaders

Gets the headers that were sent in the RFC822 header section of the request message.

STDMETHOD(GetRequestHeaders) (  
    THIS_  
    REF(IHXValues*) pRequestHeaders  
) PURE;

pRequestHeaders
Returns a pointer to an IHXValues interface that manages the header information.

IHXRequest::GetResponseHeaders

Gets the headers that were returned in the RFC822 header section of the response message.

STDMETHOD(GetResponseHeaders) (  
    THIS_  
    REF(IHXValues*) pResponseHeaders  
) PURE;
pResponseHeaders
Returns a pointer to an IHXValues interface that manages the header information.

IHXRequest::GetURL
Returns the fully qualified path associated with a file object. On the server, this path does not include
the file system mount point.

```
STDMETHOD(GetURL) (
    THIS_
    REF(const char*) pURL
) PURE;
```

pURL
Returns a pointer to the fully qualified path associated with the file object.

**Note:** The returned pointer should be copied immediately if it is to be used later.

IHXRequest::SetRequestHeaders
Sets the headers that are to be sent in the RFC822 header section of the request message.

```
STDMETHOD(SetRequestHeaders) (
    THIS_
    IHXValues* pRequestHeaders
) PURE;
```

pRequestHeaders
Pointer to an IHXValues interface that manages the header information.

IHXRequest::SetResponseHeaders
Sets the headers that are to be returned in the RFC822 header section of the response message.

```
STDMETHOD(SetResponseHeaders) (
    THIS_
    IHXValues* pResponseHeaders
) PURE;
```

pResponseHeaders
Pointer to an IHXValues interface that manages the header information.

IHXRequest::SetURL
Sets the fully qualified path associated with an IHXFileObject. On the server, this path does not include
the file system mount point.

```
STDMETHOD(SetURL) (
    THIS_
    const char* pURL
) PURE;
```

pURL
Pointer to the fully qualified path associated with the IHXFileObject.
IHXRequestHandler

Purpose: Associates objects with request objects.
Implemented by: File objects and broadcast objects
Used by: Helix Universal Server and Helix client
Header file: hxfiles.h

A file object or broadcast object implements this interface to receive a pointer to a request object. It can then use IHXRequest to get the request URL or modify the object’s response headers. File format plug-ins do not need to implement this interface because they receive the request object pointer during initialization.

For More Information: See “Chapter 4: File Handling” beginning in Volume 1, on page 49 or “Broadcast Plug-in” in Volume 1, on page 91.

The IHXRequestHandler interface contains the following methods:
- IHXRequestHandler::GetRequest
- IHXRequestHandler::SetRequest

As with all COM interfaces, the IHXRequestHandler interface inherits the following IUnknown methods:
- IUnknown::AddRef
- IUnknown::QueryInterface
- IUnknown::Release

IHXRequestHandler::GetRequest

Gets the IHXRequest interface associated with the object calling this method.

STDMETHOD(GetRequest) ( 
THIS_ 
  REF(IHXRequest*) pRequest 
) PURE;

pRequest
Returns a pointer to the indicated IHXRequest interface.

IHXRequestHandler::SetRequest

Associates an IHXRequest interface with the object calling this method.

STDMETHOD(SetRequest) ( 
THIS_ 
  IHXRequest* pRequest 
) PURE;

pRequest
Pointer to an IHXRequest interface to associate with the object.
IHXResolver

Purpose: Resolves the Domain Name System (DNS) host name to an IP address.

Implemented by: Resolver object (Network Services)

Used by: Any component

Header file: hxengin.h

Any component that needs to resolve a DNS host name into an IP address can use this interface. The component first creates a resolver object with IHXNetworkServices. It then specifies the DNS name with this interface. The resolver object then uses IHXResolverResponse to pass the component a 4-byte IP address in native byte order.


The IHXResolver interface contains the following methods:

• IHXResolver::GetHostByName
• IHXResolver::Init

As with all COM interfaces, the IHXResolver interface inherits the following IUnknown methods:

• IUnknown::AddRef
• IUnknown::QueryInterface
• IUnknown::Release

IHXResolver::GetHostByName

Gets the Domain Name System (DNS) host name.

STDMETHOD(GetHostByName) (  
    THIS_  
    const char* pHostName  
) PURE;

pHostName
Pointer to the DNS host name.

IHXResolver::Init

Associates a resolver response interface with this interface.

STDMETHOD(Init) (  
    THIS_  
    IHXResolverResponse* pResponse  
) PURE;

pResponse
Pointer to an IHXResolverResponse interface that returns the resolved information to the calling component.
**IHXResolverResponse**

Purpose: Returns the IP address for a resolved DNS host name.

Implemented by: Any component

Used by: Resolver object (Network Services)

Header file: hxengin.h

This is the response interface to IHXResolver. Through it a component receives a 4-byte IP address in native byte order for a specified Domain Name System (DNS) host name.

**For More Information:** See “Chapter 6: Network Services” beginning in Volume 1, on page 81.

The IHXResolverResponse interface contains the IHXResolverResponse::GetHostByNameDone method.

As with all COM interfaces, the IHXResolverResponse interface inherits the following IUnknown methods:

- IUnknown::AddRef
- IUnknown::QueryInterface
- IUnknown::Release

**IHXResolverResponse::GetHostByNameDone**

Contains the IP address that was resolved from the Domain Name System (DNS) host name.

```c
STDMETHOD(GetHostByNameDone) ( 
    THIS_ 
    HX_RESULT status, 
    ULONG32 ulAddr 
) PURE;
```

- **status**
  The status of the IHXResolver::GetHostByName operation. A value of HXR_OK indicates that the operation was completed successfully.

- **ulAddr**
  A 4-byte IP address in native byte order.
Helix components use IHXRTPPacket objects to stream RTP packets between Helix Universal Server and the Helix clients. A file format plug-in, for example, prepares RTP packets that Helix Universal Server streams to the client. Also, the client’s rendering plug-in can use the system’s back channel to send back packets of information. The preferred implementation is to use IHXCommonClassFactory to create the RTP packet objects.

The IHXRTPPacket interface contains the following methods:
- IHXRTPPacket::Get
- IHXRTPPacket::GetASMFlags
- IHXRTPPacket::GetASMRuleNumber
- IHXRTPPacket::GetBuffer
- IHXRTPPacket::GetRTP
- IHXRTPPacket::GetRTPTime
- IHXRTPPacket::GetStreamNumber
- IHXRTPPacket::GetTime
- IHXRTPPacket::IsLost
- IHXRTPPacket::Set
- IHXRTPPacket::SetAsLost
- IHXRTPPacket::SetRTP

As with all COM interfaces, the IHXRTPPacket interface inherits the following IUnknown methods:
- IUnknown::AddRef
- IUnknown::QueryInterface
- IUnknown::Release

**IHXRTPPacket::Get**

Retrieves the values from the packet all at one time.

```cpp
STDMETHOD(Get) (        
    THIS_        
    REF(IHXBuffer*) pBuffer,        
    REF(UINT32) ulTime,        
    REF(UINT16) unStreamNumber,        
    REF(UINT8) unASMFlags,        
    REF(UINT16) unASMRuleNumber        
) PURE;
```

*pBuffer*

Returns a pointer to the IHXBuffer interface that manages the packet data.
ulTime
Returns the time stamp for the packet.

unStreamNumber
Returns the stream number to which the packet belongs.

unASMFlags
Returns the Adaptive Stream Management (ASM) flags associated with the packet.

unASMRuleNumber
Returns the ASM rule number of the packet.


IHXRTPPacket::GetASMFlags
Returns the ASM flags. The ASM flags can be either HX_ASM_SWITCH_ON or HX_ASM_SWITCH_OFF.

STDMETHOD_(UINT8,GetASMFlags) (THIS)
) PURE;

IHXRTPPacket::GetASMRuleNumber
Returns the ASM rule number.

STDMETHOD_(UINT16,GetASMRuleNumber) (THIS
) PURE;

IHXRTPPacket::GetBuffer
Returns an IHXBuffer interface that manages the packet data.

STDMETHOD_(IHXBuffer*,GetBuffer) (THIS
) PURE;

IHXRTPPacket::GetRTP
Retrieves the values from the RTP packet all at one time.

STDMETHOD(GetRTP) (THIS_
  REF(IHXBuffer**) pBuffer,
  REF(UINT32) ulTime,
  REF(UINT32) ulRTPTime,
  REF(UINT16) unStreamNumber,
  REF(UINT8) unASMFlags,
  REF(UINT16) unASMRuleNumber
) PURE;
pBuffer
   Returns a pointer to the IHXBuffer interface that manages the packet data.
ulTime
   Returns the time stamp for the packet.
ulRTPTime
   Returns the RTP time for the packet.
unStreamNumber
   Returns the stream number to which the packet belongs.
unASMFags
   Returns the Adaptive Stream Management (ASM) flags associated with the packet.
unASMRuleNumber
   Returns the ASM rule number of the packet.


IHXRTPPacket::GetRTPTime

Returns the RTP time.
STDMETHOD_(ULONG32,GetRTPTime) (
   THIS
) PURE;


IHXRTPPacket::GetStreamNumber

Returns the stream number.
STDMETHOD_(UINT16,GetStreamNumber) (
   THIS
) PURE;

IHXRTPPacket::GetTime

Returns the time in milliseconds.
STDMETHOD_(ULONG32,GetTime) (
   THIS
) PURE;

IHXRTPPacket::IsLost

Indicates whether the packet has been lost. If the return value is TRUE, the packet has been lost. If the packet is lost, none of the values of the packet have any meaning.
STDMETHOD_(BOOL,IsLost) (
   THIS
) PURE;
IHXRTPPacket::Set

Sets the values of the packet. This method only succeeds if there is only one reference to the packet. That is, if more than one component has a reference to the packet, you cannot use this method to change the packet’s values.

STDMETHOD(Set) ( 
    THIS_,
    IHXBuffer* pBuffer,
    UINT32 ulTime,
    UINT16 uStreamNumber,
    UINT8 unASMFlags,
    UINT16 unASMRuleNumber
) PURE;

pBuffer
    Pointer to an IHXBuffer interface that manages the packet data.

ulTime
    The time stamp for the packet.

unStreamNumber
    The stream number to which the packet belongs.

unASMFlags
    The Adaptive Stream Management (ASM) flags associated with the packet.

unASMRuleNumber
    The ASM rule number of the packet.


IHXRTPPacket::SetAsLost

Marks a packet as lost. If the packet is lost, this method is called instead of IHXPacket::Set. This method fails if the IHXBuffer of the packet is not NULL.

STDMETHOD(SetAsLost) ( 
    THIS
) PURE;

IHXRTPPacket::SetRTP

Sets the values of the RTP packet. This method only succeeds if there is only one reference to the packet. That is, if more than one component has a reference to the packet, you cannot use this method to change the packet’s values.

STDMETHOD(SetRTP) ( 
    THIS_,
    IHXBuffer* pBuffer,
    UINT32 ulTime,
    UINT32 ulRTPTime,
UINT16 uStreamNumber,
UINT8 unASMFlags,
UINT16 unASMRuleNumber
) PURE;

pBuffer
    Pointer to an IHXBuffer interface that manages the packet data.

ulTime
    The time stamp for the packet.

ulRTPTime
    The RTP time for the packet.

unStreamNumber
    The stream number to which the packet belongs.

unASMFlags
    The Adaptive Stream Management (ASM) flags associated with the packet.

unASMRuleNumber
    The ASM rule number of the packet.

IHXScheduler

Purpose: Schedules callbacks.
Implemented by: Scheduler (Helix architecture)
Used by: Any component
Header file: hxengin.h

The scheduler is a general-purpose feature that enables Helix components to schedule work for later servicing, thus enabling “lightweight” cooperative multitasking. Using IUnknown, a component queries the context for a scheduler object, and then uses the IHXScheduler methods to set an absolute or relative time for a callback. The callback is provided to the designated callback object through IHXCallback.

For More Information: See “Using the Scheduler” in Volume 1, on page 127.

The IHXScheduler interface contains the following methods:

• IHXScheduler::AbsoluteEnter
• IHXScheduler::GetCurrentSchedulerTime
• IHXScheduler::RelativeEnter
• IHXScheduler::Remove

As with all COM interfaces, the IHXScheduler interface inherits the following IUnknown methods:

• IUnknown::AddRef
• IUnknown::QueryInterface
• IUnknown::Release

IHXScheduler::AbsoluteEnter

Schedules a callback to be executed at the specified time. Returns the handle to the callback.

STDMETHOD_(CallbackHandle,AbsoluteEnter) (THIS_
   IHXCallback* pCallback,
   HXTimeval tVal
) PURE;

pCallback
   Pointer to an IHXCallback interface to be executed.

tVal
   An HXTimeval structure that contains the relevant time information.

IHXScheduler::GetCurrentSchedulerTime

Returns the current time (in the time line of the scheduler) in an HXTimeval structure.

STDMETHOD_(HXTimeval,GetCurrentSchedulerTime) (THIS
) PURE;
**IHXScheduler::RelativeEnter**

Schedules a callback to be executed a specified time from now. Returns the handle to the callback. This method is less precise than IHXScheduler::AbsoluteEnter and should only be used when accurate timing is not critical.

```cpp
STDMETHOD_(CallbackHandle, RelativeEnter) (THIS_
      IHXCallback* pCallback,
      UINT32 ms
    ) PURE;
```

- **pCallback**
  - Pointer to an IHXCallback interface to be executed.

- **ms**
  - The number of milliseconds that must elapse before the callback is executed. A relative enter of 0 (zero) milliseconds is the same as an absolute enter of 0 milliseconds.

**IHXScheduler::Remove**

Removes a specified callback from the scheduler.

```cpp
STDMETHOD(Remove) (THIS_
      CallbackHandle Handle
    ) PURE;
```

- **Handle**
  - The handle of the callback to be removed.
IHXServerAuthConversation

Purpose: Performs the server side of the authentication protocol.
Implemented by: Server authenticator manager, authentication plug-in
Used by: Encoding server, file system, and pay-per-view allowance plug-ins
Header file: hxauthn.h

This interface manages secure resources that only certain clients can access. Generally, IHXServerAuthConversation::MakeChallenge is called to pass in Real-Time Streaming Protocol (RTSP) or HTTP headers from the client to try to authenticate (the first call may not have any authentication). Then IHXServerAuthConversation::IsAuthenticated is used to find out if the client is authenticated yet. If the client is authenticated, IHXServerAuthConversation::GetUserContent is called to get the information about the user. The response interface is IHXServerAuthResponse.

The IHXServerAuthConversation interface contains the following methods:
• IHXServerAuthConversation::GetUserContext
• IHXServerAuthConversation::IsAuthenticated
• IHXServerAuthConversation::MakeChallenge

As with all COM interfaces, the IHXServerAuthConversation interface inherits the following IUnknown methods:
• IUnknown::AddRef
• IUnknown::QueryInterface
• IUnknown::Release

IHXServerAuthConversation::GetUserContext

Retrieves an object that provides additional information about the user. Generally, this object implements the IHXUserContext interface, which is useful in NT LAN Manager (NTLM) authentication for determining whether a user belongs to a certain Windows NT administrative group in the NT security model. This object may also implement the IHXUserImpersonation interface, which enables a Windows NT process to temporarily have the same security as another Windows NT user. In addition, this object may also implement the IHXUserProperties interface, which can get the user’s name for logging or other purposes.

STDMETHOD(GetUserContext) (THIS_
   REF(IUnknown*) pUnknownUser
) PURE;

pUnknownUser
Pointer to a user context that provides more information about the user.

IHXServerAuthConversation::IsAuthenticated

Determines whether the last response from the client completed the authentication successfully. Returns TRUE when the authentication is completed successfully.
STDMETHOD_(BOOL, IsAuthenticated) (THIS) PURE;

**IHXServerAuthConversation::MakeChallenge**

Creates a challenge for a client. If the request passed in does not contain a response from the client, then it will generate the initial challenge.

STDMETHOD(MakeChallenge) (THIS
   IHXServerAuthResponse* pServerAuthResponseRequester,
   IHXRequest* pRequestResponseHeaders
) PURE;

**pServerAuthResponseRequester**

Pointer to an IHXServerAuthResponse interface that manages the responses generated by this method.

**pRequestResponseHeaders**

Pointer to an IHXRequest interface that manages the request for a secured URL. If this is the initial request for the URL, this method probably does not have any credentials from the client.
**IHXServerAuthResponse**

Purpose: Provides the request challenge information.
Implemented by: Server plug-in
Used by: Server core
Header file: hauthn.h

This interface handles the response to the `IHXServerAuthConversation::MakeChallenge` method.

The `IHXServerAuthResponse` interface contains the `IHXServerAuthResponse::ChallengeReady` method.

As with all COM interfaces, the `IHXServerAuthResponse` interface inherits the following `IUnknown` methods:
- `IUnknown::AddRef`
- `IUnknown::QueryInterface`
- `IUnknown::Release`

**IHXServerAuthResponse::ChallengeReady**

Reports the success or failure of `IHXServerAuthConversation::MakeChallenge`.

```c
STDMETHOD(ChallengeReady) ( 
    THIS_ 
    HX_RESULT ResultStatus, 
    IHXRequest* pRequestChallengeHeaders 
) PURE;
```

**ResultStatus**
The status of the `IHXServerAuthConversation::MakeChallenge` operation. A value of HXR_OK indicates that the operation was completed successfully.

**pRequestResponseHeaders**
Pointer to an IHXRequest interface that manages the request challenge headers. This IHXRequest interface should be the same as that passed in `IHXServerAuthConversation::MakeChallenge`, and should contain CString values for each MIME header that needs to be sent to the client.
IHXServerControl

Purpose: Terminates all connections and shuts down the server.
Implemented by: Server core
Used by: Server plug-in
Header file: hxcomm.h

This interface is used when a plug-in wants to shut down the server.
The IHXServerControl interface contains the IHXServerControl::ShutdownServer method.
As with all COM interfaces, the IHXServerControl interface inherits the following IUnknown methods:
  • IUnknown::AddRef
  • IUnknown::QueryInterface
  • IUnknown::Release

IHXServerControl::ShutdownServer

Quits the server program.
STDMETHOD(ShutdownServer) (  
    THIS_  
    UINT32 status  
) PURE;

status
    Indicates whether or not the shutdown was completed successfully.
IHXServerControl2

Purpose: Controls various server processes
Implemented by: Server core
Used by: Server plug-in
Header file: hxcomm.h

This interface is provided in addition to IHXServerControl to extend the control over various server processes.
The IHXServerControl2 interface contains the following methods:
  • IHXServerControl2::ReconfigServer
  • IHXServerControl2::RestartServer
As with all COM interfaces, the IHXServerControl2 interface inherits the following IUnknown methods:
  • IUnknown::AddRef
  • IUnknown::QueryInterface
  • IUnknown::Release

IHXServerControl2::ReconfigServer

Causes the server to reread in the configuration from a file or registry (however it was started) and attempt to use the values.

STDMETHOD(ReconfigServer) ( 
  THIS_
  IHXReconfigServerResponse* pResp
) PURE;

pResp
  Pointer to an IHXReconfigServerResponse interface that manages the response to this method.

IHXServerControl2::RestartServer

Directs the server to completely shut down, then restart. This method is used primarily so that changes that require a server restart take effect.

STDMETHOD(RestartServer) ( 
  THIS
) PURE;
IHXServerFork

Purpose: Forks UNIX processes.

Implemented by: Helix architecture

Used by: UNIX plug-ins

Header file: hxcomm.h

On UNIX platforms, this interface enables plug-ins to fork processes. A forked process cannot use any Helix interfaces, however. For more information, see hxcomm.h.

The IHXServerFork interface contains the IHXServerFork::Fork method.

As with all COM interfaces, the IHXServerFork interface inherits the following IUnknown methods:

• IUnknown::AddRef
• IUnknown::QueryInterface
• IUnknown::Release

IHXServerFork::Fork

Forks to create a child process. The child process cannot use any Helix APIs. Upon successful completion, this method returns 0 (zero) to the child process and the PID of the child to the parent. A return value of -1 indicates an error.

STDMETHOD_(INT32, Fork) (THIS)

Note: The child process should not release (IUnknown::Release) any interfaces. The cleanup of the IHXServerFork interface and other HX interfaces is performed by the parent.
IHXServerReconfigNotification

Purpose: Provides notification of a reconfiguration request.
Implemented by: Server core
Used by: Server plug-in
Header file: hxcomm.h

This interface provides a way for you to register with the server that you want notification when a reconfiguration request comes in and need to take part in the reconfiguration. For example, you might need to participate in the reconfiguration if you have some configuration information saved outside the server configuration file that needs to be re-initialized.

The IHXServerReconfigNotification interface contains the following methods:

- IHXServerReconfigNotification::CancelReconfigNotification
- IHXServerReconfigNotification::WantReconfigNotification

As with all COM interfaces, the IHXServerReconfigNotification interface inherits the following IUnknown methods:

- IUnknown::AddRef
- IUnknown::QueryInterface
- IUnknown::Release

IHXServerReconfigNotification::CancelReconfigNotification

Indicates to the server that you no longer want reconfiguration notification.

STDMETHOD(CancelReconfigNotification) (THIS_ IHXWantServerReconfigNotification* pResponse) PURE;

pResponse
Pointer to an IHXWantServerReconfigNotification interface that manages the reconfiguration notification for a component that has to do manual reconfiguration. On UNIX systems, this is currently only activated by a SIGHUP.

IHXServerReconfigNotification::WantReconfigNotification

Indicates to the server that you want reconfiguration notification.

STDMETHOD(WantReconfigNotification) (THIS_ IHXWantServerReconfigNotification* pResponse) PURE;

pResponse
Pointer to an IHXWantServerReconfigNotification interface that manages the reconfiguration notification for a component that has to do manual reconfiguration. On UNIX systems, this is currently only activated by a SIGHUP.
IHXSetSocketOption

Purpose: Sets the state of the socket to the new state.
Implemented by: Server core
Used by: Server plug-ins
Header file: hxengin.h

The IHXSetSocketOption interface contains the IHXSetSocketOption::SetOption method. As with all COM interfaces, the IHXSetSocketOption interface inherits the following IUnknown methods:
• IUnknown::AddRef
• IUnknown::QueryInterface
• IUnknown::Release

IHXSetSocketOption::SetOption
Sets the socket to the new state.

STDMETHOD(SetOption) ( 
    THIS_ 
    HX_SOCKET_OPTION option, 
    UINT32 ulValue 
) PURE;

option
The option to be set. This can be any of the following:
• HX_SOCKOPT_REUSE_ADDR
  Allows local address reuse.
• HX_SOCKOPT_REUSE_PORT
  Allows local port reuse.
• HX_SOCKOPT_BROADCAST
  Allows sending broadcast messages.
• HX_SOCKOPT_SET_RECVBUF_SIZE
  Sets the network's receive buffer size.
• HX_SOCKOPT_SET_SENDBUF_SIZE
  Sets the network's send buffer size.
• HX_SOCKOPT_MULTICAST_IF
  Sets the network interface to be used for multicasting.

ulValue
The value of the option selected in option.
IHXS

Purpose: Controls site rendering.
Implemented by: Site object (client core)
Used by: Display renderer
Header file: hxwin.h

This is the main interface to a site in which a display rendering plug-in renders data. The methods in this interface enable the plug-in to control the site size and position, attach site watches with IHXSWatcher, and create child windows.

For More Information: See “Sites (Windowing)” in Volume 1, on page 183.

The IHXS interface contains the following methods:

• IHXS:AttachUser
• IHXS:AttachWatcher
• IHXS:CreateChild
• IHXS:DamageRect
• IHXS:DamageRegion
• IHXS:DestroyChild
• IHXS:DetachUser
• IHXS:DetachWatcher
• IHXS:ForceRedraw
• IHXS:GetPosition
• IHXS:GetSize
• IHXS:GetUser
• IHXS:SetPosition
• IHXS:SetSize

As with all COM interfaces, the IHXS interface inherits the following IUnknown methods:

• IUnknown: :AddRef
• IUnknown: :QueryInterface
• IUnknown: :Release

IHXS::AttachUser

Associates a site user interface with this site.

METHOD(AttachUser) (THIS_ IHXSUser* pUser
) PURE;

pUser
  Pointer to an IHXSUser interface associated with this site.
**IHXSite::AttachWatcher**

Associates a site watcher interface with the current site.

```cpp
STDMETHOD(AttachWatcher) ( 
    THIS_ 
    IHXSiteWatcher* pWatcher 
) PURE;
```

*pWatcher*  
Pointer to an IHXSiteWatcher interface that is being associated with the current site.

**IHXSite::CreateChild**

Creates a child site associated with the current site.

```cpp
STDMETHOD(CreateChild) ( 
    THIS_ 
    REF(IHXSite*) pChildSite 
) PURE;
```

*pChildSite*  
Returns a pointer to an IHXSite interface that manages the child site.

**IHXSite::DamageRect**

Sets aside a rectangle to be filled in the future.

```cpp
STDMETHOD(DamageRect) ( 
    THIS_ 
    HXxRect rect 
) PURE;
```

*rect*  
A HXxRect structure that defines the rectangle.

**IHXSite::DamageRegion**

Sets aside a region (a collection of rectangles) to be filled in the future.

```cpp
STDMETHOD(DamageRegion) ( 
    THIS_ 
    HXxRegion region 
) PURE;
```

*region*  
A HXxRegion that defines the region.

**IHXSite::DestroyChild**

Removes the indicated child site.
STDMETHOD(DestroyChild) (  
    THIS_  
    IHXSite* pChildSite  
) PURE;

pChildSite
    Pointer to an IHXSite interface that manages the child site to be removed.

**IHXSite::DetachUser**

    Detaches the site user from the site.
    STDMETHOD(DetachUser) (  
        THIS  
    ) PURE;

**IHXSite::DetachWatcher**

    Detaches the site watcher from the site.
    STDMETHOD(DetachWatcher) (  
        THIS  
    ) PURE;

**IHXSite::ForceRedraw**

    Forces a redraw of the site.
    STDMETHOD(ForceRedraw) (  
        THIS  
    ) PURE;

**IHXSite::GetPosition**

    Gets the current position.
    STDMETHOD(GetPosition) (  
        THIS_  
        REF(HXxPoint) position  
    ) PURE;

    position
        Returns a HXxPoint structure that indicates the current position.

**IHXSite::GetSize**

    Gets the size of the site.
    STDMETHOD(GetSize) (  
        THIS_  
        REF(HXxSize) size  
    ) PURE;

    size
        Returns a HXxSize structure that indicates the size of the site.
**IHXSite::GetUser**

Gets the site user associated with this site.

```
STDMETHOD(GetUser) (
    THIS_
    REF(IHXSiteUser*) pUser
) PURE;
```

*pUser*

Returns a pointer to an IHXSiteUser interface that manages the user.

**IHXSite::SetPosition**

Sets the position on the site. As a general rule, site users should not call this method.

```
STDMETHOD(SetPosition) (
    THIS_
    HXxPoint position
) PURE;
```

*position*

A HXxPoint structure that defines the point to set the position.

**IHXSite::SetSize**

Sets the size of the site.

```
STDMETHOD(SetSize) (
    THIS_
    HXxSize size
) PURE;
```

*size*

A HXxSize structure that contains the size, in pixels.
IHXSite2

Purpose: Controls site rendering.
Implemented by: Site object (client core)
Used by: Display renderer
Header file: hxsite2.h

This interface complements the IHXSite interface, permitting cross-platform image rendering.


The IHXSite2 interface contains the following methods:

• IHXSite2::AddPassiveSiteWatcher
• IHXSite2::GetNumberOfChildSites
• IHXSite2::GetVideoSurface
• IHXSite2::GetZOrder
• IHXSite2::IsSiteVisible
• IHXSite2::MoveSiteToTop
• IHXSite2::RemovePassiveSiteWatcher
• IHXSite2::SetCursor
• IHXSite2::SetZOrder
• IHXSite2::ShowSite
• IHXSite2::UpdateSiteWindow

As with all COM interfaces, the IHXSite2 interface inherits the following IUnknown methods:

• IUnknown::AddRef
• IUnknown::QueryInterface
• IUnknown::Release

IHXSite2::AddPassiveSiteWatcher

Adds a watcher that does not affect the site.

STDMETHOD(AddPassiveSiteWatcher) (THIS
  IHXPassiveSiteWatcher* pWatcher)
) PURE;

pWatcher
  Pointer to an IHXPassiveSiteWatcher interface that the site notifies of a site change.

IHXSite2::GetNumberOfChildSites

Returns the number of child sites.

STDMETHOD_(UINT32,GetNumberOfChildSites) (THIS
  ) PURE;
IHXSite2::GetVideoSurface

    Gets the site's video surface.
    STDMETHOD(GetVideoSurface) (  
        THIS_  
        REF(IHXVideoSurface*) pSurface  
    ) PURE;
    pSurface
    Returns a pointer to an IHXVideoSurface interface that manages the video surface.

IHXSite2::GetZOrder

    Gets the site's z-order.
    STDMETHOD(GetZOrder) (  
        THIS_  
        REF(INT32) lZOrder  
    ) PURE;
    lZOrder
    Returns the site's z-order.

IHXSite2::IsSiteVisible

    Indicates whether the site is visible. If this method returns TRUE, the site is visible.
    STDMETHOD_(BOOL, IsSiteVisible) (  
        THIS  
    ) PURE;

IHXSite2::MoveSiteToTop

    Sets the site at the top of the z-order.
    STDMETHOD(MoveSiteToTop) (  
        THIS  
    ) PURE;

IHXSite2::RemovePassiveSiteWatcher

    Removes a watcher that does not affect the site.
    STDMETHOD(RemovePassiveSiteWatcher) (  
        THIS_  
        IHXPassiveSiteWatcher* pWatcher  
    ) PURE;
    pWatcher
    Pointer to the IHXPassiveSiteWatcher interface to remove.

IHXSite2::SetCursor

    Sets the cursor type. This method is not currently implemented.
STDMETHOD(SetCursor) (
    THIS_
    HXxCursor ulCursor,
    REF(HXxCursor) ulOldCursor
) PURE;

ulCursor
    A HXxCursor that defines the new cursor type.

ulOldCursor
    Returns a HXxCursor that defines the previous cursor type.

**IHXSite2::SetZOrder**

Sets the site’s z-order.

STDMETHOD(SetZOrder) (
    THIS_
    INT32 lZOrder
) PURE;

lZOrder
    The site’s z-order.

**IHXSite2::ShowSite**

Shows or hides a site.

STDMETHOD(ShowSite) (
    THIS_
    BOOL bShow
) PURE;

bShow
    If TRUE, the site is shown; if FALSE, the site is hidden.

**IHXSite2::UpdateSiteWindow**

Updates the site window attributes (such as the window handle, position, size, and so on) after they have been changed.

STDMETHOD(UpdateSiteWindow) (
    THIS_
    HXxWindow* pWindow
) PURE;

pWindow
    A pointer to a HXxWindow structure that contains the new attributes.
IHXSiteEnumerator

Purpose: Provides an interface to enumerate sites.
Implemented by: Client core
Used by: Rendering plug-ins
Header file: hxwin.h

This interface allows you to iterate through all the sites in the system. Therefore, you can use this interface if, for example, you need to perform some action on every existing site.

The IHXSiteEnumerator interface contains the following methods:

- IHXSiteEnumerator::GetFirstSite
- IHXSiteEnumerator::GetNextSite

As with all Component Object Model (COM) interfaces, the IHXSiteEnumerator interface inherits the following IUnknown methods:

- IUnknown::AddRef
- IUnknown::QueryInterface
- IUnknown::Release

IHXSiteEnumerator::GetFirstSite

Retrieves both the first site in the enumeration and the position of the next site (if any). This method returns HXR_OK if the first site is available or HXR_FAIL if the first site is not available.

STDMETHOD(GetFirstSite) (THIS_
    REF(IHXSite*) pFirstSite,
    REF(SitePosition) nextPosition
) PURE;

pFirstSite
    Returns a pointer to an IHXSite interface that manages the first site to be enumerated.

nextPosition
    Returns the position of the next site (if any).

IHXSiteEnumerator::GetNextSite

Retrieves both the next site in the enumeration and the position of the following site (if any). Returns HXR_OK if the site is available or HXR_FAIL if the site is not available.

STDMETHOD(GetNextSite) (THIS_
    REF(IHXSite*) pNextSite,
    REF(SitePosition) nextPosition
) PURE;

pNextSite
    Returns a pointer to an IHXSite interface that manages the site to be enumerated.
nextPosition

Returns the position of the next site (if any).
IHXSiteFullScreen

Purpose: Turns full-screen mode on or off.
Implemented by: Client core
Used by: Top-level client
Header file: hxwin.h

Interface implemented by the client core for use by the top-level client.

For More Information: See “Sites (Windowing)” in Volume 1, on page 183.

The IHXSiteFullScreen interface contains the following methods:
• IHXSiteFullScreen::EnterFullScreen
• IHXSiteFullScreen::ExitFullScreen
• IHXSiteFullScreen::IsFullScreen
• IHXSiteFullScreen::TestFullScreen

As with all COM interfaces, the IHXSiteFullScreen interface inherits the following IUnknown methods:
• IUnknown::AddRef
• IUnknown::QueryInterface
• IUnknown::Release

IHXSiteFullScreen::EnterFullScreen

Enters full-screen mode.

STDMETHOD(EnterFullScreen) (THIS) PURE;

IHXSiteFullScreen::ExitFullScreen

Exits full-screen mode.

STDMETHOD(ExitFullScreen) (THIS) PURE;

IHXSiteFullScreen::IsFullScreen

If the return value is TRUE, the site is in full-screen mode.

STDMETHOD_(BOOL, IsFullScreen) (THIS) PURE;
**IHXSiteFullScreen::TestFullScreen**

Tests full-screen mode by displaying a bitmap and determining how fast it can transfer bit blocks, or “blit.”

```c
STDMETHOD(TestFullScreen) ( 
    THIS_ 
    void* hTestBitmap, 
    const char* pszStatusText 
) PURE;
```

**hTestBitmap**

Pointer to the bitmap to be blitted.

**pszStatusText**

Pointer to the text that is displayed at the bottom of the test screen.
The IHXSiteManager interface contains the following methods:

- IHXSiteManager::AddSite
- IHXSiteManager::RemoveSite

As with all COM interfaces, the IHXSiteManager interface inherits the following IUnknown methods:

- IUnknown::AddRef
- IUnknown::QueryInterface
- IUnknown::Release

**IHXSiteManager::AddSite**

Informs the site manager of the existence of a site.

```cpp
STDMETHOD(AddSite) (
    THIS_ 
    IHXSite* pSite
) PURE;
```

`pSite`  
Pointer to an IHXSite interface that manages the site that was added.

**IHXSiteManager::RemoveSite**

Informs the site manager that a site is no longer available.

```cpp
STDMETHOD(RemoveSite) (
    THIS_ 
    IHXSite* pSite
) PURE;
```

`pSite`  
Pointer to an IHXSite interface that managed the site that was removed.
**IHXSiteSupplier**

**Purpose:** Informs the top-level client when sites are needed or changed.

**Implemented by:** Top-level client

**Used by:** Client core

**Header file:** hxwin.h

Interface implemented by the top-level client for use by the client core.

**For More Information:** See “Sites (Windowing)” in Volume 1, on page 183.

The IHXSiteSupplier interface contains the following methods:

- IHXSiteSupplier::BeginChangeLayout
- IHXSiteSupplier::DoneChangeLayout
- IHXSiteSupplier::SitesNeeded
- IHXSiteSupplier::SitesNotNeeded

As with all COM interfaces, the IHXSiteSupplier interface inherits the following IUnknown methods:

- IUnknown::AddRef
- IUnknown::QueryInterface
- IUnknown::Release

**IHXSiteSupplier::BeginChangeLayout**

Informs the site supplier that a layout change has begun and that it can expect to receive calls to the IHXSiteSupplier::SitesNeeded and IHXSiteSupplier::SitesNotNeeded methods while a layout change is in progress.

```cpp
STDMETHOD(BeginChangeLayout) (
    THIS
) PURE;
```

**IHXSiteSupplier::DoneChangeLayout**

Informs the site supplier that a layout change was completed.

```cpp
STDMETHOD(DoneChangeLayout) (
    THIS
) PURE;
```

**IHXSiteSupplier::SitesNeeded**

Informs the site supplier that a site with a particular set of characteristics is needed. If the site supplier can fulfill the request, it should call the site manager and add one or more new sites. The request for sites is associated with a request ID. The client core will inform the site supplier when this requested site is no longer needed.

```cpp
```
STDMETHOD(SitesNeeded) (  
    THIS_,
    UINT32 uRequestID,
    IHXValues* pSiteProps
  ) PURE;

  uRequestID  
  The request ID used to map between corresponding “sites needed” and “sites not needed” calls.

  pSiteProps  
  Pointer to an IHXValues interface that manages the site properties for the requested sites.

IHXSitesSupplier::SitesNotNeeded

Informs the site supplier that all sites from a previous site request are no longer needed. If the site supplier had previously created nonpersistent sites (such as pop-up windows) to fulfill a request for sites, it should call the site manager and remove those sites.

STDMETHOD(SitesNotNeeded) (  
    THIS_,
    UINT32 uRequestID
  ) PURE;

  uRequestID  
  The request ID used to map between corresponding “sites needed” and “sites not needed” calls.
**IHXSiteUser**

Purpose: Associates a display renderer with a site and informs the renderer of events.

Implemented by: Display renderer

Used by: Client core

Header file: hxwin.h

A display rendering plug-in implements this interface, which the client uses to associate the renderer with a site object and inform it of events. The client queries for this interface if, during initialization, the renderer declares itself to use windows. The renderer’s response to `IHXSiteUser::NeedsWindowedSite` determines whether it uses `IHXSiteWindowed` or `IHXSiteWindowless` along with `IHXSite` to render data to the site.

*For More Information:* See “Sites (Windowing)” in Volume 1, on page 183. See also `IHXSiteUserSupplier` in Volume 2, on page 399.

The `IHXSiteUser` interface contains the following methods:

- `IHXSiteUser::AttachSite`
- `IHXSiteUser::DetachSite`
- `IHXSiteUser::HandleEvent`
- `IHXSiteUser::NeedsWindowedSites`

As with all COM interfaces, the `IHXSiteUser` interface inherits the following `IUnknown` methods:

- `IUnknown::AddRef`
- `IUnknown::QueryInterface`
- `IUnknown::Release`

**IHXSiteUser::AttachSite**

Associates a site interface with the site user interface.

```c++
STDMETHOD(AttachSite) ( 
    IHXSite* pSite 
) PURE;
```

**pSite**

Pointer to an `IHXSite` interface that is to be associated with this `IHXSiteUser` interface.

**IHXSiteUser::DetachSite**

Removes the site interface that was previously associated with this site user interface.

```c++
STDMETHOD(DetachSite) ( 
    IHXSite* pSite 
) PURE;
```
IHXS\texttt{SiteUser::HandleEvent}

Informs the renderer of an event.

\texttt{STDMETHOD(\text{HandleEvent}) (}
\texttt{THIS_}
\texttt{HXxEvent* \ pEvent}
\texttt{)} \texttt{PURE;}

\texttt{pEvent}
Point to a HXxEvent structure that contains a description of the event.

\textit{For More Information:} See “Handling Events” in Volume 1, on page 186.

IHXS\texttt{SiteUser::NeedsWindowedSites}

Returns \texttt{TRUE} if the plug-in will render to an operating system-specific window (Windows or UNIX). Returns \texttt{FALSE} if the plug-in will render to a windowless site.

\texttt{STDMETHOD_(BOOL,\text{NeedsWindowedSites}) (}
\texttt{THIS}
\texttt{)} \texttt{PURE;}


The \texttt{IHXSiteUserSupplier} interface contains the following methods:

- \texttt{IHXSiteUserSupplier::CreateSiteUser}
- \texttt{IHXSiteUserSupplier::DestroySiteUser}
- \texttt{IHXSiteUserSupplier::NeedsWindowedSites}

As with all COM interfaces, the \texttt{IHXSiteUserSupplier} interface inherits the following \texttt{IUnknown} methods:

- \texttt{IUnknown::AddRef}
- \texttt{IUnknown::QueryInterface}
- \texttt{IUnknown::Release}

### \texttt{IHXSiteUserSupplier::CreateSiteUser}

Creates a site user interface that will be attached to a site.

\begin{verbatim}
STDMETHOD(CreateSiteUser) ( 
  THIS_ 
  REF(IHXSiteUser*) pSiteUser
) PURE;
\end{verbatim}

\texttt{pSiteUser}

Returns a pointer to an \texttt{IHXSiteUser} interface that manages the site user.

### \texttt{IHXSiteUserSupplier::DestroySiteUser}

Releases the site user interface.

\begin{verbatim}
STDMETHOD(DestroySiteUser) ( 
  THIS_ 
  IHXSiteUser* pSiteUser
) PURE;
\end{verbatim}
pSiteUser

Pointer to the IHXSiteUser interface to be released.

IHXSiteUserSupplier::NeedsWindowedSites

Returns TRUE if the plug-in needs a windowed site; otherwise, returns FALSE.

STDMETHOD_(BOOL, NeedsWindowedSites) (THIS)

) PURE;
A display rendering plug-in implements this interface to monitor and, if necessary, override changes in a site window’s size or position. It attaches itself as a watcher with IHXSite::AttachWatcher. The site then informs the renderer of size and position changes through IHXSiteWatcher. The renderer can override these changes with IHXSite methods.


The IHXSiteWatcher interface contains the following methods:

- IHXSiteWatcher::AttachSite
- IHXSiteWatcher::ChangingPosition
- IHXSiteWatcher::ChangingSize
- IHXSiteWatcher::DetachSite

As with all Component Object Model (COM) interfaces, the IHXSiteWatcher interface inherits the following IUnknown methods:

- IUnknown::AddRef
- IUnknown::QueryInterface
- IUnknown::Release

IHXSiteWatcher::AttachSite

Attaches a site interface to this site watcher interface.

STDMETHOD(AttachSite) (
    THIS_ IHXSite* pSite
) PURE;

pSite
    Pointer to an IHXSite interface that is to be attached.

IHXSiteWatcher::ChangingPosition

Indicates an attempt is being made to change the site’s position. The site watcher must return HXR_OK for the change to occur. If the site watcher returns any value other than HXR_OK, the position does not change. The site watcher can also modify the new position.

STDMETHOD(ChangingPosition) ( 
    THIS_ HXxPoint posOld, 
    REF(HXxPoint) posNew
) PURE;

Purpose: Informs the renderer of site size and position changes.
Implemented by: Display renderer
Used by: Site object (client core)
Header file: hxwin.h
posOld
   An HXxPoint structure that indicates the previous position of the site.

posNew
   Returns an HXxPoint structure that indicates the new position.

IHXSiteWatcher::ChangingSize

Indicates an attempt is being made to change the site’s size. The site watcher must return HXR_OK for
the change to occur. If the site watcher returns any value other than HXR_OK, the size does not change.
The site watcher can also modify the new size.

STDMETHOD(ChangingSize) (  
   THIS  
   HXxSize sizeOld,  
   REF(HXxSize) sizeNew 
) PURE;

sizeOld
   An HXxSize structure that indicates the previous size of the site.

sizeNew
   Returns an HXxSize structure that indicates the new size.

IHXSiteWatcher::DetachSite

Detaches the site interface that was previously attached to this site watcher.

STDMETHOD(DetachSite) (  
   THIS 
) PURE;
IHXS Site Windowed

Purpose: Controls windowed site rendering.
Implemented by: Windowed site object (client core)
Used by: Display renderer
Header file: hxwin.h

A display rendering plug-in uses this interface along with IHXS Site if it responds with TRUE to IHXS SiteUser::NeedsWindowedSites. The IHXS SiteWindowed interface provides operating system-specific methods for manipulating sites.

For More Information: See “Sites (Windowing)” in Volume 1, on page 183.

The IHXS SiteWindowed interface contains the following methods:

- IHXS SiteWindowed::AttachWindow
- IHXS SiteWindowed::Create
- IHXS SiteWindowed::Destroy
- IHXS SiteWindowed::DetachWindow
- IHXS SiteWindowed::GetWindow

As with all Component Object Model (COM) interfaces, the IHXS SiteWindowed interface inherits the following IUnknown methods:

- IUnknown::AddRef
- IUnknown::QueryInterface
- IUnknown::Release

IHXS SiteWindowed::AttachWindow

Associates a previously created, and externally managed, window with the site. In Windows, this method will “subclass” that window. In UNIX, the site supplier must pass events from the externally managed window to the core using IHX ClientEngine::EventOccurred. Note that the pointer to the HXxWindow structure must remain in scope for the life of the site, that is, the pointer must not be a stack variable.

 STDMETHODCALLTYPE(AttachWindow) ( 
    THIS_
    HXxWindow* pWindow
) PURE;

Pointer to an HXxWindow structure that contains the window information.

IHXS SiteWindowed::Create

Creates a default, top-level window for the site.
STDMETHOD(Create) (  
  THIS  
  void* ParentWindow,  
  UINT32 style  
) PURE;

ParentWindow
Pointer to the parent window. If this parameter is NULL, the site has no parent, that is, it will be a “floating” window. To have the site in a user interface, a parent must be given.

style
The style of the window. The value of this parameter is native to the operating system. For example, in Windows this parameter can be WS_OVERLAPPED, WS_VISIBLE, WS_CLIPCHILDREN, and so on.

IHXSiteWindowed::Destroy
Destroys the previously created top-level window for the site.

STDMETHOD(Destroy) (  
  THIS  
) PURE;

IHXSiteWindowed::DetachWindow
Detaches a previously attached window from the site.

STDMETHOD(DetachWindow) (  
  THIS  
) PURE;

IHXSiteWindowed::GetWindow
Returns the actual window of the site in the indicated HXxWindow structure.

STDMETHOD_(HXxWindow*, GetWindow) (  
  THIS  
) PURE;
**IHXSiteWindowless**

Purpose: Controls windowless site rendering.
Implemented by: Windowless site object (client core)
Used by: Display renderer
Header file: hxwin.h

A display rendering plug-in uses this interface along with IHXSite if it responds with FALSE to IHXSiteUser::NeedsWindowedSites. The IHXSiteWindowless interface provides operating system-generic methods for manipulating sites.

**For More Information:** See “Sites (Windowing)” in Volume 1, on page 183.

The IHXSiteWindowless interface contains the following methods:

- IHXSiteWindowless::EventOccurred
- IHXSiteWindowless::GetParentWindow

As with all Component Object Model (COM) interfaces, the IHXSiteWindowless interface inherits the following IUnknown methods:

- IUnknown::AddRef
- IUnknown::QueryInterface
- IUnknown::Release

**IHXSiteWindowless::EventOccurred**

Indicates that a native operating system event occurred.

```c
STDMETHOD(EventOccurred) (  
  THIS_  
  HXxEvent* pEvent  
) PURE;
```

`pEvent` An HXxEvent structure that describes the event.

**IHXSiteWindowless::GetParentWindow**

Returns a parent window that owns the windowless site in the indicated HXxWindow structure. This method is useful for right-click menus and dialog box calls.

```c
STDMETHOD_(HXxWindow*,GetParentWindow) (  
  THIS  
) PURE;
```
IHXSLTA

**Note:** The IHXSLTA interface is obsolete and should not be used in any new programming. Use the IHXIQSLTA interface in a simulated live transfer agent application instead.

| Purpose: | Simulates a live stream from a file. |
| Implemented by: | Helix simulated live transfer agent (SLTA) library |
| Used by: | Stand-alone executable files |
| Header file: | hxslta.h |

Helix Universal Server includes the standard SLTA application that simulates live broadcasts. This interface provides a means of designing custom applications that deviate from the standard application.

**For More Information:** See the broadcast simulation chapter in *Helix Universal Server Administration Guide* and “Simulated Live Transfer Agent” in Volume 1, on page 101 in this guide.

The IHXSLTA interface contains the following methods:

- IHXSLTA::Connect
- IHXSLTA::Disconnect
- IHXSLTA::Encode
- IHXSLTA::SetTAC
- IHXSLTA::SetTargetBandwidth

As with all COM interfaces, the IHXSLTA interface inherits the following IUnknown methods:

- IUnknown::AddRef
- IUnknown::QueryInterface
- IUnknown::Release

**IHXSLTA::Connect**

Connects the simulated live transfer agent (SLTA) to a server.

```c
STDMETHOD(Connect) (
    THIS_
    const char* host,
    UINT16 uPort,
    const char* username,
    const char* passwd,
    const char* livefile
) PURE;

host
    Pointer to the host to which you want to connect.
```
**uPort**
The port to which you want to connect.

**username**
Pointer to the user name supplied to make the connection.

**passwd**
Pointer to the password supplied to make the connection.

**livefile**
Pointer to the file name of the live stream on the server that will be requested by the player.

**IHXSLTA::Disconnect**
Disconnects the simulated live transfer agent (SLTA) from the server.

```cpp
STDMETHOD(Disconnect) (
    THIS
) PURE;
```

**IHXSLTA::Encode**
Starts streaming the file to the server.

```cpp
STDMETHOD(Encode) (
    THIS const char* filename
) PURE;
```

**filename**
Pointer to the name of the file to be streamed.

**IHXSLTA::SetTAC**
Sets the title, author, and copyright information for the stream. This method must be called before **IHXSLTA::Encode** will have any effect.

**IHXSLTA::SetTAC** overrides the displayed title, author, and copyright. This method does not need to be used unless, for example, you have several items to play and you want to override the given title, author, and copyright with your own information. If you do not use this method, the title, author, and copyright for each individual file will still be displayed.

```cpp
STDMETHOD(SetTAC) (
    THIS const char* Title,
    const char* Author,
    const char* Copyright
) PURE;
```

**Title**
Pointer to the title information for the stream.

**Author**
Pointer to the author information for the stream.
Copyright

Pointer to the copyright information for the stream.

IHXSLTA::SetTargetBandwidth

Sets the target bandwidth for rule subscription. This method should not be used unless you want to purposely override the bandwidth of the stream. Normally, the simulated live transfer agent (SLTA) sends data at whatever bandwidth the connection between the server and client can handle.

STDMETHOD(SetTargetBandwidth) ( 
    THIS, 
    UINT32 ulTargetBW 
) PURE;

ulTargetBW
    The target bandwidth, in bits per second.
**IHXSltaEvent**

Purpose: Sends events through a SLTA stream.

Implemented by: SLTA library

Used by: SLTA library users

Header file: hxslta.h

This interface sends event notifications and the event information through a simulated live stream.

*For More Information:* See the broadcast simulation chapter in *Helix Universal Server Administration Guide* and “Simulated Live Transfer Agent” in Volume 1, on page 101 in this guide.

The IHXSltaEvent interface contains the following methods:

- IHXSltaEvent::SetEvent
- IHXSltaEvent::SetRepeatedEvent

As with all COM interfaces, the IHXSltaEvent interface inherits the following IUnknown methods:

- IUnknown::AddRef
- IUnknown::QueryInterface
- IUnknown::Release

**IHXSltaEvent::SetEvent**

Sends an event to the SLTA stream.

If this method is called while actually streaming (that is, during a call to IHXiQSLTA::BeginTransmission), the event is sent immediately. If this method is called while streaming is not occurring (that is, before or between calls to IHXiQSLTA::BeginTransmission), the event is queued. All queued events are then sent at the beginning of the next IHXiQSLTA::BeginTransmission.

This method does not check the event code—the event is streamed no matter what it is. However, if the event code is not valid, the client ignores the event.

```cpp
STDMETHOD(SetEvent) ( 
  THIS_ 
  UINT16 nEventID, 
  const char* szEventText 
) PURE;
```

**nEventID**

The type of event. One of the following:

- HX_EVENT_AUTHOR
  
  The author of the presentation is being sent.

- HX_EVENT_BROWSER_OPEN_URL
  
  This flag is obsolete and should not be used in any new programming.

- HX_EVENT_COPYRIGHT
The copyright information for the presentation is being sent.

• HX_EVENT_CUSTOM_BEGIN
  This flag is obsolete and should not be used in any new programming.

• HX_EVENT_EMPTY
  This flag is obsolete and should not be used in any new programming.

• HX_EVENT_PROGRESS_MESSAGE
  This flag is obsolete and should not be used in any new programming.

• HX_EVENT_SERVER_ALERT
  This flag is obsolete and should not be used in any new programming.

• HX_EVENT_TEXT
  This flag is obsolete and should not be used in any new programming.

• HX_EVENT_TEXT_ANCHOR
  This flag is obsolete and should not be used in any new programming.

• HX_EVENT_TEXT_SIZE
  This flag is obsolete and should not be used in any new programming.

• HX_EVENT_TITLE
  The title information for the presentation is being sent.

• HX_EVENT_TOPIC
  This flag is obsolete and should not be used in any new programming.

szEventText
Contains any text for the event specified in nEventID. For example, if nEventID is set to HX_EVENT_TITLE, this parameter contains the title of the presentation. If nEventID is set to HX_EVENT_COPYRIGHT, this parameter contains the copyright information for the presentation.

IHXSLtaEvent::SetRepeatedEvent
Sets an event to be repeated at a specified frequency. This method is not currently supported and returns HXR_NOTIMPL.

STDMETHOD(SetRepeatedEvent) (  
  THIS_  
  UINT16 nEventID,  
  const char* szEventText,  
  UINT32 ulFrequency  
) PURE;

nEventID
The type of event. One of the following:

• HX_EVENT_AUTHOR
  The author of the presentation is being sent.
• HX_EVENT_BROWSER_OPEN_URL
  This flag is obsolete and should not be used in any new programming.

• HX_EVENT_COPYRIGHT
  The copyright information for the presentation is being sent.

• HX_EVENT_CUSTOM_BEGIN
  This flag is obsolete and should not be used in any new programming.

• HX_EVENT_EMPTY
  This flag is obsolete and should not be used in any new programming.

• HX_EVENT_PROGRESS_MESSAGE
  This flag is obsolete and should not be used in any new programming.

• HX_EVENT_SERVER_ALERT
  This flag is obsolete and should not be used in any new programming.

• HX_EVENT_TEXT
  This flag is obsolete and should not be used in any new programming.

• HX_EVENT_TEXT_ANCHOR
  This flag is obsolete and should not be used in any new programming.

• HX_EVENT_TEXT_SIZE
  This flag is obsolete and should not be used in any new programming.

• HX_EVENT_TITLE
  The title information for the presentation is being sent.

• HX_EVENT_TOPIC
  This flag is obsolete and should not be used in any new programming.

szEventText
Contains any text for the event specified in nEventID. For example, if nEventID is set to HX_EVENT_TITLE, this parameter contains the title of the presentation. If nEventID is set to HX_EVENT_COPYRIGHT, this parameter contains the copyright information for the presentation.

ulFrequency
The frequency, in milliseconds, that specifies when the event is to be repeated.
IHXSourceFinderObject

Purpose: Finds a stream by URL.
Implemented by: Server core
Used by: Server’s live packet sink plug-ins
Header file: hxsrc.h

This interface enables a sink interface to search for a raw packet source.

The IHXSourceFinderObject interface contains the following methods:

• IHXSourceFinderObject::Done
• IHXSourceFinderObject::Find
• IHXSourceFinderObject::Init

As with all Component Object Model (COM) interfaces, the IHXSourceFinderObject interface inherits the following IUnknown methods:

• IUnknown::AddRef
• IUnknown::QueryInterface
• IUnknown::Release

IHXSourceFinderObject::Done

Finished searching for the stream.

STDMETHOD(Done) ( 

    THIS
    ) PURE;

IHXSourceFinderObject::Find

Searches for the requested live stream source.

STDMETHOD(Find) ( 

    THIS_
    IHXRequest* pRequest
    ) PURE;

pRequest

Pointer to an IHXRequest interface that manages the source information.

IHXSourceFinderObject::Init

Sets a context for the source finder response.

STDMETHOD(Init) ( 

    THIS_
    IUnknown* pUnknown
    ) PURE;

pUnknown

Pointer to an instance of an IHXSourceFinderResponse interface.
**IHXSourceFinderResponse**

**Purpose:** Returns a raw packet source to a sink.

**Implemented by:** Server's live packet sink plug-in

**Used by:** Server core

**Header file:** hxsrc.h

This interface notifies the plug-in when the live source that it was requested to find has been located. The IHXSourceFinderObject interface then locates the live source and calls back to this interface when the find operation is complete.

The IHXSourceFinderResponse interface contains the following methods:

- **IHXSourceFinderResponse::FindDone**
- **IHXSourceFinderResponse::InitDone**

As with all Component Object Model (COM) interfaces, the IHXSourceFinderResponse interface inherits the following IUnknown methods:

- IUnknown::AddRef
- IUnknown::QueryInterface
- IUnknown::Release

**IHXSourceFinderResponse::FindDone**

Indicates the source finder has finished searching for the requested live stream source.

```
 STDMETHOD(FindDone) ( 
     THIS_ 
     HX_RESULT status, 
     IUnknown* pUnknown 
 ) PURE;
```

**status**

The status of the IHXSourceFinderObject::Find operation. A value of HXR_OK indicates that the operation has been completed successfully.

**pUnknown**

Pointer to an instance of an IHXRawSourceObject interface.

**IHXSourceFinderResponse::InitDone**

Indicates the status of the source finder interface initialization.

```
 STDMETHOD(InitDone) ( 
     THIS_ 
     HX_RESULT status 
 ) PURE;
```

**status**

The status of the IHXSourceFinderObject::Init operation. A value of HXR_OK indicates that the operation has been completed successfully.
IHXStatistics

Purpose: Directs the client core to update registry statistics for a specified object.

Implemented by: Client core

Used by: Top-level client

Header file: hxcomm.h

Through this interface, the top-level client directs the client core to update registry statistics for an object. The top-level client uses IHXRegistryID to get the object’s registry ID. It then uses IHXStatistics to pass this ID to the client core and tell the client to update the statistics.

The IHXStatistics interface contains the following methods:

• IHXStatistics::InitializeStatistics
• IHXStatistics::Update

As with all Component Object Model (COM) interfaces, the IHXStatistics interface inherits the following IUnknown methods:

• IUnknown::AddRef
• IUnknown::QueryInterface
• IUnknown::Release

IHXStatistics::InitializeStatistics

Passes the registry ID of the statistics to the caller.

STDMETHOD(InitializeStatistics) (
    THIS
    UINT32 ulRegistryID
) PURE;

ulRegistryID
The registry ID of the statistics.

IHXStatistics::Update

Notifies the client to update its statistics stored in the registry.

STDMETHOD(UpdateStatistics) (
    THIS
) PURE;
**IHXStatusMessage**

Purpose: Sets the status text in the top-level client.

Implemented by: Client core and top-level clients

Used by: Top-level clients and rendering plug-ins

Header file: hxwin.h

Rendering plug-ins use this interface to inform the system that some event has occurred (such as the mouse pointer passing over a hyperlink). The status text is displayed on the status bar of the client. In the case of a mouse pointer passing over a hyperlink, the URL to which you would hyper navigate to is displayed. If no status bar is being displayed, the URL will be placed in the site.

The IHXStatusMessage interface contains the IHXStatusMessage::SetStatus method.

As with all Component Object Model (COM) interfaces, the IHXStatusMessage interface inherits the following IUnknown methods:

- IUnknown::AddRef
- IUnknown::QueryInterface
- IUnknown::Release

**IHXStatusMessage::SetStatus**

Sets the status text.

```
STDMETHOD(SetStatus) ( 
    THIS_
    const char* pText 
) PURE;
```

*pText*

The text to be displayed in the top-level client status bar, or in the site if no status bar is available.
IHXStream

Purpose: Provides information about a stream; provides feedback to Helix on the quality of service.

Implemented by: Stream object (Helix architecture)

Used by: Rendering plug-ins and Helix client

Header file: hxcore.h

This interface to a stream object enables a rendering plug-in or the client to gather information about a stream, such as its stream number and MIME type. The interface also enables the renderer to give Helix feedback on the quality of service, informing it of the effect of lost packets on the presentation.


The IHXStream interface contains the following methods:

- IHXStream::GetHeader
- IHXStream::GetRenderer
- IHXStream::GetRendererCount
- IHXStream::GetSource
- IHXStream::GetStreamNumber
- IHXStream::GetStreamType
- IHXStream::ReportQualityOfService
- IHXStream::ReportRebufferStatus
- IHXStream::SetGranularity

As with all Component Object Model (COM) interfaces, the IHXStream interface inherits the following IUnknown methods:

- IUnknown::AddRef
- IUnknown::QueryInterface
- IUnknown::Release

IHXStream::GetHeader

Returns the header for this stream using the indicated IHXValues interface.

STDMETHOD_(IHXValues*, GetHeader) (THIS)

IHXStream::GetRenderer

Returns the indicated renderer instance (IHXRenderer) supported by this stream.
STDMETHOD(GetRenderer) (
   THIS_
   UINT16 nIndex,
   REF(IUnknown*) pUnknown
) PURE;

nIndex
The index of the renderer supported by this stream. Currently, this should always be 0.

pUnknown
Returns a pointer to a context from other interfaces can be queried.

**IHXStream::GetRendererCount**

Returns the current number of renderer instances supported by this stream instance. Currently, this always returns 1.

STDMETHOD_(UINT16, GetRendererCount) (
   THIS
) PURE;

**IHXStream::GetSource**

Gets the interface to the stream source of which this stream is a part.

STDMETHOD(GetSource) (
   THIS_
   REF(IHXStreamSource*) pSource
) PURE;

pSource
Returns a pointer to an IHXStreamSource interface that manages the stream source.

**IHXStream::GetStreamNumber**

Returns the stream number for this stream relative to the IHXStreamSource interface of which the stream is a part.

STDMETHOD_(UINT16,GetStreamNumber) (
   THIS
) PURE;

**IHXStream::GetStreamType**

Gets the MIME type for this stream.

STDMETHOD_(const char*,GetStreamType) (
   THIS
) PURE;

**Note:** The returned string is assumed to be valid for the life of the IHXStream from which it was returned.
**IHXStream::ReportQualityOfService**

Reports to the playback context that the quality of service for this stream has changed. Although the transport engine can determine lost packets and report these through the user interface, only the renderer of this stream can determine the “real” perceived damage associated with this loss.

```cpp
STDMETHOD(ReportQualityOfService) (
    THIS_,
    UINT8 unQuality
) PURE;
```

**unQuality**

The quality of service for this stream. This parameter is set on a scale of 0 to 100, where 100 is the best possible quality for this stream.

**Note:** The playback context can use this value to indicate loss in quality to the user interface. When the effects of a lost packet are eliminated the renderer should call this method with a `unQuality` of 100.

**IHXStream::ReportRebufferStatus**

**Note:** This method is currently not implemented.

Reports to the playback context that the available data has dropped to a critically low level, and that rebuffering should occur. The renderer should call back into this interface as it receives additional data packets to indicate the status of its rebuffering effort.

The values of `unNeeded` and `unAvailable` indicate the general status of the rebuffering effort. For example, if a renderer has “run dry” and needs five data packets to play smoothly again, it should call this method with `unNeeded` set to 5 and `unAvailable` set to 0. Then, as packets arrive, the renderer should call again with 5,1, then 5,2, and eventually 5,5.

```cpp
STDMETHOD(ReportRebufferStatus) (
    THIS_,
    UINT8 unNeeded,
    UINT8 unAvailable
) PURE;
```

**unNeeded**

The number of packets needed to render the presentation smoothly.

**unAvailable**

The number of packets currently available.

**IHXStream::SetGranularity**

Sets the required granularity for this stream. The actual granularity will be the lowest granularity of all streams. Although it is valid to call this method before the stream actually begins, it is best to call this method during `IHXRenderer::OnHeader`.

```cpp
```
STDMETHOD(SetGranularity) ( 
    THIS_,
    ULONG32 ulGranularity
) PURE;

ulGranularity
    The required granularity.
IHXStreamSource

Purpose: Provides information about a stream source.
Implemented by: Stream source object (Helix architecture)
Used by: Rendering plug-ins and Helix client
Header file: hxcore.h

This interface to a stream source object enables a rendering plug-in or the client to gather information about the stream source, such as its URL and the number of streams it supports. A component can get this interface through IHXStream::GetSource.


The IHXStreamSource interface contains the following methods:

• IHXStreamSource::GetPlayer
• IHXStreamSource::GetStream
• IHXStreamSource::GetStreamCount
• IHXStreamSource::GetURL
• IHXStreamSource::IsLive

As with all Component Object Model (COM) interfaces, the IHXStreamSource interface inherits the following IUnknown methods:

• IUnknown::AddRef
• IUnknown::QueryInterface
• IUnknown::Release

IHXStreamSource::GetPlayer

Gets the interface to the player of which the source is a part.

STDMETHOD(GetPlayer) (  
    THIS_  
    REF(IHXPlayer*) pPlayer  
) PURE;

pPlayer
Returns a pointer to the IHXPlayer interface.

IHXStreamSource::GetStream

Returns the indicated stream instance supported by this source.

STDMETHOD(GetStream) (  
    THIS_  
    UINT16 nIndex,  
    REF(IUnknown*) pUnknown  
) PURE;
nIndex
The index number of the stream instance.

pUnknown
Returns a pointer that identifies the instance of the stream (IHXStream).

IHXStreamSource::GetStreamCount

Returns the current number of stream instances supported by this source instance.

STDMETHOD_(UINT16, GetStreamCount) (THIS ) PURE;

IHXStreamSource::GetURL

Returns a pointer to the requested URL for the stream source. The returned string is assumed to be valid for the life of the IHXStreamSource from which it was returned.

STDMETHOD_(const char*, GetURL) (THIS ) PURE;

IHXStreamSource::IsLive

Determines whether the stream source is live. Returns TRUE if the stream source is live.

STDMETHOD_ (BOOL, IsLive) (THIS ) PURE;
IHXSyncFileFormatObject

Purpose: Provides methods of communicating with file format plug-ins.
Implemented by: File format plug-ins
Used by: Helix architecture
Header file: hxformt.h

This interface provides all of the methods and functionality of IHXFileFormatObject. However, IHXFileFormatObject uses the IHXFormatResponse interface to pass back information to the caller asynchronously. In contrast, this interface passes back the information in the parameters of its methods.

The IHXSsyncFileFormatObject interface contains the following methods:
- IHXSsyncFileFormatObject::Close
- IHXSsyncFileFormatObject::GetFileFormatInfo
- IHXSsyncFileFormatObject::GetFileHeader
- IHXSsyncFileFormatObject::GetPacket
- IHXSsyncFileFormatObject::GetStreamHeader
- IHXSsyncFileFormatObject::InitFileFormat
- IHXSsyncFileFormatObject::Seek

As with all Component Object Model (COM) interfaces, the IHXSyncFileFormatObject interface inherits the following IUnknown methods:
- IUnknown::AddRef
- IUnknown::QueryInterface
- IUnknown::Release

IHXSsyncFileFormatObject::Close

Manages any cleanup required before closing the plug-in. All references to objects should be released and memory deallocated. This method is called when the playback is finished or stopped.

STDMETHOD(Close) (THIS)

IHXSsyncFileFormatObject::GetFileFormatInfo

Returns crucial information required to associate this plug-in with a given MIME type. This information tells the core which file format plug-in to use for a particular URL. The method is called when the Helix Architecture (HX) core application is started.

STDMETHOD(GetFileFormatInfo) (THIS,
  REF(const char**) pFileMimeTypes,
  REF(const char**) pFileExtensions,
  REF(const char**) pFileOpenNames)

) PURE;
pFileMimeTypes
Returns a pointer to the file MIME types supported by this plug-in.

pFileExtensions
Returns a pointer to the file extensions supported by this plug-in.

pFileOpenNames
Returns a pointer to the file open names supported by this plug-in.

IHXSynchronousFileFormatObject::GetFileHeader
Returns the file header information. This method is called after the file has been initialized.

STDMETHOD(GetFileHeader) (
    THIS_
    REF(IHXValues*) pHeader
) PURE;

pHeader
Returns a pointer to an IHXValues interface that manages the file header information.

IHXSynchronousFileFormatObject::GetPacket
Returns the packet data for the current stream. This method is called each time the server or client core needs another packet.

STDMETHOD(GetPacket) (
    THIS_
    REF(IHXPacket*) pPacket
) PURE;

pPacket
Returns a pointer to an IHXPacket interface that manages the requested data packet.

IHXSynchronousFileFormatObject::GetStreamHeader
Returns the stream header information for a particular stream. This method is called (after the file header has been read) for each stream in the file format.

STDMETHOD(GetStreamHeader) (
    THIS_
    REF(IHXValues*) pStreamHeader,
    UINT16 unStreamNumber
) PURE;

pStreamHeader
Returns a pointer to an IHXValues interface that manages the specified stream header.

unStreamNumber
Number of the stream from which to get the stream header.

IHXSynchronousFileFormatObject::InitFileFormat
Initializes the file. This method is called whenever a URL associated with this plug-in is opened.
STDMETHOD(InitFileFormat) (  
    THIS_  
    IHXRequest* pRequest  
) PURE;

pRequest  
    Pointer to an IHXRequest interface that manages the request for file format initialization.

IHXSynchronousFileFormatObject::Seek

Places the file at a specified position.

STDMETHOD(Seek) (  
    THIS_  
    ULONG32 ulSeekTime  
) PURE;

ulSeekTime  
    The time, in milliseconds, from the beginning of the presentation to which to position the file.
IHXTCPResponse

Purpose: Returns the status of TCP network operations.
Implemented by: Any component
Used by: TCP socket object (Network Services)
Header file: hxengin.h

After a component creates a TCP socket object with IHXNetworkServices, it uses IHXTCPSocket to read from and write to that socket. The TCP socket object then uses IHXTCPResponse to notify the component of the success or failure of these operations, as well as pass it pointers to IHXBuffer objects of read data.


The IHXTCPResponse interface contains the following methods:
- IHXTCPResponse::Closed
- IHXTCPResponse::ConnectDone
- IHXTCPResponse::ReadDone
- IHXTCPResponse::WriteReady

As with all Component Object Model (COM) interfaces, the IHXTCPResponse interface inherits the following IUnknown methods:
- IUnknown::AddRef
- IUnknown::QueryInterface
- IUnknown::Release

IHXTCPResponse::Closed

Informs you that the TCP channel has been closed by the peer or closed due to error.

STDMETHOD(Closed) (THIS_

HX_RESULT status
) PURE;

status
The status of the operation. A value of HXR_OK indicates that the operation was closed successfully by the peer.

IHXTCPResponse::ConnectDone

Indicates the IHXTCPSocket::Connect operation has completed.

STDMETHOD(ConnectDone) (THIS_

HX_RESULT status
) PURE;
status
The status of the IHXTCP::Connect operation. A value of HXR_OK indicates that the operation has been completed successfully.

IHXTCPResponse::ReadDone
Indicates the IHXTCP::Read operation has completed.

STDMETHOD(ReadDone) (  
    THIS_  
    HX_RESULT status,  
    IHXBuffer* pBuffer  
) PURE;

status
The status of the IHXTCP::Read operation. A value of HXR_OK indicates that the operation has been completed successfully.

pBuffer
Pointer to an IHXBuffer interface that manages the data returned from the read operation.

IHXTCPResponse::WriteReady
Indicates the IHXTCP::WantWrite operation has completed.

STDMETHOD(WriteReady) (  
    THIS_  
    HX_RESULT status  
) PURE;

status
The status of the IHXTCP::WantWrite operation. A value of HXR_OK indicates that the operation has been completed successfully and that you can now write to the TCP channel.
**IHXTCPSocket**

**Purpose:** Performs TCP network operations.

**Implemented by:** TCP socket object (Network Services)

**Used by:** Any component

**Header file:** hxengin.h

Any component that needs to use a TCP socket for network communications can use this interface. The component first creates a TCP socket object with IHXNetworkServices. It then uses this interface to perform read and write operations. The TCP socket object notifies the component about the success or failure of operations through IHXTCPResponse.

**For More Information:** See “Using a TCP Socket” in Volume 1, on page 82. See IHXListenSocket in Volume 2, on page 206 and IHXUDPSocket in Volume 2, on page 443.

The IHXTCPSocket interface contains the following methods:

- IHXTCPSocket::Bind
- IHXTCPSocket::Connect
- IHXTCPSocket::GetForeignAddress
- IHXTCPSocket::GetForeignPort
- IHXTCPSocket::GetLocalAddress
- IHXTCPSocket::GetLocalPort
- IHXTCPSocket::Init
- IHXTCPSocket::Read
- IHXTCPSocket::SetResponse
- IHXTCPSocket::WantWrite
- IHXTCPSocket::Write

As with all Component Object Model (COM) interfaces, the IHXTCPSocket interface inherits the following IUnknown methods:

- IUnknown::AddRef
- IUnknown::QueryInterface
- IUnknown::Release

**IHXTCPSocket::Bind**

Binds the TCP socket to a local address and port. This method specifies the port and address in native byte order.

```c
STDMETHOD(Bind) ( 
    THIS_,
    UINT32 ulLocalAddr,
    UINT16 nPort
) PURE;
```

ulLocalAddr
The local address.
nPort
The number of the local port.

IHXTCP::Connect
Connects to the specified address.

STDMETHOD(Connect) (  
    THIS_  
    const char* pDestination,  
    UINT16 nPort  
) PURE;

pDestination
Pointer to the destination to which to connect. The destination to which this parameter points consists of either a host name in the form www.myserver.com or an IP address in the form nnn.nnn.nnn.nnn.

nPort
The port to which to connect.

IHXTCP::GetForeignAddress
Retrieves the address of the other end of the TCP socket.

STDMETHOD(GetForeignAddress) (  
    THIS_  
    REF(ULONG32) lAddress  
) PURE;

lAddress
Returns the address of the other end of the TCP socket. This address is returned in local host order.

IHXTCP::GetForeignPort
Retrieves the port of the other end of the TCP socket.

STDMETHOD(GetForeignPort) (  
    THIS_  
    REF(UINT16) port  
) PURE;

port
Returns the port of the other end of the TCP socket. This port is returned in local host order.

IHXTCP::GetLocalAddress
Retrieves the local address on which the socket is connected.

STDMETHOD(GetLocalAddress) (  
    THIS_  
    REF(ULONG32) lAddress  
) PURE;
IAddress
Returns the local address. This address is returned in local host order.

IHXTCPSocket::GetLocalPort
Retrieves the local port on which the socket is connected.

STDMETHOD(GetLocalPort) (
    THIS_
    REF(UINT16) port
) PURE;

port
Returns the local port. This port is returned in local host order.

IHXTCPSocket::Init
Initializes the TCP socket and identifies the response interface.

STDMETHOD(Init) (
    THIS_
    IHXTCPResponse* pTCPResponse
) PURE;

pTCPResponse
Pointer to an IHXTCPResponse interface that manages the responses to various IHXTCPSocket methods.

IHXTCPSocket::Read
Reads the specified number of bytes from the TCP source.

STDMETHOD(Read) (
    THIS_
    UINT16 Size
) PURE;

Size
The size, in bytes, of the data to read from the TCP source.

IHXTCPSocket::SetResponse
Identifies the response interface. This method is used if the calling component has inherited an existing, initialized TCP socket.

STDMETHOD(SetResponse) (
    THIS_
    IHXTCPResponse* pTCPResponse
) PURE;

pTCPResponse
Pointer to an IHXTCPResponse interface that manages responses to various IHXTCPSocket methods.
**IHXTCPSocket::WantWrite**

Indicates that the component wants to write a large amount of data (more that 500K, approximately) to the TCP socket. When the TCP channel is ready to be written to, the IHXTCPResponse::WriteReady method is called.

If you are only writing small amounts of data, you can just call IHXTCPsocket::Write (all the data not ready to be transmitted is buffered on your behalf).

```cpp
STDMETHOD(WantWrite) ( 
    THIS 
) PURE;
```

**IHXTCPSocket::Write**

Writes data to the TCP socket.

```cpp
STDMETHOD(Write) ( 
    THIS_ 
    IHXBuffer* pBuffer 
) PURE;
```

**pBuffer**

Pointer to an IHXBuffer interface that manages the data to be written.
IHXThreadSafeMethods

Purpose: Indicates which methods are thread-safe.
Implemented by: Server plug-ins
Used by: Server core
Header file: hxengine.h

This interface registers with the server core what methods of the plug-in are thread-safe.
The IHXThreadSafeMethods interface contains the IHXThreadSafeMethods::IsThreadSafe method.
As with all Component Object Model (COM) interfaces, the IHXThreadSafeMethods interface inherits the following IUnknown methods:
- IUnknown::AddRef
- IUnknown::QueryInterface
- IUnknown::Release

IHXThreadSafeMethods::IsThreadSafe

Indicates which methods are thread-safe. This method returns one or more of the following values:

- HX_THREADSAFE_METHOD_FF_GETPACKET
  The IHXFileFormatObject::GetPacket method is thread-safe.
- HX_THREADSAFE_METHOD_FS_READ
  The IHXFileObject::Read method is thread-safe.
- HX_THREADSAFE_METHOD_FSR_READDONE
  The IHXFileResponse::ReadDone method is thread-safe.
- HX_THREADSAFE_METHOD_CACHE_FILE
  All methods in the IHXCacheFile interface are thread-safe.
- HX_THREADSAFE_METHOD_CACHE_FILE_RESPONSE
  All methods in the IHXCacheFileResponse interface are thread-safe.
- HX_THREADSAFE_METHOD_CONVERT_HEADERS
  The IHXDataConvert::ConvertFileHeader and IHXDataConvert::ConvertStreamHeader interfaces are thread-safe.
- HX_THREADSAFE_METHOD_CONVERT_DATA
  The IHXDataConvert::ConvertData method is thread-safe.
- HX_THREADSAFE_METHOD_CONVERT_CTRL_BUFFER_READY
  The IHXDataConvert::ControlBufferReady method is thread-safe.

STDMETHOD_(UINT32, IsThreadSafe) (THIS ) PURE;
IHXThreadSafeScheduler

Purpose: Schedules thread-safe callbacks.
Implemented by: Scheduler (Helix architecture)
Used by: Any component
Header file: hxengin.h

This interface provides the user with a way of scheduling callbacks that will be executed at some time in the future. This interface is identical to IHXScheduler except the scheduler events are considered thread-safe.

The IHXThreadSafeScheduler interface contains the following methods:

• IHXThreadSafeScheduler::AbsoluteEnter
• IHXThreadSafeScheduler::RelativeEnter
• IHXThreadSafeScheduler::Remove

As with all Component Object Model (COM) interfaces, the IHXThreadSafeScheduler interface inherits the following IUnknown methods:

• IUnknown::AddRef
• IUnknown::QueryInterface
• IUnknown::Release

IHXThreadSafeScheduler::AbsoluteEnter

Schedules a callback to be executed at the specified time. Returns the handle to the callback.

STDMETHOD_(CallbackHandle,AbsoluteEnter) (THIS_ IHXCallback* pCallback, HXTimeval tVal ) PURE;

pCallback
Pointer to an IHXCallback interface to be executed.

tVal
An HXTimeval structure that contains the relevant time information.

IHXThreadSafeScheduler::RelativeEnter

Schedules a callback to be executed a specified time from now. Returns the handle to the callback. This method is less precise than IHXThreadSafeScheduler::AbsoluteEnter and should only be used when accurate timing is not critical.

STDMETHOD_(CallbackHandle,RelativeEnter) (THIS_ IHXCallback* pCallback, UINT32 ms ) PURE;
pCallback
Pointer to an IHXCallback interface to be executed.

ms
The number of milliseconds that must elapse before the callback is executed. A relative enter of 0 (zero) milliseconds is the same as an absolute enter of 0 milliseconds.

IHXThreadSafeScheduler::Remove
Removes a specified callback from the scheduler.

STDMETHOD(Remove) (THIS_
CallbackHandle Handle
) PURE;

Handle
The handle of the callback to be removed.
**IHXTrack**

Purpose: Sets timing properties for a media element.
Implemented by: Client core
Used by: Helix architecture
Header file: hxgroup.h

This interface provides control of a track belonging to a specific group, and is included for SMIL 2.0 support. Also included is support for sound level animation, the process of varying the audio levels between two specific points in a track.

The IHXTrack interface contains the following methods:

- IHXTrack::AddRepeat
- IHXTrack::Begin
- IHXTrack::BeginSoundLevelAnimation
- IHXTrack::EndSoundLevelAnimation
- IHXTrack::GetSoundLevel
- IHXTrack::GetSource
- IHXTrack::GetTrackProperties
- IHXTrack::Pause
- IHXTrack::Seek
- IHXTrack::SetSoundLevel
- IHXTrack::Stop

As with all COM interfaces, the IHXTrack interface inherits the following IUnknown methods:

- IUnknown::AddRef
- IUnknown::QueryInterface
- IUnknown::Release

**IHXTrack::AddRepeat**

Repeats a track.

STDMETHOD(AddRepeat) (
    THIS_
    IHXValues* pValues
) PURE;

pValues
    Pointer to an IHXValues interface that manages the track to be repeated.

**IHXTrack::Begin**

Starts the track.

STDMETHOD(Begin) (
    THIS
) PURE;
**IHXTrack::BeginSoundLevelAnimation**

Indicates the volume at the start of sound level animation.

```cpp
STDMETHOD(BeginSoundLevelAnimation) ( 
    THIS_ 
    UINT16 uSoundLevelBeginWith 
) PURE;
```

`uSoundLevelBeginWith`

The volume, in percent, at which to begin the sound level animation. This parameter can be set to a value between 0 and 100.

**IHXTrack::EndSoundLevelAnimation**

Indicates the volume at the end of the sound level animation.

```cpp
STDMETHOD(EndSoundLevelAnimation) ( 
    THIS_ 
    UINT16 uSoundLevelEndWith 
) PURE;
```

`uSoundLevelEndWith`

The volume, in percent, at which to end the sound level animation. This parameter can be set to a value between 0 and 100.

**IHXTrack::GetSoundLevel**

Gets the audio level of the current track. Returns the audio level as a percentage of the total available volume (a value between 0 and 100).

```cpp
STDMETHOD_(UINT16, GetSoundLevel) ( 
    THIS 
) PURE;
```

**IHXTrack::GetSource**

Gets the interface to the stream source supported by this player.

```cpp
STDMETHOD(GetSource) ( 
    THIS_ 
    REF(IHXStreamSource*) pStreamSource 
) PURE;
```

`pStreamSource`

Returns a pointer to an IHXStreamSource interface that manages the stream source.

**IHXTrack::GetTrackProperties**

Retrieves the properties of the track.

```cpp
STDMETHOD(GetTrackProperties) ( 
    THIS_ 
    REF(IHXValues*) pValues 
) PURE;
```
pValues
    Returns a pointer to an IHXValues interface that manages the track's properties.

IHXTrack::Pause
    Pauses the track.
    STDMETHOD(Pause) (
        THIS
    ) PURE;

IHXTrack::Seek
    Seeks to the specified time in the track.
    STDMETHOD(Seek) (
        THIS
        UINT32 ulSeekTime
    ) PURE;

    ulSeekTime
    The time, in milliseconds, to which to seek in the track.

IHXTrack::SetSoundLevel
    Sets the audio level of this track.
    STDMETHOD(SetSoundLevel) (
        THIS
        UINT16 uSoundLevel
    ) PURE;

    uSoundLevel
    The sound level, in percent, of the current track. This parameter can be set to a value between 0 and 100.

IHXTrack::Stop
    Stops the track.
    STDMETHOD(Stop) (
        THIS
    ) PURE;
IHXTrackSink

Purpose: Monitors a track in a presentation.
Implemented by: Helix architecture
Used by: Client core
Header file: hxgroup.h

This interface handles responses and supplies information to the IHXTrack interface methods.
The IHXTrackSink interface contains the following methods:
• IHXTrackSink::BeginDone
• IHXTrackSink::OnSoundLevelAnimation
• IHXTrackSink::PauseDone
• IHXTrackSink::SeekDone
• IHXTrackSink::StopDone

As with all COM interfaces, the IHXTrackSink interface inherits the following IUnknown methods:
• IUnknown::AddRef
• IUnknown::QueryInterface
• IUnknown::Release

IHXTrackSink::BeginDone
Indicates the IHXTrack::Begin method has completed.

STDMETHOD(BeginDone) (    
    THIS_    
    UINT16 uGroupIndex,    
    UINT16 uTrackIndex
) PURE;

uGroupIndex
Index of the group to which the track belongs.

uTrackIndex
Index of the track that was started.

IHXTrackSink::OnSoundLevelAnimation
Indicates sound level animation is in progress.

STDMETHOD(OnSoundLevelAnimation) (    
    THIS_    
    UINT16 uGroupIndex,    
    UINT16 uTrackIndex,    
    UINT32 ulSoundLevelAnimationTime
) PURE;

uGroupIndex
Index of the group to which the track belongs.
uTrackIndex
Index of the track on which sound level animation is being applied.

ulSoundLevelAnimationTime
The time, in milliseconds, between the beginning of the sound level animation and the end of the sound level animation.

**IHXTrackSink::PauseDone**

Indicates the IHXTrack::Pause method has completed.

```cpp
STDMETHOD(PauseDone) (
    THIS_
    UINT16 uGroupIndex,
    UINT16 uTrackIndex
) PURE;
```

**uGroupIndex**
Index of the group to which the track belongs.

**uTrackIndex**
Index of the track that was paused.

**IHXTrackSink::SeekDone**

Indicates the IHXTrack::Seek method has completed.

```cpp
STDMETHOD(SeekDone) (
    THIS_
    UINT16 uGroupIndex,
    UINT16 uTrackIndex,
    UINT32 ulSeekTime
) PURE;
```

**uGroupIndex**
Index of the group to which the track belongs.

**uTrackIndex**
Index of the track on which the seek occurred.

**ulSeekTime**
The time, in milliseconds, to which the seek occurred.

**IHXTrackSink::StopDone**

Indicates the IHXTrack::Stop method has completed.

```cpp
STDMETHOD(StopDone) (
    THIS_
    UINT16 uGroupIndex,
    UINT16 uTrackIndex
) PURE;
```

**uGroupIndex**
Index of the group to which the track belongs.
uTrackIndex
Index of the track that was stopped.
IHXTransportControl

Purpose: Sets the transport type.
Implemented by: Remote broadcast library
Used by: Remote broadcast application
Header file: hxencod.h

Sets the transport type.

The IHXTransportControl interface contains the IHXTransportControl::SetTransportType method.

As with all Component Object Model (COM) interfaces, the IHXTransportControl interface inherits the following IUnknown methods:
• IUnknown::AddRef
• IUnknown::QueryInterface
• IUnknown::Release

IHXTransportControl::SetTransportType

Sets the transport type.

STDMETHOD(SetTransportType) ( const char* pTransportType ) PURE;

pTransportType
The transport type. The two encoder transport types currently supported are:
• x-hx-tng/udp
• x-hx-tng/tcp
**IHXUDP MulticastInit**

**Purpose:** Sets the TTL for outgoing multicast UDP packets.

**Implemented by:** Client core

**Used by:** Plug-ins

**Header file:** hxengin.h

Plug-ins using IHXUDPsocket that must send multicast data would use IHXUDP MulticastInit.

The IHXUDP MulticastInit interface contains the IHXUDP MulticastInit::InitMulticast method.

As with all Component Object Model (COM) interfaces, the IHXUDP MulticastInit interface inherits the following IUnknown methods:

- IUnknown::AddRef
- IUnknown::QueryInterface
- IUnknown::Release

### IHXUDP MulticastInit::InitMulticast

Sets the TTL (time to live) for the UDP socket so that you can use it as a multicast socket.

```cpp
STDMETHOD(InitMulticast) (
    THIS_
    UINT8 chTTL
) PURE;
```

`chTTL`

The number of routers across which to send packets.
IHXUDPResponse

Purpose: Returns the status of UDP network operations.
Implemented by: Any component
Used by: UDP socket object (Network Services)
Header file: hxengin.h

After a component creates a UDP socket object with IHXNetworkServices, it uses IHXUDPSocket to read from and write to that socket. The UDP socket object then uses IHXUDPResponse to notify the component of the success or failure of the read operation.


The IHXUDPResponse interface contains the IHXUDPResponse::ReadDone method.

As with all Component Object Model (COM) interfaces, the IHXUDPResponse interface inherits the following IUnknown methods:
- IUnknown::AddRef
- IUnknown::QueryInterface
- IUnknown::Release

IHXUDPResponse::ReadDone

Indicates the IHXUDPSocket::Read method has completed.

STDMETHOD(ReadDone) (  
    THIS_  
    HX_RESULT status,  
    IHXBuffer* pBuffer,  
    ULONG32 ulAddr,  
    UINT16 nPort  
) PURE;

status
The status of the IHXUDPSocket::Read operation. A value of HXR_OK indicates that the operation has been completed successfully.

pBuffer
Pointer to an IHXBuffer interface that manages the data that was read.

ulAddr
Indicates the remote address, in local host order.

nPort
Indicates the remote port, in local host order.
**IHXUDPSocket**

- **Purpose:** Performs UDP network operations.
- **Implemented by:** UDP socket object (Network Services)
- **Used by:** Any component
- **Header file:** hxengin.h

Any component that needs to use a UDP socket for network communications can use this interface. The component first creates a UDP socket object with IHXNetworkServices. It then uses IHXUDPSocket to perform read and write operations. The UDP socket object notifies the component about the success or failure of operations through IHXUDPResponse.

**For More Information:** See “Using a UDP Socket” in Volume 1, on page 83. See IHXTCPSocket in Volume 2, on page 427.

The IHXUDPSocket interface contains the following methods:

- IHXUDPSocket::Bind
- IHXUDPSocket::GetLocalPort
- IHXUDPSocket::Init
- IHXUDPSocket::JoinMulticastGroup
- IHXUDPSocket::LeaveMulticastGroup
- IHXUDPSocket::Read
- IHXUDPSocket::Write
- IHXUDPSocket::WriteTo

As with all Component Object Model (COM) interfaces, the IHXUDPSocket interface inherits the following IUnknown methods:

- IUnknown::AddRef
- IUnknown::QueryInterface
- IUnknown::Release

**IHXUDPSocket::Bind**

Binds the UDP socket to a local address and port.

```c
STDMETHOD(Bind) (  
    THIS_  
    UINT32 ulLocalAddr,  
    UINT16 nPort  
) PURE;
```

- **ulLocalAddr**
  - The local address, in local host order.

- **nPort**
  - The number of the local port, in local host order.
IHXUDPSocket::GetLocalPort

Retrieves the local port.

```
HRESULT GetLocalPort( 
    UINT16* port 
) PURE;
```

*port*

Returns the locally bound port, in local host order.

IHXUDPSocket::Init

Initializes the UDP socket and identifies the response interface.

```
HRESULT Init( 
    ULONG32 ulAddr, 
    UINT16 nPort, 
    IHXUDPResponse* pUDPResponse 
) PURE;
```

*ulAddr*

The remote host address, in local host order.

*nPort*

The remote host port, in local host order.

*pUDPResponse*

Pointer to an IHXUDPResponse interface that manages responses to various IHXUDPSocket methods.

IHXUDPSocket::JoinMulticastGroup

Connects through Helix’s UDP multicast.

```
HRESULT JoinMulticastGroup( 
    ULONG32 ulMulticastAddr, 
    ULONG32 ulInterfaceAddr 
) PURE;
```

*ulMulticastAddr*

The multicast address (the address to which this method is sending).

*ulInterfaceAddr*

The interface address (the address from which this method is sending).

IHXUDPSocket::LeaveMulticastGroup

Leaves Helix's UDP multicast.
STDMETHOD(LeaveMulticastGroup) (  
    THIS_,  
    ULONG32 ulMulticastAddr,  
    ULONG32 ulInterfaceAddr  
) PURE;

ulMulticastAddr  
The multicast address (the address to which this method was sending).

ulInterfaceAddr  
The interface address (the address from which this method was sending).

**IHXUDPSocket::Read**

Reads a specified number of bytes from the UDP source.

STDMETHOD(Read) (  
    THIS_,  
    UINT16 Size  
) PURE;

Size  
The size, in bytes, of the data to read from the UDP source.

**IHXUDPSocket::Write**

Writes data to the UDP socket.

STDMETHOD(Write) (  
    THIS_,  
    IHXBuffer* pBuffer  
) PURE;

pBuffer  
Pointer to an IHXBuffer interface that manages the data to write to the UDP socket.

**IHXUDPSocket::WriteTo**

Writes data to the specified UDP socket.

STDMETHOD(WriteTo) (  
    THIS_,  
    ULONG32 ulAddr,  
    UINT16 nPort,  
    IHXBuffer* pBuffer  
) PURE;

ulAddr  
The remote host address, in local host order.

nPort  
The remote host port, in local host order.

pBuffer  
Pointer to an IHXBuffer interface that manages the data to write to the specified UDP socket.
IHXUpdateProperties

Purpose: Updates any offset-related properties.
Implemented by: Renderer plug-ins
Used by: Client core
Header file: hxcore.h

This interface is implemented by renderer plug-ins to receive offset-related property changes from the client core.

The IHXUpdateProperties interface contains the IHXUpdateProperties::UpdatePacketTimeOffset method.

As with all Component Object Model (COM) interfaces, the IHXUpdateProperties interface inherits the following IUnknown methods:
  • IUnknown::AddRef
  • IUnknown::QueryInterface
  • IUnknown::Release

IHXUpdateProperties::UpdatePacketTimeOffset

Receives the change in the packet time stamp offset during playback.

STDMETHOD(UpdatePacketTimeOffset) (  
   THIS_  
   INT32 lTimeOffset  
) PURE;

lTimeOffset

The time, in milliseconds, to which the time stamp of cached packets are offset.
**IHXUpgradeCollection**

Purpose: Enables collection of upgrade components.

Implemented by: Client core

Used by: Renderer plug-ins, file format plug-in

Header file: hxupgrd.h

This interface reports missing components for participation in AutoUpdate. A renderer or file format plug-in can query IHXUpgradeCollection from the core context and then use IHXUpgradeCollection::Add to report the missing component.

The IHXUpgradeCollection interface contains the following methods:

- IHXUpgradeCollection::Add
- IHXUpgradeCollection::GetAt
- IHXUpgradeCollection::GetCount
- IHXUpgradeCollection::Remove
- IHXUpgradeCollection::RemoveAll

As with all Component Object Model (COM) interfaces, the IHXUpgradeCollection interface inherits the following IUnknown methods:

- IUnknown::AddRef
- IUnknown::QueryInterface
- IUnknown::Release

**IHXUpgradeCollection::Add**

Adds the specified upgrade information to the collection. This method returns the number of the component just added to the upgrade list.

```cpp
STDMETHOD_(UINT32, Add) (  
    THIS_  
    HXUpgradeType upgradeType,  
    IHXBuffer* pPluginId,  
    UINT32 majorVersion,  
    UINT32 minorVersion  
) PURE;
```

**upgradeType**

An HXUpgradeType enumerator that specifies the type of upgrade. The type can be required, recommended, or optional.

**pPluginID**

Pointer to an IHXBuffer interface that manages the plug-in's ID.

**majorVersion**

If pPluginID is the name of an installed component, this parameter identifies the lowest major version of the required component.
minorVersion
If pPluginID is the name of an installed component, this parameter identifies the lowest minor
version of the required component.

IHXUpgradeCollection::GetAt
Retrieves the specified item’s upgrade information.

STDMETHOD(GetAt) (  
      THIS_  
      UINT32 index,  
      REF(HXUpgradeType) upgradeType,  
      IHXBuffer* pPluginId,  
      REF(UINT32) majorVersion,  
      REF(UINT32) minorVersion  
    ) PURE;

index
The item from which to get the upgrade information.

upgradeType
Returns the type of upgrade. This can be required, recommended, or optional.

pPluginID
Pointer to an IHXBuffer interface that manages the plug-in’s ID.

majorVersion
Returns the major version of the plug-in.

minorVersion
Returns the minor version of the plug-in.

IHXUpgradeCollection::GetCount
Gets the count of the collection.

STDMETHOD_(UINT32, GetCount) (  
      THIS  
    ) PURE;

IHXUpgradeCollection::Remove
Remove the specified item from the collection.

STDMETHOD(Remove) (  
      THIS_  
      UINT32 index  
    ) PURE;

index
The item to remove.

IHXUpgradeCollection::RemoveAll
Removes all items from the collection.
STDMETHOD(RemoveAll) ( THIS ) PURE;
Before starting playback, the client core checks if its IHXUpgradeCollection implementation has any components registered by plug-ins. If so, it passes the component collection to IHXUpgradeHandler::HasComponents, which checks if the requested components are already present. All components that are present are removed from IHXUpgradeCollection. After the IHXUpgradeHandler::HasComponents call, IHXUpgradeCollection still contains the components that need to be upgraded. IHXUpgradeHandler::RequestUpgrade is then used to start the AutoUpdate process.

The IHXUpgradeHandler interface contains the following methods:

- IHXUpgradeHandler::HasComponents
- IHXUpgradeHandler::RequestUpgrade

As with all Component Object Model (COM) interfaces, the IHXUpgradeHandler interface inherits the following IUnknown methods:

- IUnknown::AddRef
- IUnknown::QueryInterface
- IUnknown::Release

### IHXUpgradeHandler::HasComponents

Checks if required components are present on the system.

```c
STDMETHOD(HasComponents) ( 
    THIS
    IHXUpgradeCollection* pComponents
) PURE;
```

**pComponents**

Pointer to an IHXUpgradeCollection interface that manages the required upgrade components. If this method returns HXR_OK, all the components are already available, and no components are required. If this method returns HXR_FAILED, some components are missing and this interface contains only those components that need to be upgraded.

### IHXUpgradeHandler::RequestUpgrade

Asks if the user wants to do an upgrade and, if so, starts the upgrade.

```c
STDMETHOD(RequestUpgrade) ( 
    THIS
    IHXUpgradeCollection* pComponents,
    BOOL bBlocking
) PURE;
```
pComponents
   Pointer to an IHXUpgradeCollection interface that manages the components that need to be upgraded.

bBlocking
   If TRUE, this is a synchronous call. If FALSE, this is an asynchronous call.
IHXUserContext

Purpose: Provides access to information about the currently authenticated user.
Implemented by: Server core
Used by: Server plug-in
Header file: hxauthn.h

You can use this interface to find out to which Windows NT group an authenticated user belongs. This interface is only used in NT LAN Manager (NTLM) authentication.

The IHXUserContext interface contains the IHXUserContext::IsMemberOf method.

As with all Component Object Model (COM) interfaces, the IHXUserContext interface inherits the following IUnknown methods:
  • IUnknown::AddRef
  • IUnknown::QueryInterface
  • IUnknown::Release

IHXUserContext::IsMemberOf

Determines whether the authenticated user is a member of the specified group.

STDMETHOD(IsMemberOf) (THIS_ 
  IHXBuffer* pBufferGroupID
) PURE;

pBufferGroupID
  Pointer to an IHXBuffer interface that manages the group ID.
**IHXUserImpersonation**

Purpose: Enables the server to impersonate the currently authenticated user.

Implemented by: Server core

Used by: Server plug-in

Header file: hxauthn.h

With this interface, you can access files in Windows NT as if you were the currently authorized user.

The IHXUserImpersonation interface contains the following methods:

- IHXUserImpersonation::Start
- IHXUserImpersonation::Stop

As with all Component Object Model (COM) interfaces, the IHXUserImpersonation interface inherits the following IUnknown methods:

- IUnknown::AddRef
- IUnknown::QueryInterface
- IUnknown::Release

**IHXUserImpersonation::Start**

Begins impersonating the authenticated user.

```c
STDMETHOD(Start) ( 
    THIS 
) PURE;
```

**IHXUserImpersonation::Stop**

Stops impersonating the authenticated user.

```c
STDMETHOD(Stop) ( 
    THIS 
) PURE;
```
IHXUserProperties

Purpose: Provides access to properties of the currently authenticated user.
Implemented by: Server core
Used by: Server plug-ins
Header file: hxauthn.h

This interface gets information about the currently authenticated user, specifically the user name and the realm or authorization name.

The IHXUserProperties interface contains the following methods:
  • IHXUserProperties::GetAuthorityName
  • IHXUserProperties::GetPrincipalID

As with all Component Object Model (COM) interfaces, the IHXUserProperties interface inherits the following IUnknown methods:
  • IUnknown::AddRef
  • IUnknown::QueryInterface
  • IUnknown::Release

IHXUserProperties::GetAuthorityName

Determines who authorized the authenticated user (the realm, domain name, or other).

STDMETHOD(GetAuthorityName) (THIS_ REF(IHXBuffer*) pBufferAuthorityName ) PURE;

pBufferAuthorityName
Returns a pointer to an IHXBuffer interface that manages the name of the authorizer of the currently authenticated user.

IHXUserProperties::GetPrincipalID

Determine the principal ID of the authenticated user.

STDMETHOD(GetPrincipalID) (THIS_ REF(IHXBuffer*) pBufferPrincipalID ) PURE;

pBufferPrincipalID
Returns a pointer to an IHXBuffer interface that manages the principal ID.
IHXValues

Purpose: Stores name/value pairs.
Implemented by: Values object (Helix architecture)
Used by: Any component
Header file: ihxpckts.h

Like IHXBuffer, the IHXValues interface is widely used in Helix. It enables Helix components to store general lists that pair names with values, and is commonly used to hold stream initialization information. Each value in an IHXValues name/value pair is an unsigned long, a pointer to a buffer of arbitrary data, or a pointer to a buffer of NULL-terminated C-string data. The preferred implementation is to use IHXCommonClassFactory to create the values object.

For More Information: See “Using IHXValues to Create Indexed Lists” in Volume 1, on page 32.

The IHXValues interface contains the following methods:
- IHXValues::GetFirstPropertyBuffer
- IHXValues::GetFirstPropertyCString
- IHXValues::GetFirstPropertyULONG32
- IHXValues::GetNextPropertyBuffer
- IHXValues::GetNextPropertyCString
- IHXValues::GetNextPropertyULONG32
- IHXValues::GetPropertyBuffer
- IHXValues::GetPropertyCString
- IHXValues::GetPropertyULONG32
- IHXValues::SetPropertyBuffer
- IHXValues::SetPropertyCString
- IHXValues::SetPropertyULONG32

As with all Component Object Model (COM) interfaces, the IHXValues interface inherits the following IUnknown methods:
- IUnknown::AddRef
- IUnknown::QueryInterface
- IUnknown::Release

IHXValues::GetFirstPropertyBuffer

Retrieves the first buffer property name and value in an existing list of names and values.

STDMETHOD(GetFirstPropertyBuffer) (THIS_
    REF(const char*) pPropertyName,
    REF(IHXBuffer*) pPropertyValue
) PURE;
pPropertyName
Returns a pointer to the name of the property at the beginning of the list.

pPropertyValue
Returns a pointer to an IHXBuffer interface that manages the buffer value at the beginning of the list.

**IHXValues::GetFirstPropertyCString**

Retrieves the first CString property name and value in an existing list of names and values.

```cpp
STDMETHOD(GetFirstPropertyCString) (
    THIS_
    REF(const char*) pPropertyName,
    REF(IHXBuffer*) pPropertyValue
) PURE;
```

pPropertyName
Returns a pointer to the name of the property at the beginning of the list.

pPropertyValue
Returns a pointer to an IHXBuffer interface that manages the CString value at the beginning of the list.

**IHXValues::GetFirstPropertyULONG32**

Retrieves the first 32-bit unsigned long property name and value in an existing list of names and values.

```cpp
STDMETHOD(GetFirstPropertyULONG32) (
    THIS_
    REF(const char*) pPropertyName,
    REF(ULONG32) uPropertyValue
) PURE;
```

pPropertyName
Returns a pointer to the name of the property at the beginning of the list.

uPropertyValue
Returns the 32-bit unsigned long value at the beginning of the list.

**IHXValues::GetNextPropertyBuffer**

Retrieves the next buffer property name and value in an existing list of names and values.

```cpp
STDMETHOD(GetNextPropertyBuffer) (
    THIS_
    REF(const char*) pPropertyName,
    REF(IHXBuffer*) pPropertyValue
) PURE;
```

pPropertyName
Returns a pointer to the name of the property that is next in the list.
pPropertyValue
Returns a pointer to an IHXBuffer interface that manages the next buffer value in the list.

IHXValues::GetNextPropertyCString
Retrieves the next CString property name and value in an existing list of names and values.
STDMETHOD(GetNextPropertyCString) (  
    THIS_,
    REF(const char*) pPropertyName,
    REF(IHXBuffer*) pPropertyValue
) PURE;

pPropertyName
Returns a pointer to the name of the property that is next in the list.

pPropertyValue
Returns a pointer to an IHXBuffer interface that manages the next CString value in the list.

IHXValues::GetNextPropertyULONG32
Retrieves the next 32-bit unsigned long property name and value in an existing list of names and values.
STDMETHOD(GetNextPropertyULONG32) (  
    THIS_,
    REF(const char*) pPropertyName,
    REF(ULONG32) uPropertyValue
) PURE;

pPropertyName
Returns a pointer to the name of the property that is next in the list.

uPropertyValue
Returns the 32-bit unsigned long value that is next in the list.

IHXValues::GetPropertyBuffer
Retrieves the buffer value specified by the given property name.
STDMETHOD(GetPropertyBuffer) (  
    THIS_,
    const char* pPropertyName,
    REF(IHXBuffer*) pPropertyValue
) PURE;

pPropertyName
Pointer to the name of the property.

pPropertyValue
Returns a pointer to an IHXBuffer interface that manages the buffer value.

IHXValues::GetPropertyCString
Retrieves the CString value specified by the given property name.
STDMETHOD(GetPropertyCString) (  
    THIS_,  
    const char* pPropertyName,  
    REF(IHXBuffer*) pPropertyValue  
) PURE;

pPropertyName  
    Pointer to the name of the property.

pPropertyValue  
    Returns a pointer to an IHXBuffer interface that manages the CString value.

IHXValues::GetPropertyULONG32

Retrieves the 32-bit unsigned long value specified by the given property name.

STDMETHOD(GetPropertyULONG32) (  
    THIS_,  
    const char* pPropertyName,  
    REF(ULONG32) uPropertyName  
) PURE;

pPropertyName  
    Pointer to the name of the property.

uPropertyName  
    Returns the 32-bit unsigned long value.

IHXValues::setPropertyBuffer

Sets a value in a named buffer.

STDMETHOD(SetPropertyBuffer) (  
    THIS_,  
    const char* pPropertyName,  
    IHXBuffer* pPropertyValue  
) PURE;

pPropertyName  
    Pointer to the name assigned to this property.

pPropertyValue  
    Pointer to an IHXBuffer interface that manages this property value.

IHXValues::setPropertyCString

Sets a value in a named CString.

STDMETHOD(SetPropertyCString) (  
    THIS_,  
    const char* pPropertyName,  
    IHXBuffer* pPropertyValue  
) PURE;
pPropertyName
    Pointer to the name assigned to this property.

pPropertyValue
    Pointer to an IHXBuffer interface that manages this property value.

IHXValues::SetPropertyULONG32

Sets a value in a named 32-bit unsigned long.
STDMETHOD(SetPropertyULONG32) (  
    THIS_  
    const char* pPropertyName,  
    ULONG32 uPropertyValue  
) PURE;

pPropertyName
    Pointer to the name assigned to this property.

uPropertyValue
    The property value assigned to this 32-bit unsigned long.
IHXValuesRemove

Purpose: Removes name/value pairs.
Implemented by: Values object (Helix architecture)
Used by: Any component
Header file: ihxpckts.h

The IHXValuesRemove interface augments the IHXValues interface. A class that supports the IHXValues interface should also support IHXValuesRemove if the class is to allow users to remove values from the collection.

The IHXValuesRemove interface contains the following methods:
- IHXValuesRemove::Remove
- IHXValuesRemove::RemoveBuffer
- IHXValuesRemove::RemoveCString
- IHXValuesRemove::RemoveULONG32

As with all Component Object Model (COM) interfaces, the IHXValuesRemove interface inherits the following IUnknown methods:
- IUnknown::AddRef
- IUnknown::QueryInterface
- IUnknown::Release

IHXValuesRemove::Remove

Removes all items matching the item you specify. Use this method if you are uncertain of the data type. If you know what data type the item was saved as, use the specific method.

STDMETHOD(Remove) (THIS_ const char* pKey) PURE;

pKey
The item that is to be removed.

IHXValuesRemove::RemoveBuffer

Removes all buffer items that you specify.

STDMETHOD( RemoveBuffer ) ( THIS_ const char* pKey ) PURE;

pKey
The buffer item that is to be removed.
**IHXValuesRemove::RemoveCString**

Removes all CString items that you specify.

```cpp
STDMETHOD(RemoveCString) ( 
    THIS_ 
    const char* pKey 
) PURE;
```

**pKey**

The CString item that is to be removed.

---

**IHXValuesRemove::RemoveULONG32**

Removes all ULONG32 items that you specify.

```cpp
STDMETHOD(RemoveULONG32) ( 
    THIS_ 
    const char* pKey 
) PURE;
```

**pKey**

The ULONG item that is to be removed.
IHXVideoHook

**Note:** This interface is not currently supported.

**Purpose:** Modifies the video output of a video hook sink.

**Implemented by:** Custom renderer plug-ins

**Used by:** Video hook sinks

**Header file:** hxvsurf.h

This interface, which is implemented in a custom renderer plug-in (the video hook), is used along with IHXVideoHookSink on the client renderer (the video hook sink) to display modified video data on the video surface. To modify the video data, you must create a video hook to receive a custom datatype you have generated (this datatype transmits any modification data you may need on the client side). RealVideo data is then supplied in parallel with your own from the server (for example, as two separate files placed between `<par></par>` tags in a SMIL file). The client core receives both sources from the server, uses the built-in renderer to decompress the RealVideo source, and passes the other source through to your video hook.

After your video hook registers with the video hook sink, the video hook sink calls the methods in this interface to supply the video hook with the video frames that result from decompressing the RealVideo source so that you can modify them. When using these interfaces, the video hook need not display the video frame directly on the video surface. Instead, after the video data has been modified, the video hook passes the data back to the video hook sink, which then displays the modified video frame as its own. Any number of video hooks with their own parallel data can be registered with the video hook sink at the same time.

The IHXVideoHook interface contains the following methods:

- IHXVideoHook::GetProperties
- IHXVideoHook::HandleHookEvent
- IHXVideoHook::HookAddedNotification
- IHXVideoHook::OnVideoFrame

As with all COM interfaces, the IHXVideoHook interface inherits the following IUnknown methods:

- IUnknown::AddRef
- IUnknown::QueryInterface
- IUnknown::Release

**IHXVideoHook::GetProperties**

Provides user-level information about the video hook.

```cpp
STDMETHOD(GetProperties) ( 
    THIS_ 
    REF(IHXValues*) properties 
) PURE;
```
properties

Returns a pointer to an IHXValues interface that manages the property bindings. Suggested property bindings are:

• Description
  A CString describing the plug-in functionality.
• Copyright
  A CString copyright notice.
• MoreInfo
  A CString URL for obtaining more information.
• Version
  A ULONG32 version number.
• InputFormat
  An IHXBuffer interface managing an input HXBitmapInfoHeader structure flattened as a UCHAR array.
• OutputFormat
  An IHXBuffer interface managing an output HXBitmapInfoHeader structure flattened as a UCHAR array.

You can include other property bindings for specific hook-to-hook communications.

IHXVideoHook::HandleHookEvent

Handles events, such as mouse clicks, that were received by the video hook sink and are then passed on to registered video hooks. Video hooks can use or ignore this information, but setting the handled member of the passed HXxEvent structure is not allowed.

STDMETHOD(HandleHookEvent) (THIS_ HXxEvent* pEvent) PURE;

pEvent
  Pointer to an HXxEvent structure that contains information about the event that occurred.

IHXVideoHook::HookAddedNotification

Provides notification to the video hook that the video hook sink has registered another video hook. This allows registered video hooks to redo their checking of the other video hooks whenever new ones are added.

STDMETHOD(HookAddedNotification) (THIS_ IHXVideoHook* pNewHook) PURE;

pNextVideoHook
  Pointer to an IHXVideoHook interface that manages the new video hook that was added.
**IHXVideoHook::OnVideoFrame**

Performs modification of the video frame received from the video hook sink. This method will be called from the drawing thread, with the current video frame, right before the video frame is blitted to the screen. All calculations that occur as a result of this method are delaying the timely delivery of the video frame to the screen, so they should be as optimized as much as possible.

```cpp
STDMETHOD(OnVideoFrame) ( 
    THIS_ 
    UCHAR*pInFrame, 
    ULONG32 ulTimestamp, 
    REF(HXxRect) srcRect, 
    REF(HXxRect) destRect, 
    REF(UCHAR*) pOutFrame 
) PURE;
```

`pInFrame`
Pointer to the input frame buffer.

`ulTimestamp`
The time stamp of the video in the frame buffer.

`srcRect`
Returns an HXxRect structure that defines the source rectangle.

`destRect`
Returns an HXxRect structure that defines the destination rectangle. For example, you can use this information to determine the position of the mouse pointer in the video frame (especially if the frame has been resized). You cannot change this parameter, and it does not have any direct bearing on image size.

`pOutFrame`
Returns a pointer to the output frame buffer. This frame buffer might or might not be a modified version of the input frame buffer. The client is responsible for maintaining and freeing the memory returned in this parameter.
**IHXVideoHookSink**

*Note:* This interface is not currently supported.

**Purpose:** Distributes video frames and events.

**Implemented by:** Client renderer plug-in

**Used by:** Video hooks

**Header file:** hxvsurf.h

This interface, which is implemented in a client renderer (the video hook sink), is used along with IHXVideoHook on a custom renderer plug-in (the video hook) to display a modified video frame on the video surface.

The IHXVideoHookSink interface contains the following methods:

- IHXVideoHookSink::AddVideoHook
- IHXVideoHookSink::ForceSurfaceUpdate
- IHXVideoHookSink::GetVideoHook
- IHXVideoHookSink::GetVideoHookCount
- IHXVideoHookSink::GetInputFormat
- IHXVideoHookSink::GetOutputFormat

As with all COM interfaces, the IHXVideoHookSink interface inherits the following IUnknown methods:

- IUnknown::AddRef
- IUnknown::QueryInterface
- IUnknown::Release

**IHXVideoHookSink::AddVideoHook**

Connects a video hook to the video hook sink. The video hook can then obtain and modify video frames immediately prior to blitting.

```c
STDMETHOD(AddVideoHook) ( 
    THIS_
    IHXVideoHook* pHook, 
    IHXValues* pOptions, 
    HXBitmapInfoHeader* pFrameFormat
) PURE;
```

**pHook**
Pointer to an IHXVideoHook interface that manages the video hook.

**pOptions**
Pointer to an IHXValues interface that manages the optional special flags affecting the use of the video hook by the video hook sink. The flags are:

- ReceivesEvents
- ReceivesHookAdded
- ReceivesFrames
• TakesAnySize

pFrameFormat
Pointer to a preallocated HXBitmapInfoHeader structure that, on exit from this method, is filled in
with the bitmap format in which video frame data will be supplied to IHXVideoHook::OnVideoFrame,
and to which it is expected to conform on exit from IHXVideoHook::OnVideoFrame. If this format is
inappropriate for input or output formats, then after registration the video hook should use either
the IHXVideoHookSink::SetInputFormat or the IHXVideoHookSink::SetOutputFormat method to arrive at
a mutually-acceptable exchange format, or to determine that none is available and take
appropriate action.

IHXVideoHookSink::ForceSurfaceUpdate

Forces the video surface to be redrawn. This method provides a mechanism by which event-driven
video hooks can request that they be given a new image if, for example, the mouse is clicked. The entire
frame is “damaged” so that everything is redrawn.

STDMETHOD(ForceSurfaceUpdate) (THIS ) PURE;

IHXVideoHookSink::GetVideoHook

Provides access to registered video hooks by index. Valid indices range from 0 to n-1 inclusive, where n
is the value returned by IHXVideoHookSink::GetVideoHookCount, provided no sources have been added or
removed since it was last called.

STDMETHOD(GetVideoHook) (THIS_, UINT16 nIndex, REF(IUnknown*) pUnknown ) PURE;

nIndex
The index number of the video hook.

pUnknown
Returns a pointer to the specified IHXVideoHook interface specified by the index number provided
in nIndex.

IHXVideoHookSink::GetVideoHookCount

Returns the number of video hooks registered with this video hook sink. This value is only useful in
the short term as order and count may change for video hooks without notice. Therefore, do not use it
to store access to a given video hook.

STDMETHOD_(UINT16,GetVideoHookCount) (THIS ) PURE;
**IHXVideoHookSink::SetInputFormat**

Provides a means for the video hook sink to negotiate with its registered video hooks about what image formats they require. This method returns HXR_OK if the pRequested parameter exactly matches an acceptable format (color conversion only), and that format is registered for future frame transactions. Otherwise this method returns HXR_FAIL and does not change formats. This method must be used immediately after IHXVideoHookSink::AddVideoHook.

```cpp
STDMETHOD(SetInputFormat) (    
    THIS_,    
    IHXVideoHook* pHook,    
    HXBitmapInfoHeader* pRequested    
) PURE;
```

- **pHook**: Pointer to an IHXVideoHook interface that manages the video hook that is requesting a change in format.
- **pRequested**: Pointer to an HXBitmapInfoHeader structure that contains the format being requested by the video hook.

**IHXVideoHookSink::SetOutputFormat**

Provides a means for the video hook sink to negotiate with its registered video hooks about what image formats they return. This method returns HXR_OK if the pRequested parameter exactly matches an acceptable format (color conversion and size change only), and that format is registered for future frame transactions. Otherwise this method returns HXR_FAIL and does not change formats. This method must be used immediately after IHXVideoHookSink::AddVideoHook.

```cpp
STDMETHOD(SetOutputFormat) (    
    THIS_,    
    IHXVideoHook* pHook,    
    HXBitmapInfoHeader* pRequested    
) PURE;
```

- **pHook**: Pointer to an IHXVideoHook interface that manages the video hook that is requesting a change in format.
- **pRequested**: Pointer to an HXBitmapInfoHeader structure that contains the format that is returned by the video hook.
IHXVideoSurface

Purpose: Provides cross-platform image rendering for Helix clients.
Implemented by: Client core
Used by: Rendering plug-ins
Header file: hxvsurf.h

This interface manages platform-specific rendering details. When you provide the image data, this interface does the rest.

The IHXVideoSurface interface contains the following methods:
• IHXVideoSurface::BeginOptimizedBlt
• IHXVideoSurface::Blt
• IHXVideoSurface::EndOptimizedBlt
• IHXVideoSurface::GetOptimizedFormat
• IHXVideoSurface::GetPreferredFormat
• IHXVideoSurface::OptimizedBlt

As with all Component Object Model (COM) interfaces, the IHXVideoSurface interface inherits the following IUnknown methods:
• IUnknown::AddRef
• IUnknown::QueryInterface
• IUnknown::Release

IHXVideoSurface::BeginOptimizedBlt

Sets the drawing format, such as the size, compression type, and so on, for all future calls to IHXVideoSurface::OptimizedBlt.

STDMETHOD(BeginOptimizedBlt) ( THIS_
            HXBitmapInfoHeader* pBitmapInfo
        ) PURE;

pBitmapInfo
Pointer to an HXBitmapInfoHeader structure that contains the drawing format.

IHXVideoSurface::Blt

Performs a blit from the source rectangle to the destination rectangle.

Note: You can also use IHXVideoSurface::BeginOptimizedBlt and IHXVideoSurface::OptimizedBlt to blit to the surface without reloading the HXBitmapInfoHeader structure for each blit.
STDMETHOD(Blt) (  
    THIS_,  
    UCHAR* pImageBits,  
    HXBitmapInfoHeader* pBitmapInfo,  
    REF(HXxRect) rDestRect,  
    REF(HXxRect) rSrcRect  
) PURE;

pImageBits  
  Pointer to the image data to be displayed.

pBitmapInfo  
  Pointer to an HXBitmapInfoHeader structure that contains the drawing format.

rDestRect  
  Returns an HXxRect structure that describes the destination rectangle.

rSrcRect  
  Returns an HXxRect structure that describes the source rectangle.

IHXVideoSurface::EndOptimizedBlt

  Enables the video surface to clean up after all optimized blit calls have been made.

STDMETHOD(EndOptimizedBlt) (  
    THIS  
) PURE;

IHXVideoSurface::GetOptimizedFormat

  Inquires about the compression type the renderer committed to when it called  
IHXVideoSurface::BeginOptimizedBlt.

STDMETHOD(GetOptimizedFormat) (  
    THIS_  
    REF(HX_COMPRESSION_TYPE) ulType  
) PURE;

ulType  
  Returns the compression type. One of the following:

  • HX_ARGB, ARGB32
  • HX_BITFIELDS, Windows-compatible RGB format
  • HX_I420, planar YCrCb
  • HX_RGB, Windows-compatible RGB format
  • HX_RLE4, Windows-compatible RGB format
  • HX_RLE8, Windows-compatible RGB format
  • HX_UYVY, packed YUV422
  • HX_YUY2, packed YUV422
  • HX_YV12, planar YVU420
• HX_YVU9, Intel YVU9

Also included are some nonstandard FOURCC formats. The following are just a few aliases to what the standard formats can represent and are included only for backward compatibility:

• HX_8BIT_ID, RGB-8 w. pal-e
• HX_RGB24_ID, top-down RGB-24
• HX_RGB3_ID
• HX_RGB555_ID, RGB-16 555
• HX_RGB565_ID, RGB-16 565
• HX_YUV411_ID
• HX_YUV420_ID, planar YCrCb
• HX_YUVRAW_ID

IHXVideoSurface::GetPreferredFormat

Inquires about the compression type the video surface would prefer to be given in IHXVideoSurface::BeginOptimizedBlt.

STDMETHOD(GetPreferredFormat) (THIS_
    REF(HX_COMPRESSION_TYPE) ulType
) PURE;

ulType

Returns the compression type. One of the following:

• HX_ARGB, ARGB32
• HX_BITTIELDS, Windows-compatible RGB format
• HX_I420, planar YCrCb
• HX_RGB, Windows-compatible RGB format
• HX_RLE4, Windows-compatible RGB format
• HX_RLE8, Windows-compatible RGB format
• HX_UYVY, packed YUV422
• HX_YUY2, packed YUV422
• HX_YV12, planar YVU420
• HX_YVU9, Intel YVU9

Also included are some nonstandard FOURCC formats. The following are just a few aliases to what the standard formats can represent and are included only for backward compatibility:

• HX_8BIT_ID, RGB-8 w. pal-e
• HX_RGB24_ID, top-down RGB-24
• HX_RGB3_ID
• HX_RGB555_ID, RGB-16 555
• HX_RGB565_ID, RGB-16 565
• HX_YUV411_ID
• HX_YUV420_ID, planar YCrCb
• HX_YUVRAW_ID

**IHXVideoSurface::OptimizedBlt**

Draws to the video surface in the format previously specified by calling
IHXVideoSurface::BeginOptimizedBlt.

```c
STDMETHOD(OptimizedBlt) (THIS_ 
UCHAR* pImageBits, 
REF(HXxRect) rDestRect, 
REF(HXxRect) rSrcRect 
) PURE;
```

- **pImageBits**
  - Pointer to the image data to be displayed.

- **rDestRect**
  - Returns an HXxRect structure that describes the destination rectangle.

- **rSrcRect**
  - Returns an HXxRect structure that describes the source rectangle.
IHXViewDRMRights

Purpose: Displays the Digital Rights Management (DRM) rights.
Implemented by: Client core
Used by: Top-level client
Header file: hxvsrc.h

The top-level client queries an IHXPlayer for IHXViewDRMRights whenever it would like to view the current DRM rights in the system. The top-level client first inquires to see if a DRM with rights viewing capabilities is present in the system, then the top-level client can actually request to view the actual rights.

The IHXViewDRMRights interface contains the following methods:
  • IHXViewDRMRights::CanViewRights
  • IHXViewDRMRights::ViewRights

As with all Component Object Model (COM) interfaces, the IHXViewDRMRights interface inherits the following IUnknown methods:
  • IUnknown::AddRef
  • IUnknown::QueryInterface
  • IUnknown::Release

IHXViewDRMRights::CanViewRights

Returns TRUE if a DRM is available to provide rights for viewing. Returns FALSE if no DRM is present.

STDMETHOD_(BOOL, CanViewRights) (THIS ) PURE;

IHXViewDRMRights::ViewRights

Displays the DRM rights in an HTML Web browser.

STDMETHOD(ViewRights) (THIS ) PURE;
IHXViewSourceCommand

Purpose: Provides access to view source functionality.
Implemented by: Client core
Used by: Top-level client
Header file: hxvsrc.h

The top-level client queries an IHXPlayer for IHXViewSourceCommand whenever it needs to take advantage of view source functionality. The top-level client first inquires to see whether a particular stream supports ViewSource, then the top-level client can actually request for the source to be viewed on that stream. These streams default to NULL. For the default case, the IHXViewSourceCommand interface decides which stream to use for view source. If there is a persistent stream (SMIL), it is viewed; otherwise the first stream in the player is viewed. The response interface is IHXViewSourceURLResponse.

The IHXViewSourceCommand interface contains the following methods

• IHXViewSourceCommand::CanViewSource
• IHXViewSourceCommand::DoViewSource
• IHXViewSourceCommand::GetViewSourceURL

As with all Component Object Model (COM) interfaces, the IHXViewSourceCommand interface inherits the following IUnknown methods:

• IUnknown::AddRef
• IUnknown::QueryInterface
• IUnknown::Release

IHXViewSourceCommand::CanViewSource

Determines whether a specified stream supports ViewSource. Returns TRUE if the stream does support ViewSource.

STDMETHOD_(BOOL, CanViewSource) ( 
    THIS_
    IHXStreamSource* pStream 
) PURE;

pStream
Pointer to an IHXStreamSource interface that manages the particular stream being queried for its support of ViewSource.

IHXViewSourceCommand::DoViewSource

Requests a source be viewed on the specified stream.

STDMETHOD(DoViewSource) ( 
    THIS_
    IHXStreamSource* pStream 
) PURE;
pStream
    Pointer to an IHXStreamSource interface that manages the stream on which the source is to be viewed.

IHXViewSourceCommand::GetViewSourceURL

Requests the URL of the source to be viewed.

STDMETHOD(GetViewSourceURL) ( 
    THIS_
    IHXStreamSource* pSource, 
    IHXViewSourceURLResponse* pResp 
) PURE;

pSource
    Pointer to an IHXStreamSource interface that manages the stream on which the source is to be viewed.

pResp
    Pointer to an IHXViewSourceURLResponse interface that manages the response to this method.
IHXViewSourceURLResponse

Purpose: Provides the URL for the source to be viewed.
Implemented by: Top-level client
Used by: Client core
Header file: hxvsrc.h

This is the response interface for the IHXViewSourceCommand interface.
The IHXViewSourceURLResponse interface contains the IHXViewSourceURLResponse::ViewSourceURLReady method.

As with all Component Object Model (COM) interfaces, the IHXViewSourceURLResponse interface inherits the following IUnknown methods:
• IUnknown::AddRef
• IUnknown::QueryInterface
• IUnknown::Release

IHXViewSourceURLResponse::ViewSourceURLReady

Provides the URL of the source to be viewed.

```c
STDMETHOD(ViewSourceURLReady) ( 
    const char* pUrl
) PURE;

pUrl
    Pointer to the URL that contains the source to be viewed.
```
IHXVolume

Purpose: Controls audio volume.
Implemented by: Audio stream objects, audio device object (Audio Services)
Used by: Audio rendering plug-ins
Header file: hxausvc.h

Through this interface, an audio rendering plug-in can query, set, and mute audio volume, as well as register to receive volume change notifications through IHXVolumeAdviseSink. The plug-in gets a pointer to the volume object for an individual stream through IHXAudioStream. It gets a pointer to the volume object for the final mixed stream or the audio device object from IHXAudioPlayer.


The IHXVolume interface contains the following methods:
- IHXVolume::AddAdviseSink
- IHXVolume::GetMute
- IHXVolume::GetVolume
- IHXVolume::RemoveAdviseSink
- IHXVolume::SetMute
- IHXVolume::SetVolume

As with all Component Object Model (COM) interfaces, the IHXVolume interface inherits the following IUnknown methods:
- IUnknown::AddRef
- IUnknown::QueryInterface
- IUnknown::Release

IHXVolume::AddAdviseSink
Associates a volume advise sink interface with this interface.

STDMETHOD(AddAdviseSink) (  
    THIS_  
    IHXVolumeAdviseSink* pSink  
) PURE;

pSink  
    Pointer to an IHXVolumeAdviseSink interface that advises the calling component of changes in volume and muting.

IHXVolume::GetMute
Determines whether the volume is muted. Returns TRUE if the volume is muted.

STDMETHOD_(BOOL,GetMute) (  
    THIS  
) PURE;
**IHXVolume::GetVolume**

Returns the current volume level.

```cpp
STDMETHOD_(UINT16, GetVolume) (THIS)
) PURE;
```

**IHXVolume::RemoveAdviseSink**

Removes a previously associated volume advise sink interface. Use this method when you no longer need to receive volume or mute change notifications.

```cpp
STDMETHOD(RemoveAdviseSink) (THIS_
    IHXVolumeAdviseSink* pSink
) PURE;
```

**pSink**

Pointer to the IHXVolumeAdviseSink interface to be removed.

**IHXVolume::SetMute**

Mutes the volume.

```cpp
STDMETHOD(SetMute) (THIS_
    const BOOL bMute
) PURE;
```

**bMute**

If TRUE, the volume is muted.

**IHXVolume::SetVolume**

Sets the volume level.

```cpp
STDMETHOD(SetVolume) (THIS_
    const UINT16 uVolume
) PURE;
```

**uVolume**

The volume level, in the range of 1 to 100.
IHXVolumeAdviseSink

Purpose: Receives notice of changes in audio volume.
Implemented by: Audio rendering plug-ins, top-level clients
Used by: Audio Services
Header file: hxausvc.h

Through the IHXVolume interface, an audio rendering plug-in or the top-level client can register to receive volume change notifications. Audio Services then uses the IHXVolumeAdviseSink interface to inform the component of changes in volume or muting.


The IHXVolumeAdviseSink interface contains the following methods:

- IHXVolumeAdviseSink::OnMuteChange
- IHXVolumeAdviseSink::OnVolumeChange

As with all Component Object Model (COM) interfaces, the IHXVolumeAdviseSink interface inherits the following IUnknown methods:

- IUnknown::AddRef
- IUnknown::QueryInterface
- IUnknown::Release

IHXVolumeAdviseSink::OnMuteChange

Indicates a mute change has occurred. This interface is called whenever the associated IHXVolume::SetMute method is called.

STDMETHOD(OnMuteChange) ( 
  THIS_ const BOOL bMute
) PURE;

bMute
If TRUE, the volume has been muted.

IHXVolumeAdviseSink::OnVolumeChange

Indicates a volume change has occurred. This method is called whenever the associated IHXVolume::SetVolume method is called.

STDMETHOD(OnVolumeChange) ( 
  THIS_ const UINT16 uVolume
) PURE;

uVolume
The new volume level, in the range of 1 to 100.
IHXWantServerReconfigNotification

Purpose: Provides reconfiguration notification from the server.
Implemented by: Server plug-in
Used by: Server core
Header file: hxcomm.h

This interface notifies the user the server received the reconfiguration request and it is now time for you to do your reconfiguration.

Note: This interface is not needed if all of your configuration information is stored in the configuration file; that is taken care of through IHXActiveRegistry.

The IHXWantServerReconfigNotification interface contains the IHXWantServerReconfigNotification::ServerReconfig method.

As with all Component Object Model (COM) interfaces, the IHXWantServerReconfigNotification interface inherits the following IUnknown methods:
  • IUnknown::AddRef
  • IUnknown::QueryInterface
  • IUnknown::Release

IHXWantServerReconfigNotification::ServerReconfig

Notifies the user that a server reconfiguration request has come in and you can now do a reconfiguration on your separate server configuration file.

Note: This method cannot be used to reconfigure the server configuration file.

STDMETHOD(ServerReconfig) (THIS_
    IHXReconfigServerResponse* pResponse
) PURE;

pResponse
  Pointer to an IHXReconfigServerResponse interface that manages the server reconfiguration response.
IHXXMLParser

Purpose: Parses XML documents.
Implemented by: XML Parser plug-in
Used by: Server, client, data type plug-ins
Header file: hxxml.h

This interface provides a means for an XML document to be parsed to its component parts. A buffer containing all or part of an XML document is passed to the IHXXMLParser::Parse method, which then parses the document and passes the results to the appropriate IHXXMLParserResponse method.

The IHXXMLParser interface contains the following methods:

• IHXXMLParser::Close
• IHXXMLParser::GetCurrentByteIndex
• IHXXMLParser::GetCurrentColumnNumber
• IHXXMLParser::GetCurrentErrorText
• IHXXMLParser::GetCurrentLineNumber
• IHXXMLParser::Init
• IHXXMLParser::Parse

As with all Component Object Model (COM) interfaces, the IHXXMLParser interface inherits the following IUnknown methods:

• IUnknown::AddRef
• IUnknown::QueryInterface
• IUnknown::Release

IHXXMLParser::Close

Releases all resources associated with the parser component.

STDMETHOD(Close) (THIS) PURE;

IHXXMLParser::GetCurrentByteIndex

Gets the byte index of the location of the parser's internal cursor.

STDMETHOD(GetCurrentByteIndex) (THIS_
    REF(ULONG32) ulByteIndex
) PURE;

ulByteIndex
    Returns the byte index that identifies the internal cursor's position.

IHXXMLParser::GetCurrentColumnNumber

Gets the column number of the location of the parser's internal cursor.
STDMETHOD(GetCurrentColumnNumber) (  
    THIS_  
    REF(ULONG32) ulColumnNumber  
) PURE;

ulColumnNumber  
Returns the column number that identifies the internal cursor’s position.

IHXXMLParser::GetCurrentErrorText  
Gets the text of the last error.

STDMETHOD(GetCurrentErrorText) (  
    THIS_  
    REF(IHXBuffer*) pBuffer  
) PURE;

pBuffer  
Returns a pointer to an IHXBuffer interface that manages the text of the last error.

IHXXMLParser::GetCurrentLineNumber  
Gets the line number of the position of the parser’s internal cursor.

STDMETHOD(GetCurrentLineNumber) (  
    THIS_  
    REF(ULONG32) ulLineNumber  
) PURE;

ulLineNumber  
Returns the line number that identifies the internal cursor’s position.

IHXXMLParser::Init  
Initializes an XML parser component.

STDMETHOD(Init) (  
    THIS_  
    IHXXMLParserResponse* pResponse,  
    const char* pEncoding,  
    BOOL bStrict  
) PURE;

pResponse  
Pointer to an IHXXMLParserResponse interface that manages the parsed XML components.

pEncoding  
The type of encoding. This parameter is normally set to NULL, but it can be any supported character-encoding string (it is usually safe to report NULL for the encoding unless you have a specific character encoding to use). The following are commonly supported encodings:

• Default (assumes DBCS; any non-ASCII character is a lead byte)
• US-ASCII
• Shift-JIS

- Big5
- GB2312
- EUC-KR
- ISO-8859-1

bStrict
If this parameter is set to FALSE, the client allows a “loose” interpretation of the XML specification, including tag/attribute case insensitivity.

IHXXMLParser::Parse
Parses the buffer containing all or part of an XML document. You can pass an XML document to this method in several pieces; each piece must be passed in the same order as it appears in the XML document. The portion of each piece of the XML document passed in this buffer is determined by the user, although a minimum should be a single XML component (such as a single declaration, a single tag, or a single processing instruction). As each XML component is parsed, this method calls the appropriate IHXXMLParserResponse method for that component.

 STDMETHODCALLTYPE (Parse) {
    THIS_
    IHXBuffer* pBuffer,
    BOOL bIsFinal
) PURE;

pBuffer
Pointer to an IHXBuffer interface that manages the XML code to be parsed.

bIsFinal
If set to TRUE, the buffer in pBuffer is the last to be parsed.
IHXXMLParserResponse

Purpose: Manages the pieces of parsed XML documents.

Implemented by: Third-party clients, plug-ins

Used by: XML Parser Plug-in

Header file: hxml.h

This interface receives the parsed pieces of the XML document. After the IHXXMLParser interface has parsed the buffer containing the document, it calls the appropriate method for the parsed component, supplying the method with information about that component.

Note: You can parse an XML document by splitting the document into several pieces, passing each piece in a buffer, one after the other in the order in which they appear in the document, to the IHXXMLParser::Parse method. The line number and column number specified in each of the IHXXMLParserResponse methods indicates the location of the tag or entity within the XML document, not the location in the parsed buffer.

The IHXXMLParserResponse interface contains the following methods:

- IHXXMLParserResponse::HandleCharacterData
- IHXXMLParserResponse::HandleComment
- IHXXMLParserResponse::HandleDefault
- IHXXMLParserResponse::HandleEndElement
- IHXXMLParserResponse::HandleNotationDecl
- IHXXMLParserResponse::HandleProcessingInstruction
- IHXXMLParserResponse::HandleStartElement
- IHXXMLParserResponse::HandleUnparsedDoctypeDecl
- IHXXMLParserResponse::HandleUnparsedEntityDecl

As with all Component Object Model (COM) interfaces, the IHXXMLParserResponse interface inherits the following IUnknown methods:

- IUnknown::AddRef
- IUnknown::QueryInterface
- IUnknown::Release

IHXXMLParserResponse::HandleCharacterData

Contains the XML character data parsed by IHXXMLParser::Parse.

STDMETHOD(HandleCharacterData) (THIS_
  IHXBuffer* pBuffer,
  UINT32 ulLineNumber,
  UINT32 ulColumnNumber
) PURE;

pBuffer
  Pointer to an IHXBuffer interface that manages the character data that was parsed.
ulLineNumber  
The line number in the XML document indicating the start of the character data that was parsed.

ulColumnNumber  
The column number in the XML document indicating the start of the character data that was parsed.

IHXXMLParserResponse::HandleComment
Contains the XML comment information (<!-- this is a comment -->) parsed by IHXXMLParser::Parse.

 STDMETHOD(HandleComment) ( 
   THIS_  
   const char* pComment,  
   UINT32 ulLineNumber,  
   UINT32 ulColumnNumber  
) PURE;

pComment  
Pointer to the XML comment information that was parsed.

ulLineNumber  
The line number in the XML document indicating the start of the comment information that was parsed.

ulColumnNumber  
The line number in the XML document indicating the start of the comment information that was parsed.

IHXXMLParserResponse::HandleDefault
Contains any XML component not explicitly handled by the existing IHXXMLParser methods. This method is also included so that you can enhance the XML parser to support any new XML components and then pass them back through this method.

 STDMETHOD(HandleDefault) ( 
   THIS_  
   IHXBuffer* pBuffer,  
   UINT32 ulLineNumber,  
   UINT32 ulColumnNumber  
) PURE;

pBuffer  
Pointer to an IHXBuffer interface that manages the XML component.

ulLineNumber  
The line number in the XML document indicating the start of the XML component that was parsed.

ulColumnNumber  
The line number in the XML document indicating the start of the XML component that was parsed.
IHXXMMLParserResponse::HandleEndElement

Contains the XML end tag (<\tag>) that was parsed by IHXXMMLParser::Parse.

STDMETHOD(HandleEndElement) ( 
    THIS_ 
    const char* pName, 
    UINT32 ulLineNumber, 
    UINT32 ulColumnNumber  
) PURE;

pName
Pointer to the name of the end tag that was parsed.

ulLineNumber
The line number in the XML document indicating the start of the end tag that was parsed.

ulColumnNumber
The column number in the XML document indicating the start of the end tag that was parsed.

IHXXMMLParserResponse::HandleNotationDecl

Contains the XML notation information (<!NOTATION gif PUBLIC "http://www.gif.com/gif.not">) that was parsed by IHXXMMLParser::Parse.

STDMETHOD(HandleNotationDecl) ( 
    THIS_ 
    const char* pNotationName, 
    const char* pSystemID, 
    const char* pPublicID, 
    UINT32 ulLineNumber, 
    UINT32 ulColumnNumber  
) PURE;

pNotationName
Pointer to the root element name of the notation (in the preceding example, this would be a pointer to “gif”).

pSystemID
Pointer to the external identifier (if a SYSTEM keyword is used) or a pointer to the public identifier (if a PUBLIC keyword is used).

pPublicID
Pointer to the relative or absolute URL for the location where the public identifier resides. If the SYSTEM keyword is used, this parameter is set to NULL.

ulLineNumber
The line number in the XML document indicating the start of the notation information that was parsed.

ulColumnNumber
The column number in the XML document indicating the start of the notation information that was parsed.
IHXXMLParserResponse::HandleProcessingInstruction

Contains the XML processing instruction (<?foobar flotz="fred" glootz="mary"?>) that was parsed by IHXXMLParser::Parse.

STDMETHOD(HandleProcessingInstruction) (  
    THIS_
    const char* pTarget,
    IHXValues* pAttributes,
    UINT32 ulLineNumber,
    UINT32 ulColumnNumber
) PURE;

pTarget
    Pointer to the target that identifies the application to which the instruction is directed.

pAttributes
    Pointer to an IHXValues interface that manages the data to be given to the application.

ulLineNumber
    The line number in the XML document indicating the start of the processing instruction that was parsed.

ulColumnNumber
    The column number in the XML document indicating the start of the processing instruction that was parsed.

IHXXMLParserResponse::HandleStartElement

Contains the XML start tag (<tag a="foo" b="bar">) that was parsed by IHXXMLParser::Parse.

STDMETHOD(HandleStartElement) (  
    THIS_
    const char* pName,
    IHXValues* pAttributes,
    UINT32 ulLineNumber,
    UINT32 ulColumnNumber
) PURE;

pName
    Pointer to the name of the start tag.

pAttributes
    Pointer to an IHXValues interface that manages the tag attributes.

ulLineNumber
    The line number in the XML document indicating the start of the start tag that was parsed.

ulColumnNumber
    The column number in the XML document indicating the start of the start tag that was parsed.

IHXXMLParserResponse::HandleUnparsedDoctypeDecl

Contains the XML document type declaration information (<!DOCTYPE foo SYSTEM "foo.dtd">) that was parsed by IHXXMLParser::Parse.
STDMETHOD(HandleUnparsedDoctypeDecl) ( 
    THIS_,
    const char* pDoctype,
    const char* pSystemID,
    const char* pPublicID,
    UINT32 ulLineNumber,
    UINT32 ulColumnNumber
) PURE;

pDoctype
Pointer to the root element name of the document type (in the preceding example, this would be a pointer to “foo”).

pSystemID
Pointer to the external identifier (if a SYSTEM keyword is used) or a pointer to the public identifier (if a PUBLIC keyword is used).

pPublicID
Pointer to the relative or absolute URL for the location where the public identifier resides. If the SYSTEM keyword is used, this parameter is set to NULL.

ulLineNumber
The line number in the XML document indicating the start of the document type declaration that was parsed.

ulColumnNumber
The column number in the XML document indicating the start of the document type declaration that was parsed.

IHXXMLParserResponse::HandleUnparsedEntityDecl
Contains XML entity reference information (<!ENTITY hatch-pic SYSTEM "../grafix/OpenHatch.gif" NDATA gif>that was parsed by IHXXMLParser::Parse.

STDMETHOD(HandleUnparsedEntityDecl) ( 
    THIS_,
    const char* pEntityName,
    const char* pSystemID,
    const char* pPublicID,
    const char* pNotationName,
    UINT32 ulLineNumber,
    UINT32 ulColumnNumber
) PURE;

pEntityName
Pointer to the name associated with this entity.

pSystemID
Pointer to the external identifier (if a SYSTEM keyword is used) or a pointer to the public identifier (if a PUBLIC keyword is used).

pPublicID
Pointer to the relative or absolute URL for the location where the public identifier resides. If the SYSTEM keyword is used, this parameter is set to NULL.
**pNotationName**
   - Pointer to the entity’s type if the external entity is not XML (in the preceding example, this would be a pointer to “gif”).

**ulLineNumber**
   - The line number in the XML document indicating the start of the entity reference that was parsed.

**ulColumnNumber**
   - The column number in the XML document indicating the start of the entity reference that was parsed.
**IHXXMXMLTagHandler**

- **Purpose:** Registers a specific tag.
- **Implemented by:** Tag handler plug-ins
- **Used by:** Server tag file-system plug-in
- **Header file:** hxxmltg.h

This interface creates tag objects and registers tags.

The `IHXXMXMLTagHandler` interface contains the following methods:

- `IHXXMXMLTagHandler::CreateTagObject`
- `IHXXMXMLTagHandler::GetTagHandlerInfo`
- `IHXXMXMLTagHandler::InitTagHandler`

As with all Component Object Model (COM) interfaces, the `IHXXMXMLTagHandler` interface inherits the following `IUnknown` methods:

- `IUnknown::AddRef`
- `IUnknown::QueryInterface`
- `IUnknown::Release`

**IHXXMXMLTagHandler::CreateTagObject**

Creates a tag object to manage tags.

```cpp
STDMETHOD(CreateTagObject) (THIS_ IHXXMXMLTagObject** ppObj)
```

- `ppObj` Pointer to a pointer to an `IHXXMXMLTagObject` interface that manages the tag object.

**IHXXMXMLTagHandler::GetTagHandlerInfo**

Indicates what tags the tag object supports (that is, the tag object created by `IHXXMXMLTagHandler::CreateTagObject`).

```cpp
STDMETHOD(GetTagHandlerInfo) (THIS_ REF(const char*) pShortName, REF(const char**) pTag, REF(BOOL) bCanHandleAsync)
```

- `pShortName` Returns a pointer to the short name of the tag handler, such as `hx- vsrctaghdlr` or `hx-includer`. The short name is placed in the list of tag handlers the tag file system is configured to use.
pTag
   Returns a pointer to a pointer to a string list of tags. IHXXMLTagObject::OnTag can then be called for each of these tags.

bCanHandleAsync
   Returns TRUE if the handler can handle more than one request at a time. Returns FALSE if all requests must be synchronous.

**IHXXMLTagHandler::InitTagHandler**

Initializes the tag handler before it can create tags.

```cpp
STDMETHOD(InitTagHandler) ( 
   THIS_ 
   IHXValues* pOptions 
) PURE;
```

pOptions
   Pointer to an IHXValues interface that manages the options in the server configuration file for this handler.
Once initialized, this interface provides tag notification for files that pass through the tag file system, specifically tags the tag handler indicated should be handled in the IHXXMLTagHandler::GetTagHandlerInfo method. This interface is called either asynchronously or synchronously, depending on what was returned by the IHXXMLTagHandler::GetTagHandlerInfo method.

The IHXXMLTagObject interface contains the following methods:

- IHXXMLTagObject::Close
- IHXXMLTagObject::InitTagObject
- IHXXMLTagObject::OnTag

As with all Component Object Model (COM) interfaces, the IHXXMLTagObject interface inherits the following IUnknown methods:

- IUnknown::AddRef
- IUnknown::QueryInterface
- IUnknown::Release

### IHXXMLTagObject::Close

Closes the tag object after the tag object is finished.

```cpp
HRESULT Close();
```

### IHXXMLTagObject::InitTagObject

Initializes the tag object to prepare the tag object to handle tags.

```cpp
HRESULT InitTagObject(IHXXMLTagObjectResponse* pResp);
```

- `pResp` Pointer to an IHXXMLTagObjectResponse interface that manages the response to this interface.

### IHXXMLTagObject::OnTag

Called for every tag in a document for which the handler was registered, and passes through the mount point on which the tag file system is mounted.
STDMETHOD(OnTag) ( 
  THIS_ 
  UINT32 ulInstance, 
  IHXBuffer* pTag 
) PURE;

ulInstance
In the case of asynchronous requests, this parameter indicates the instance of the tag request for 
IHXXMLTagObjectResponse::OnTagDone.

pTag
Pointer to an IHXBuffer interface that manages the tag to be registered for callbacks.
**IHXXMTagObjectResponse**

Purpose: Returns the replacement for a tag.

Implemented by: Server plug-in tag file system

Used by: Tag objects

Header file: hxxmltg.h

This interface returns a replacement tag for the IHXXMLTagObject::OnTag method.

The IHXXMLTagObjectResponse interface contains the IHXXMLTagObjectResponse::OnTagDone method.

As with all Component Object Model (COM) interfaces, the IHXXMLTagObjectResponse interface inherits the following IUnknown methods:

- IUnknown::AddRef
- IUnknown::QueryInterface
- IUnknown::Release

**IHXXMLTagObjectResponse::OnTagDone**

Returns the replacement tag for the specified tag instance.

```c
STDMETHOD(OnTagDone) ( 
    THIS_ 
    UINT32 ulInstance, 
    IHXBuffer* pTag 
) PURE;
```

**ulInstance**

In the case of asynchronous requests, this parameter indicates the instance of the tag request from IHXXMLTagObject::OnTag.

**pTag**

Pointer to an IHXBuffer interface that manages the tag object to be inserted in the document in place of the tag with which IHXXMLTagObject::OnTag was called. For example:

```c
OnTag(1, "<myTag options>")
OnTagDone(1, "Text to be inserted instead of <myTag options>")
```
IUnknown

The IUnknown interface is the basis of all Component Object Model (COM) interfaces. This interface contains a set of methods that control the lifetime of a specific object, and provides a means of querying for the interfaces used by an object. The IUnknown interface includes the following methods:

- IUnknown::AddRef
- IUnknown::QueryInterface
- IUnknown::Release

IUnknown::AddRef

Increases the object's reference count by one. Whenever an object is created, its reference count begins at 1. If an application calls IUnknown::AddRef, queries an interface belonging to a specific object, or uses a creation function like HXCreateInstance, the reference count is incremented by 1.

STDMETHOD_(ULONG32, AddRef) (THIS) PURE;

Return Values

Returns the new reference count.

Note: Use the IUnknown::Release method to decrease the reference count by 1.

IUnknown::QueryInterface

Queries an object to determine whether it supports a specific interface. If the call succeeds, you can then use the methods belonging to that interface.

STDMETHOD(QueryInterface) (THIS_
    REFIID riid,
    void** ppvObj
) PURE;

riid

Indicates the reference identifier of the interface being queried.

ppvObj

Points to an interface pointer that is filled in if the query succeeds.

Return Values

Returns HXR_OK if successful, or one of the following values:

- HXR_FAIL
- HXR_NOINTERFACE
- HXR_NOTIMPL
- HXR_OUTOFMEMORY
IUnknown::Release

Decreases the object's reference count by one. Every call to IUnknown::AddRef, IUnknown::QueryInterface, or a creation function such as HXCreateInstance must have a corresponding call to IUnknown::Release. When the reference count reaches 0 (zero), the object is destroyed.

```c
STDMETHOD_(ULONG32, Release) (THIS ) PURE;
```

Return Values

Returns the new reference count.
STRUCTURE LIST

HXAudioData
Contains audio data that is passed from audio renderers to the audio stream.

typedef struct _HXAudioData
{
    IHXBuffer* pData;
    ULONG32 ulAudioTime;
    AudioStreamType uAudioStreamType;
} HXAudioData;

pData
    Pointer to an IHXBuffer interface that manages the audio data.

ulAudioTime
    The start time, in milliseconds, relative to the start of the player time line for the stream.

uAudioStreamType
    An AudioStreamType enumerator that describes the type of audio stream. This can be any of the following:

    • STREAMING_AUDIO
      Indicates the audio data is being written in streaming format. The core then uses a fudge factor of 5 milliseconds to determine whether the current audio packet is to be played right after the last-written audio packet.

    • INSTANTANEOUS_AUDIO
      Indicates the audio data should be played as soon as possible.

    • TIMED_AUDIO
      Indicates the audio data is to be played at a specific time in the presentation. Typically for streaming audio, the first audio packet is written with this flag as subsequently with the STREAMING_AUDIO flag. If a jump in time ever occurs while writing packets (because of a lack of a seek or a bunch of lost packets), the user should again write a packet with this flag before continuing with the STREAMING_AUDIO flag.

HXAudioFormat
Contains the format of the audio device.
typedef struct _HXAudioFormat
{
    UINT16 uChannels;
    UINT16 uBitsPerSample;
    ULONG32 ulSamplesPerSec;
    UINT16 uMaxBlockSize;
} HXAudioFormat;

uChannels
    The number of channels. For example: 1 = monaural, 2 = stereo, and so on.

uBitsPerSample
    The number of bits per sample. This member can be either 8 or 16.

ulSamplesPerSec
    The sampling rate per second.

uMaxBlockSize
    The maximum size of an IHXBuffer that will ever be written to an audio stream in a write call.

HXBitmapInfo

Defines dimensions and color information for a bitmap.

typedef struct _HXBitmapInfo
{
    struct
    {
        UINT32 biSize;
        INT32 biWidth;
        INT32 biHeight;
        UINT16 biPlanes;
        UINT16 biBitCount;
        UINT32 biCompression;
        UINT32 biSizeImage;
        INT32 biXPelsPerMeter;
        INT32 biYPelsPerMeter;
        UINT32 biClrUsed;
        UINT32 biClrImportant;
    } bmiHeader;
    union
    {
        UINT32 dwBitMask[3];
        UINT32 dwPalette[256];
    } un;
} HXBitmapInfo;

biSize
    This member is calculated by using the offset to this structure’s dwBitMask member—that is, OFFSETOF(dwBitMask).

biWidth
    The width of the image, in pixels.
**biHeight**
The height of the image, in pixels

**biPlanes**
The number of bit planes. This member must always be set to 1.

**biBitCount**
The average number of bits per pixel.

**biCompression**
The compression type. This can be any of the following:
- RMA_ARGB, ARGB32
- HX_BITFIELDS, Windows-compatible RGB format
- HX_I420, planar YCrCb
- HX_RGB, Windows-compatible RGB format
- HX_RLE4, Windows-compatible RGB format
- HX_RLE8, Windows-compatible RGB format
- HX_UYVY, packed YUV422
- HX_YUY2, packed YUV422
- HX_YV12, planar YUV420
- HX_YVU9, Intel YVU9

Also included are some nonstandard FOURCC formats. The following compression types are just aliases to what can be represented by the standard formats; as such they are included here for backward compatibility only:
- HX_8BIT_ID, RGB-8 w. pal-e
- HX_RGB24_ID, top-down RGB-24
- HX_RGB3_ID
- HX_RGB555_ID, RGB-16 555
- HX_RGB565_ID, RGB-16 565
- HX_YUV411_ID
- HX_YUV420_ID, planar YCrCb
- HX_YUV420_ID

**biSizeImage**
The size of the image. This member is equal to `biWidth * biHeight * biBitCount / 8`.

**biXPelsPerMeter**
This member is always 0 (zero).

**biYPelsPerMeter**
This member is always 0 (zero).
biClrUsed
The number of color indexes in the color table that are actually used by the bitmap.

biClrImportant
Specifies the number of color indexes that are considered important for displaying the bitmap. If this value is 0 (zero), all colors are important.

dwBitMask[3]
The color masks (for HX_BITFIELDS).

dwPalette[256]
The palette (for an 8-bit RGB image).

HXBitmapInfoHeader
Contains the dimensions and color format information for a device-independent bitmap (DIB).

typedef struct _HXBitmapInfoHeader
{
    UINT32 biSize;
    INT32 biWidth;
    INT32 biHeight;
    UINT16 biPlanes;
    UINT16 biBitCount;
    UINT32 biCompression;
    UINT32 biSizeImage;
    INT32 biXPelsPerMeter;
    INT32 biYPelsPerMeter;
    UINT32 biClrUsed;
    UINT32 biClrImportant;
    UINT32 rcolor;
    UINT32 gcolor;
    UINT32 bcolor;
} HXBitmapInfoHeader;

biSize
The size of this structure, in bytes.

biWidth
The width of the bitmap, in pixels.

biHeight
The height of the bitmap, in pixels. If biHeight is positive, the bitmap is a bottom-up DIB and its origin is the lower left corner. If biHeight is negative, the bitmap is a top-down DIB and its origin is the upper left corner.

biPlanes
The number of planes for the target device. This parameter must be set to 1.

biBitCount
The number of bits per pixel. This parameter must be either 1, 4, 8, 16, 24, or 32.
biCompression
The type of compression for a compressed bottom-up bitmap (top-down DIBs can not be compressed). This can be any of the following:

- **BI_RGB**
  An uncompressed format.
- **BI_RLE8**
  A run-length encoded (RLE) format for bitmaps with 8 bits per pixel. The compression format is a 2-byte format consisting of a count byte followed by a byte containing a color index.
- **BI_RLE4**
  An RLE format for bitmaps with 4 bits per pixel. The compression format is a 2-byte format consisting of a count byte followed by two word-length color indexes.
- **BI_BITFIELDS**
  The bitmap is not compressed and the color table consists of three double-word color masks that specify the red, green, and blue components (respectively) of each pixel. This is valid when used with 16- and 32-bits-per-pixel bitmaps.

biSizeImage
The size of the image, in bytes. You can set this to 0 (zero) for BI_RGB bitmaps.

biXPelsPerMeter
Specifies the horizontal resolution, in pixels per meter, of the target device for the bitmap. An application can use this value to select a bitmap from a resource group that best matches the characteristics of the current device.

biYPelsPerMeter
Specifies the vertical resolution, in pixels per meter, of the target device for the bitmap.

biClrUsed
Specifies the number of color indexes in the color table that are actually used by the bitmap. If this value is 0 (zero), the bitmap uses the maximum number of colors corresponding to the value of the biBitCount member for the compression mode specified by biCompression.

If this parameter is not 0 (zero) and the biBitCount member is less than 16, this parameter specifies the actual number of colors the graphics engine or device driver accesses. If biBitCount is 16 or greater, then this parameter specifies the size of the color table used to optimize performance of Windows color palettes. If biBitCount equals 16 or 32, the optimal color palette starts immediately following the three double-word masks.

If the bitmap is a packed bitmap (a bitmap in which the bitmap array immediately follows the BITMAPINFO header and is referenced by a single pointer), this parameter must be either 0 or the actual size of the color table.

biClrImportant
The number of color indexes considered important for displaying the bitmap. If this parameter is 0 (zero), all colors are important.

rcolor
The red color mask from the dwBitMask[0] member of the HXBitmapInfo structure.
The green color mask from the dwBitMask[1] member of the HXBitmapInfo structure.

The blue color mask from the dwBitMask[2] member of the HXBitmapInfo structure.

**HXTimeval**

Contains a time value.

```c
typedef struct _HXTimeval
{
    UINT32 tv_sec;
    UINT32 tv_usec;
} HXTimeval;
```

- **tv_sec**
  The number of seconds in the current time.

- **tv_usec**
  The number of microseconds in the current time.

**HXxEvent**

Contains a cross-platform definition of an event. This structure is sufficiently flexible to describe an event in Windows, MacOS, and various versions of X-Windows. For more information on setting the members of this structure, see the file hxevent.h.

```c
typedef struct HXEXPORT_CLASS _HXxEvent
{
    ULONG32 event;    /* IN */
    void* window;     /* IN */
    void* param1;     /* IN */
    void* param2;     /* IN */
    UINT32 result;    /* OUT */
    BOOL handled;     /* OUT */
} HXxEvent;
```

- **event**
  Platform-specific event ID. This member contains one of the several Helix Architecture (HX) event IDs specified in the hxevent.h file that map onto existing platform-specific event IDs. For example, in UNIX this member maps to the X event type.

- **window**
  Pointer to a platform-specific window handle. For example, in UNIX this is a pointer to the X window ID.

- **param1**
  Pointer to a message-specific parameter. For example, in UNIX this is a pointer to Display*.

- **param2**
  Pointer to a message-specific parameter. For example, in UNIX this is a pointer to “Native XEvent*” “HX_SURFACE_UPDATE HXxWindow*”.
result
   The result of message handling.

handled
   If TRUE, the event was handled. If FALSE, the event was not handled.

**HXxPoint**

Contains a cross-platform definition of a point.

typedef struct HXEXPORT_CLASS _HXxPoint
{
   INT32 x;
   INT32 y;
} HXxPoint;

x
   The horizontal component of the point.

y
   The vertical component of the point.

**HXxRect**

Contains a cross-platform definition of a rectangle.

typedef struct HXEXPORT_CLASS _HXxRect
{
   INT32 left;
   INT32 top;
   INT32 right;
   INT32 bottom;
} HXxRect;

left
   The x-coordinate of the upper-left corner of the rectangle.

top
   The y-coordinate of the upper-left corner of the rectangle.

right
   The x-coordinate of the lower-right corner of the rectangle.

bottom
   The y-coordinate of the lower-right corner of the rectangle.

**HXxSize**

Contains a cross-platform definition of a size.
typedef struct HXEXPORT_CLASS _HXxSize
{
    INT32 cx;
    INT32 cy;
} HXxSize;

**cx**

The horizontal size, in pixels.

**cy**

The vertical size, in pixels.

**HXxWindow**

Contains a cross-platform definition of a window. This structure is sufficiently wide to describe parent or child windows in Windows, MacOS, and various flavors of X-Windows.

**Note:** The window parameter is not guaranteed to be unique for every corresponding CHXWindow. Use HXxWindowID (defined in hwxtyp.h) if you want this uniqueness.

typedef struct HXEXPORT_CLASS _HXxWindow
{
    void* window;
    ULONG32 x;
    ULONG32 y;
    ULONG32 width;
    ULONG32 height;
    HXxRect clipRect;
    #ifdef _UNIX
    void * display;
    #endif
} HXxWindow;

**window**

Pointer to a platform-specific window handle.

**x**

Horizontal position of a point. Together with y, these parameters constitute the position of the upper-left corner relative to a client page.

**y**

Vertical position of a point.

**width**

The maximum width of the window, in pixels.

**height**

The maximum height of the window, in pixels.

**clipRect**

The clipping rectangle, in port coordinates (MacIntosh).

**display**

The Display* of the window (for X11 in UNIX).
PixEffectInfo

Contains all of the information about the effect the RealPix renderer should perform. This structure is used in the IHXLiveRealPix::SendEffect method.

typedef struct HXEXPORT_CLASS _PixEffectInfo
{
    UINT32  m_ulStructLength;         /* In */
    BYTE    m_ucEffectType;           /* In */
    UINT32  m_ulStart;                /* In */
    UINT32  m_ulDuration;             /* In */
    UINT32  m_ulTarget;               /* In */
    UINT32  m_ulSrcX;                 /* In */
    UINT32  m_ulSrcY;                 /* In */
    UINT32  m_ulSrcW;                 /* In */
    UINT32  m_ulSrcH;                 /* In */
    UINT32  m_ulDstX;                 /* In */
    UINT32  m_ulDstY;                 /* In */
    UINT32  m_ulDstW;                 /* In */
    UINT32  m_ulDstH;                 /* In */
    UINT32  m_ulMaxFps;               /* In */
    BOOL    m_bAspectFlag;            /* In */
    BYTE    m_ucRed;                  /* In */
    BYTE    m_ucGreen;                /* In */
    BYTE    m_ucBlue;                 /* In */
    BYTE    m_ucWipeDirection;        /* In */
    BYTE    m_ucWipeType;             /* In */
    char   *m_pURL;                   /* In */
    char   *m_pExtPackage;            /* In */
    char   *m_pExtName;               /* In */
    char   *m_pExtData;               /* In */
    char   *m_pExtFile;               /* In */
    BOOL    m_bDiscardImage;          /* In */
    BOOL    m_bDisplayImmediately;    /* In */
    UINT32  m_ulHandle;               /* Out */
} _PixEffectInfo;

m_ulStructLength
The size of this structure.

m_ucEffectType
The effect type. This can be any of the following:

• EFFECT_FILL
    A fill effect.

• EFFECT_FADEIN
    A fade in effect.

• EFFECT_FADEOUT
    A fade out effect.
• EFFECT_CROSSFADE
  A cross-fade effect.
• EFFECT_WIPE
  A wipe effect.
• EFFECT_VIEWCHANGE
  A view change effect.
• EFFECT_EXTERNAL
  An effect provided by an external package.

m_ulStart
  Start time of the effect, in milliseconds, from when the encoder starts.

m_ulDuration
  The duration of the effect, in milliseconds.

m_ulTarget
  Handle of the image on which to perform the effect.

m_ulSrcX
  Horizontal offset of the source rectangle, in pixels. You can use this parameter and the three
  “source” parameters that follow to display only a portion of the original image. To display the
  entire image, set all of these source parameters to 0 (zero).

m_ulSrcY
  Vertical offset of the source rectangle, in pixels.

m_ulSrcW
  Width of the source rectangle, in pixels.

m_ulSrcH
  Height of the source rectangle, in pixels.

m_ulDstX
  Horizontal offset of the destination rectangle, in pixels. This parameter, and the three
  “destination” parameters that follow, position and size the image in the presentation space.

m_ulDstY
  Vertical offset of the destination rectangle, in pixels.

m_ulDstW
  Width of the destination rectangle, in pixels.

m_ulDstH
  Height of the destination rectangle, in pixels.

m_ulMaxFps
  Maximum frames per second for this effect.

m_bAspectFlag
  If TRUE, the aspect ratio on this effect is preserved. If FALSE, the aspect ratio is not preserved.

m_ucRed
  The red component of a fill or fadeout color.
m_ucGreen
The green component of a fill or fadeout color.

m_cuBlue
The blue component of a fill or fadeout color.

m_ucWipeDirection
The wipe direction. This can be any of the following:
• WIPE_DIRECTION_UP
  The wipe occurs starting from the bottom of the screen and moves to the top.
• WIPE_DIRECTION_DOWN
  The wipe occurs starting from the top of the screen and moves to the bottom.
• WIPE_DIRECTION_LEFT
  The wipe occurs starting from the right of the screen and moves left.
• WIPE_DIRECTION_RIGHT
  The wipe occurs starting from the left of the screen and moves right.

m_ucWipeType
The type of wipe. This can be any of the following:
• WIPE_TYPE_NORMAL
  A slide-over wipe.
• WIPE_TYPE_PUSH
  A push wipe.

m_pURL
Pointer to the URL to send the client to when the image is clicked.

m_pExtPackage
Pointer to the name of an external effect package.

m_pExtName
Pointer to the name of an external effect within the package.

m_pExtData
Pointer to the opaque string data for an external effect.

m_pExtFile
Pointer to the file contents of external effect data.

m_bDiscardImage
If TRUE, discard the image after doing effect. If FALSE, do not discard the image.

m_bDisplayImmediately
If TRUE, do the effect immediately. If FALSE, begin the effect at the time specified by m_ulStart.

m_ulHandle
Handle by which the effect can be referenced (returned by IHXLiveRealPix::SendEffect).
PixImageInfo

Contains all of the information for images that is sent down the stream. This structure is used in the IHXLiveRealPix::InitImage method.

typedef struct HXEXPORT_CLASS _PixImageInfo
{
    UINT32 m_ulStructLength; /* In */
    BYTE *m_pImageBuffer;   /* In */
    UINT32 m_ulImageBufferSize; /* In */
    char *m_pImageCodec;  /* In */
    UINT32 m_ulHandle; /* Out */
    UINT32 m_ulNumPackets; /* Out */
    UINT32 m_ulTimeToSend; /* Out */
} PixImageInfo;

m_ulStructLength
    The size of this structure.

m_pImageBuffer
    Pointer to the buffer holding the image data.

m_ulImageBufferSize
    The size of the image buffer pointed to by m_pImageBuffer.

m_pImageCodec
    Pointer to the image codec to be used.

m_ulHandle
    The image handle.

m_ulNumPackets
    The number of packets that make up this image.

m_ulTimeToSend
    The amount of time required to send the image, in milliseconds.

PixInitInfo

Contains initialization information and is passed in the IHXLiveRealPix::StartEncoder method. All of the members of this structure are input variables.

typedef struct HXEXPORT_CLASS _PixInitInfo
{
    UINT32 m_ulStructLength;
    char *m_pServerAddress;
    UINT32 m_ulServerPort;
    char *m_pUsername;
    char *m_pPassword;
    char *m_pFilename;
    char *m_pTitle;
    char *m_pAuthor;
    char *m_pCopyright;
} PixInitInfo;
APPENDIX B: Structure List

```
UINT32  m_ulBitrate;
UINT32  m_ulMaxFps;
UINT32  m_ulDisplayWidth;
UINT32  m_ulDisplayHeight;
BOOL    m_bPreserveAspect;
char   *m_pDefaultURL;
UINT32  m_ulNumImageCodecs;
char  **m_ppImageCodec;
UINT32  m_ulNumEffectPackages;
char  **m_ppEffectPackage;
```

PixInitInfo;

**m_ulStructLength**
The length of this structure, in bytes.

**m_pServerAddress**
Pointer to the host name or IP address of the server.

**m_ulServerPort**
The server port on which the broadcast plug-in listens. This parameter is defined by the plug-in's `FSMount` setting.

**m_pUsername**
Pointer to the name the application uses to connect to the broadcast plug-in. The name should be `encoder` if a password is used.

**m_pPassword**
Pointer to the password the application uses to connect to the broadcast plug-in. This parameter is defined by the plug-in's `FSMount` setting.

**m_pFilename**
Pointer to the “file name” that follows the broadcast plug-in's mount point in the URL. For example, in `rtsp://www.real.com/encoder/image`, the broadcast plug-in has a mount point of `/encoder/` and the “file name” is `image`.

**m_pTitle**
Pointer to the title of the live presentation.

**m_pAuthor**
Pointer to the name of the author or organization that created the live presentation.

**m_pCopyright**
Pointer to the copyright information on the live presentation.

**m_ulBitrate**
The bit rate of the RealPix stream, in kilobits per second.

**m_ulMaxFp**
The maximum number of effect frames to be sent per second.

**m_ulDisplayWidth**
The size of the presentation window width, in pixels.

**m_ulDisplayHeight**
The size of the presentation window height, in pixels.
m_bPreserveAspect
    If TRUE, the aspect ratio is preserved. If FALSE, the aspect ratio is not preserved.

m_pDefaultURL
    Pointer to the default URL to send the browser to when clicked.

m_ulNumImageCodecs
    The number of image codecs used in this stream.

m_ppImageCodec
    Returns a pointer to the string names of the codecs. The string names of codecs currently
    supported are “image/vnd.rn-realpix.jpeg” (for JPEG images) and “image/vnd.rn-realpix.gif” (for GIF
    images).

m_ulNumEffectPackages
    The number of external effect packages used.

m_ppEffectPackage
    Returns a pointer to the string names of third-party effect packages.

PPVAccessLog

Contains the user authentication access logging information for pay-per-view content.

For More Information: For information on user authentication, see the chapters on
authenticating Helix Universal Server visitors and storing authentication data in the
Helix Universal Server Administration Guide.

typedef struct _PPVAccessLog
{
    BOOL bAccessGranted;
    char* pUserid;
    char* pGUID;
    char* pIPAddress;
    char* pURL;
    PPVPermissionType nPermissionType;
    PPVAccessPermissionOn nPermOn;
    time_t tStartTime;
    time_t tStopTime;
    UINT32 lTotalTime;
    PPVAccessDisconnectType nWhyDisconnect;
} PPVAccessLog;

bAccessGranted
    If TRUE, access was granted. If FALSE, access was denied.

pUserid
    Pointer to the user ID that indicates which user accessed the content.

pGUID
    Pointer to the GUID of the user’s player that accessed the content. If the user elects to suppress
    this information, this parameter will contain a series of zeros (00000000-0000-0000-0000-
    000000000000) instead of a unique identifier. There is no way to override the client’s setting,
should the user choose to send only 0’s (zeros). Regardless of the user’s setting, Helix Universal Server can also be instructed to always show the string of zeros instead of the actual client identifier. If Helix Universal Server has this option set, all access log records show zeros, rather than the actual client identifiers. For more information, see the Helix Universal Server Administration Guide.

pIPAddress
Pointer to the IP address of the client that accessed the content.

pURL
Pointer to the URL that was accessed.

nPermissionType
A PPVPermissionType enumerator that describes the type of permission granted. This can be any of the following:
- PPV_PERMISSION_CREDIT
  The user can watch for any number of seconds and will be billed later.
- PPV_PERMISSION_DEBIT
  The user can watch x seconds of a clip.
- PPV_PERMISSION_EXPIRES
  The user can watch a clip any time before an expiration date.
- PPV_PERMISSION_GENERAL
  The user has access to watch a clip.
- PPV_PERMISSION_NONE
  No permission is granted.

pPermOn
A PPVAccessPermissionOn enumerator that indicates the kind of resource on which the user has permission. This can be any of the following:
- PPV_PERMISSION_ON_DIRECTORY
  The user has permission in a specified directory.
- PPV_PERMISSION_ON_FILE
  The user has permission on a specified file.
- PPV_PERMISSION_ON_NONE
  No permission is granted.

tStartTime
The time the pay-per-view session starts, indicated by the amount of time that has elapsed since January 1, 1970.

tStopTime
The time the pay-per-view session stops, indicated by the amount of time that has elapsed since January 1, 1970.
lTotalTime
The total amount of time allotted for the pay-per-view session.

nWhyDisconnect
A PPVAccessDisconnectType enumerator that describes the reason for disconnecting. This can be either of the following:

- PPV_DISCONNECT_CLIENT
  The client disconnected.
- PPV_DISCONNECT_TIME_EXPIRED
  The time, indicated by PPV_PERMISSION_EXPIRES or PPV_PERMISSION_DEBIT, has expired.

PPVPermission
Contains the pay-per-view permission information and URL.

typedef struct _PPVPermission
{
    char pURL[PPV_MAX_URL_LEN];
    PPVURLType nURLType;
    PPVPermissionType nPermissionType;
    time_t tExpires;
    UINT32 ulDebitTime;
} PPVPermission;

pURL
The pay-per-view URL.

nURLType
A PPVURLType enumerator that describes the type of URL. This can be either of the following:

- PPV_URL_TYPE_FILE
  The URL contains a file.
- PPV_URL_TYPE_DIRECTORY
  The URL contains a directory.

nPermissionType
A PPVPermissionType enumerator that describes the type of permission granted. This can be any of the following:

- PPV_PERMISSION_CREDIT
  The user can watch for any number of seconds and will be billed later.
- PPV_PERMISSION_DEBIT
  The user can watch x seconds of a clip.
- PPV_PERMISSION_EXPIRES
  The user can watch a clip any time before an expiration date.
- PPV_PERMISSION_GENERAL
The user has access to watch a clip.

- **PPV_PERMISSION_NONE**
  No permission is granted.

**tExpires**
The time the permission expires, indicated by the amount of time that has elapsed since January 1, 1970.

**ulDebitTime**
If debit permission was given, this is the amount of time given before permission will expire.

**PPVRegLog**
Contains the pay-per-view player validation logging information for the registration of the GUID.

**For More Information:** For information on player validation, see the chapters on authenticating Helix Universal Server visitors and storing authentication data in the *Helix Universal Server Administration Guide*.

```c
typedef struct _PPVRegLog {
    PPVRegStatus nStatus;
    char* pUserid;
    char* pGUID;
    char* pIPAddress;
    time_t tRequestTime;
    char* pURLRedirect;
} PPVRegLog;
```

**nStatus**
A PPVRegStatus enumerator that describes the status of the operation. This can be any of the following:

- **PPV_GUID_REG_SUCCESS**
The GUID was retrieved and successfully entered in the database.

- **PPV_GUID_REG_FAILED_LOCKED**
The entry was already registered in the database and the database is locked.

- **PPV_GUID_REG_FAILED_COLLISION**
Attempted to register a GUID that matches a GUID in the database with another user name.

- **PPV_GUID_REG_FAILED_OLD_PLAYER**
The player was too old to participate in this pay-per-view process (for example, the player had no GUID).

- **PPV_GUID_REG_FAILED_NO_USER**
No user by this name could be found.

- **PPV_GUID_REG_FAILED**
An unspecified failure occurred.
pUserId
  Pointer to the user ID that indicates which user accessed the content.

pGUID
  Pointer to the GUID of the user’s player that accessed the content. If the user elects to suppress this information, this parameter will contain a series of zeros (00000000-0000-0000-0000-000000000000) instead of a unique identifier. There is no way to override the client’s setting, should the user choose to send only zeros. Regardless of the user’s setting, Helix Universal Server can also be instructed to always show the string of zeros instead of the actual client identifier. If Helix Universal Server has this option set, all player validation log records show zeros, rather than the actual client identifiers. For more information, see the Helix Universal Server Administration Guide.

pIPAddress
  Pointer to the IP address of the client that accessed the content.

tRequestTime
  The start time requested, indicated by the amount of time that has elapsed since January 1, 1970.

pURLRedirect
  Pointer to a URL that provides extra content if the registration succeeds (for example, a congratulations message).
FUNCTION LIST

CloseEngine

Closes the engine that was returned in the CreateEngine function. This function is implemented by the Helix Architecture (HX) core.

STDAPI CloseEngine(
    IHXClientEngine* pEngine
);

pEngine
Pointer to an IHXClientEngine interface that manages the client engine to be closed.

CreateEngine

Returns a pointer to the client engine. This function is implemented by the Helix Architecture (HX) core and is run by top-level clients.

STDAPI CreateEngine(
    IHXClientEngine** ppEngine
);

ppEngine
Pointer to a pointer to an IHXClientEngine interface that manages the client engine being created.

CreateContext

Creates a Helix Architecture (HX) context for a remote broadcast application from the remote broadcast library. You can then query (IUnknown::QueryInterface) off of this context for any other interface you may require.

STDAPI_(IUnknown*) CreateContext( );

CreateLiveText

Creates an instance of the IHXLiveText interface and returns the pointer.

STDAPI CreateLiveText(
    IHXLiveText**
);

CreateLiveRealPix

Creates an instance of the IHXLiveRealPix interface and returns the pointer.
CreateLiveRealPix

This function is obsolete and should not be used in new applications.

HXCreateInstance

Creates an instance of any of the objects supported by the DLL. This function is similar to Window’s CoCreateInstance function in its purpose, except that this function only creates objects from this plug-in DLL. This function is implemented by all plug-in DLLs.

Note: Aggregation is never used. Therefore, an outer unknown is not passed to this function, and you do not need to write code for this situation.

HXShutdown

Frees any global resources. This function is called just before the DLL is unloaded. This function is implemented by all plug-in DLLs.

STDAPI CreateLiveRealPix(
    IHXLiveRealPix **
);

CreatePassword

STDAPI HXCreateInstance(
    IUnknown** ppIUnknown
);

ppIUnknown
    Pointer to a pointer to the object being created.

STDAPI HXShutdown(void);
This appendix describes all of the return values specifically used by the Helix (server and client). The return values used by the Helix Client and Server Software Development Kit interfaces are a subset of those contained in the hxresult.h file.

**HXR_OK**
- The operation completed successfully.

**HXR_ABORT**
- The user cancelled the process before it was finished.

**HXR_ACCESSDENIED**
- When you tried to access a local file, it was set in such a way that it could not be opened, deleted, or changed.

**HXR_ADVANCED_PROXY**
- You need to upgrade to a newer client to access this proxy.

**HXR_ADVANCED_SERVER**
- You need to upgrade to a newer version of the Helix client to access this server.

**HXR_ADVISE_PREFER_LINEAR**
- The file uses linear access instead of random access.

**HXR_ALREADY_OPEN**
- A file you are trying to access is already open.

**HXR_AT_END**
- The end of the file was already reached.

**HXR_AT_INTERRUPT**
- The process did not occur because an interrupt is currently in progress.

**HXR_AUDIO_DRIVER**
- The audio device cannot be opened, possibly because another application is using it.

**HXR_AUTOCFG_ABORT**
- The user cancelled the autoconfiguration process before it was finished.

**HXR_AUTOCFG_FAILED**
- The autoconfiguration process did not succeed.

**HXR_AUTOCFG_SUCCESS**
- The autoconfiguration process succeeded.
B

**HXR_BAD_FORMAT**
An unknown data format was specified.

**HXR_BAD_SERVER**
This server is not using a recognized protocol.

**HXR_BAD_TRANSPORT**
The transport was not recognized.

**HXR_BIND**
An error occurred binding to the network socket.

**HXR_BLOCK_CANCELED**
Indicates the blocking call was cancelled.

**HXR_BUFFERING**
The request is deferred until after buffering is complete.

C

**HXR_CANCELLED**
A pending operation was cancelled.

**HXR_CHUNK_MISSING**
While parsing a file, a critical structure was not found.

**HXR_CLASS_NOAGGREGATION**
This component does not support aggregation.

**HXR_CLOSED**
The action requested could not be performed because the resource (that is, the file) is already closed.

**HXR_COULDNOTINITCORE**
A failure occurred during the initialization of the player core.

D

**HXR_DEC_DECOMPRESS**
The Helix client was unable to decompress this content.

**HXR_DEC_INIT_FAILED**
The requested Helix client decoder cannot be found or cannot be used on this computer.

**HXR_DEC_INITED**
The requested Helix client decoder was found and initialized.

**HXR_DEC_INVALID**
The requested Helix client decoder is not valid.

**HXR_DEC_NOT_FOUND**
File compression is not supported because the requested Helix client decoder cannot be located.

**HXR_DEC_NOT_INITED**
The Helix client decoder was not initialized before you attempted to use it.
APPENDIX D: Return Values

HXR_DEC_TYPE_MISMATCH
The requested decoder cannot be loaded because of a decoder type mismatch.

HXR_DNR
The server could not be located. This server does not have a DNS entry. Check the server name in
the URL and try again.

HXR_DOC_MISSING
The requested file could not be found. The link you followed may be outdated or inaccurate.

E

HXR_ELEMENT_NOT_FOUND
An element was not found.

HXR_EXPIRED
The product you are using has expired. You can upgrade to the latest version by visiting

F

HXR_FAIL
An unspecified, general error occurred.

HXR_FILE_EXISTS
The file you were trying to create already exists.

HXR_FILE_NOT_FOUND
The file you were trying to open does not exist.

HXR_FILE_NOT_OPEN
The file to which you were trying to read or write is not open.

HXR_FORCE_PERFECTPLAY
Force PerfectPlay for this presentation.

G

HXR_GENERAL_MULTICAST
An error occurred accessing a multicast session.

HXR_GENERAL_NONET
The specified winsock services cannot be found. Remove old Internet software from your system
and try again.

H

HXR_HTTP_CONNECT
Could not connect to the server using HTTP.

I

HXR_INVALID_BANDWIDTH
The bandwidth you specified was not in the range allowed.

HXR_INVALID_FILE
The file type specified cannot be played by the Helix client.
HXR_INVALID_HOST
A connection with the server could not be established.

HXR_INVALID_HTTP_PROXY_HOST
An invalid hostname for the HTTP proxy was given.

HXR_INVALID_INTERLEAVER
The deinterleaving process necessary for this stream is not present.

HXR_INVALID_METAFILE
An invalid Metafile was given.

HXR_INVALID_OPERATION
The requested operation is not valid, therefore the request could not be processed.

HXR_INVALID_PARAMETER
An invalid parameter parameter was used, therefore the request could not be processed.

HXR_INVALID_PATH
The requested URL is not valid.

HXR_INVALID_PROTOCOL
An invalid protocol was specified in the URL. URLs should typically start with "rtsp://", "pnm://", or "http://".

HXR_INVALID_REVISION
An invalid RealMedia file version was specified.

HXR_INVALID_STREAM
An invalid RealMedia stream was specified.

HXR_INVALID_URL_HOST
An invalid host string was specified in the requested URL.

HXR_INVALID_URL_OPTION
An invalid option was specified in the URL.

HXR_INVALID_URL_PATH
An invalid resource path string was specified in the requested URL.

HXR_INVALID_VERSION
An invalid file version number was given.

L

HXR_LATE_PACKET
The packet arrived too late to be processed.

M

HXR_MISSING_COMPONENTS
Some components are not available to provide playback of this presentation on your system.

HXR_MULTICAST_JOIN
An error occurred attempting to join a multicast session.
**HXR_MULTICAST_UDP**
Audio data cannot be received from this multicast session. You may want to try the TCP data option in the Network Preferences. For more information, contact your system administrator.

**HXR_NET_CONNECT**
A connection to server could not be established. You may be experiencing network problems.

**HXR_NET_READ**
An error occurred while reading data from the network.

**HXR_NET_SOCKET_INVALID**
An invalid socket error occurred. If this value is seen on the encoder, the RTSP transport layer probably failed in an attempt to write data to the operating system-specific socket. If this value is seen on the server, a read was probably issued on a socket while a current read was pending.

**HXR_NET_TCP**
TCP data packets cannot be received. You may want to try the HTTP cloaking option in the Network Preferences. You may also want to configure the Helix client to use a firewall proxy. For more information, contact your system administrator.

**HXR_NET_UDP**
UDP data packets cannot be received. You may want to try the TCP data option in the Network Preferences. You may also want to configure the Helix client to use a firewall proxy. For more information, contact your system administrator.

**HXR_NET_WRITE**
An error occurred while writing data to the network.

**HXR_NO_CODECS**
No codecs have been installed on your system.

**HXR_NO_DATA**
No more data was available.

**HXR_NO_FILEFORMAT**
The requested file format component is missing.

**HXR_NO_FILESYSTEM**
The requested file system was not found.

**HXR_NO_LIVE_PERFECTPLAY**
PerfectPlay is not supported for live streams.

**HXR_NO_RENDERER**
The renderer component is missing.

**HXR_NOSEEK**
A seek operation is not possible. For example, you cannot perform a seek operation on a live presentation.

**HXR_NO_SESSION_ID**
The RTSP session is not valid.
HXR_NOINTERFACE
The creation function did not recognize the requested interface.

HXR_NONCONTIGUOUS_PACKET
The packet was not received in the proper order.

HXR_NOT_AUTHORIZED
Access was denied because the request was not authorized.

HXR_NOT_INITIALIZED
The component has not been initialized.

HXR_NOT_LICENSED
The operation being attempted was not licensed.

HXR_NOT_NOTIFIED
Indicates the client did not request to recieve the notification.

HXR_NOTENOUGH_BANDWIDTH
You cannot receive this content because you do not have enough network bandwidth.

HXR_NOTIMPL
The function is not implemented.

O

HXR_OLD_PROXY
The connection was closed because the proxy is too old for this client.

HXR_OLD_SERVER
The connection was closed because the host’s version of the RealNetworks server is too old for this client.

HXR_OPEN_DRIVER
The network drivers cannot be opened.

HXR_OPEN_NOT_PROCESSED
The open command has not been processed (probably because too many connections are already open).

HXR_OUTOFMEMORY
The function failed to allocate the necessary memory. You may need to close some other applications to play this content.

HXR_OUTOFORDER_PACKET
The packet could not be written because of out of order time stamps.

HXR_OVERLAPPED_PACKET
The packet could not be written because of overlapping time stamps.

P

HXR_PAUSED
The clip has been paused.

HXR_PERFECTPLAY_NOT_ALLOWED
PerfectPlay was not allowed on this clip.
APPENDIX D: Return Values

**HXR_PERFECTPLAY_NOT_SUPPORTED**
The requested server does not support PerfectPlay.

**HXR_POINTER**
A NULL pointer was passed to the method.

**HXR_PORT_IN_USE**
The RTSP port was already in use and therefore could not be opened.

**HXR_PPV_ACCOUNT_LOCKED**
A client was refused access to protected content because another use is already logged into the server with the same user name.

**HXR_PPV_AUTHORIZATION_FAILED**
An invalid password was given.

**HXR_PPV_GUID_COLLISION**
Another user has already registered the requested GUID.

**HXR_PPV_GUID_READ_ONLY**
A GUID has already been registered for the specified user name.

**HXR_PPV_NO_USER**
The requested user name does not exist in the pay-per-view database.

**HXR_PPV_OLD_PLAYER**
The connected player is unable to participate in GUID-based authentication because it is too old.

**HXR_PROP_ACTIVE**
The registry property could not be modified because it is an active property. Use instead of IHXRegistry to modify this entry.

**HXR_PROP_DUPLICATE**
The property already exists.

**HXR_PROP_INACTIVE**
The property is not active.

**HXR_PROP_NOT_COMPOSITE**
You were trying to perform a composite action on a noncomposite key.

**HXR_PROP_NOT_FOUND**
The property was not found.

**HXR_PROP_TYPE_MISMATCH**
The method you used does not match the property type.

**HXR_PROXY**
A proxy status error occurred.

**HXR_PROXY_DNR**
The proxy server could not be located. This proxy server does not have a DNS entry. Check the proxy server name and try again.

**HXR_PROXY_NET_CONNECT**
The connection to the proxy server could not be established. You may be experiencing network problems.
HXR_PROXY_RESPONSE
A proxy invalid response error occurred.

R
HXR_REGISTER_GUID_EXISTS
Another user has already registered the requested GUID.

S
HXR_SEEK_PENDING
A seek was issued while another seek was already pending.

HXR_SERVER_ALERT
A server alert occurred.

HXR_SERVER_DISCONNECTED
Connection to server has been lost. You may be experiencing network problems.

HXR_SERVER_TIMEOUT
Connection to the server has timed out. You may be experiencing network problems.

HXR_SLOW_MACHINE
Your CPU is unable to decode this content in real time. Try closing other applications, or setting
your bandwidth preference lower to receive less complex content.

HXR_SOCKET_CREATE
An error occurred while creating a network socket.

HXR_STOPPED
The end of the clip has been reached, or the clip has been terminated.

HXR_STREAM_DONE
The specified stream has finished streaming.

T
HXR_TRY_AUTOCONFIG
Could not connect to the server. Check network transport settings or run auto-configure.

U
HXR_UNEXPECTED
The operation you specified was not expected. For example, this value would be returned if you
called a method in an interface that required initialization, but you had not performed that
intialization before calling this method.

HXR_UNKNOWN_EFFECT
The number of the effect is not valid.

HXR_UNKNOWN_IMAGE
The number of the image is not valid.

HXR_UNSUPPORTED_AUDIO
The file contains an unsupported audio format. The needed codec is not installed on your system.

HXR_UNSUPPORTED_VIDEO
The file contains an unsupported video format. The needed codec is not installed on your system.
HXR_UPG_AUTH_FAILED
Download of the AutoUpdate component failed because it could not be verified as an authentic update distributed by RealNetworks.

HXR_UPG_CERT_AUTH_FAILED
The certificate portion of the AutoUpdate package (RUP file) is damaged.

HXR_UPG_CERT_EXPIRED
The certificate of the AutoUpdate package (RUP file) is expired.

HXR_UPG_CERT_REVOKED
The certificate of the AutoUpdate package (RUP file) is revoked.

HXR_UPG_RUP_BAD
The RUP file structure is invalid.

HXR_VIEW_SOURCE_DISSABLED
The provider of this content has not granted permission to view the source.

HXR_VIEW_SOURCE_NOCLIP
No clip is available.

HXR_WOULD_BLOCK
The nonblocking call was not able to complete.

HXR_WRITE_ERROR
An error occurred when you tried to perform a write.
The Helix architecture supports RealMedia File Format (RMFF), which enables Helix to deliver high-quality multimedia content over a variety of network bandwidths. Third-party developers can convert their media formats into RMFF, enabling Helix Universal Server to deliver the files to RealPlayer or other applications built with the Helix Client and Server Software Development Kit. Third-party developers can thereby use Helix to transport content over the Internet to their own applications.

RealMedia File Format is a standard tagged file format that uses four-character codes to identify file elements. These codes are 32-bit, represented by a sequence of one to four ASCII alphanumeric characters, padded on the right with space characters. The data type for four-character codes is FOURCC. Use the `HX_FOURCC` macro to convert four characters into a four-character code.

The basic building block of a RealMedia File is a chunk, which is a logical unit of data, such as a stream header or a packet of data. Each chunk contains the following fields:

- Four-character code specifying the chunk identifier
- 32-bit value specifying the size of the data member in the chunk
- Blob of opaque chunk data

Depending on its type, a top-level chunk can contain subobjects. This document describes the tagged chunks contained in RMFF, as well as the format of the data stored in each type of tagged chunk.

RealMedia File Format organizes tagged chunks into a header section, a data section, and an index section. The organization of these tagged chunks is shown in the following figure.
Sections of a RealMedia File

<table>
<thead>
<tr>
<th>Sections</th>
<th>Headers</th>
</tr>
</thead>
<tbody>
<tr>
<td>RealMedia File Header</td>
<td>Properties, Media Properties, Content Description</td>
</tr>
<tr>
<td></td>
<td>Data Chunk Header, Data Packets</td>
</tr>
<tr>
<td></td>
<td>Index</td>
</tr>
</tbody>
</table>

**Header Section**

Because RMFF is a tagged file format, the order of the chunks is not explicit, except that the RealMedia File Header must be the first chunk in the file. However, most applications write the standard headers into the file's header section. The following chunks are typically found in the header section of RMFF:

- RealMedia File Header (this must be the first chunk of the file)
- Properties Header
- Media Properties Header
- Content Description Header

After the RealMedia File Header object, the other headers may appear in any order. All headers are required except the Index Header. The following sections describe the individual header objects.

**RealMedia File Header**

Each RealMedia file begins with the RealMedia File Header, which identifies the file as RMFF. There is only one RealMedia File Header in a RealMedia file. Because the contents of the RealMedia File Header may change with different versions of RMFF, the header structure supports an object version field for determining what additional fields exists. The following pseudo-structure describes the RealMedia File Header:

```c
RealMedia_File_Header
{
    UINT32    object_id;
    UINT32    size;
    UINT16    object_version;
```
if ((object_version == 0) || (object_version == 1))
{
    UINT32 file_version;
    UINT32 num_headers;
}
}

The RealMedia File Header contains the following fields:

**object_id**
- The unique object ID for a RealMedia File (.RMF). All RealMedia files begin with this identifier. The size of this member is 32 bits.

**size**
- The size of the RealMedia header section in bytes. The size of this member is 32 bits.

**object_version**
- The version of the RealMedia File Header object. All files created according to this specification have an `object_version` number of 0 (zero) or 1. The size of this member is 16 bits.

**file_version**
- The version of the RealMedia file. The Helix Client and Server SDK only covers files with a file version of either 0 (zero) or 1. This member is present on all `RealMedia_File_Header` objects with an `object_version` of 0 (zero) or 1. The size of this member is 32 bits.

**num_headers**
- The number of headers in the header section that follow the RealMedia File Header. This member is present on all `RealMedia_File_Header` objects with an `object_version` of 0 (zero) or 1. The size of this member is 32 bits.

### Properties Header

The Properties Header describes the general media properties of the RealMedia File. Components of the RealMedia system use this object to configure themselves for handling the data in the RealMedia file or stream. There is only one Properties Header in a RealMedia file. The following pseudo-structure describes the Properties header:

Properties
{
    UINT32 object_id;
    UINT32 size;
    UINT16 object_version;

    if (object_version == 0)
    {
        UINT32 max_bit_rate;
        UINT32 avg_bit_rate;
        UINT32 max_packet_size;
        UINT32 avg_packet_size;
        UINT32 num_packets;
        UINT32 duration;
        UINT32 preroll;
    }
The Properties Header contains the following fields:

**object_id**

The unique object ID for a Properties Header ('PROP'). The size of this member is 32 bits.

**size**

The 32-bit size of the Properties Header in bytes. The size of this member is 32 bits.

**object_version**

The version of the RealMedia File Header object. All files created according to this specification have an object_version number of 0 (zero). The size of this member is 16 bits.

**max_bit_rate**

The maximum bit rate required to deliver this file over a network. This member is present on all Properties objects with an object_version of 0 (zero). The size of this member is 32 bits.

**avg_bit_rate**

The average bit rate required to deliver this file over a network. This member is present on all Properties objects with an object_version of 0 (zero). The size of this member is 32 bits.

**max_packet_size**

The largest packet size (in bytes) in the media data. This member is present on all Properties objects with an object_version of 0 (zero). The size of this member is 32 bits.

**avg_packet_size**

The average packet size (in bytes) in the media data. This member is present on all Properties objects with an object_version of 0 (zero). The size of this member is 32 bits.

**num_packets**

The number of packets in the media data. This member is present on all Properties objects with an object_version of 0 (zero). The size of this member is 32 bits.

**duration**

The duration of the file in milliseconds. This member is present on all Properties objects with an object_version of 0 (zero). The size of this member is 32 bits.

**preroll**

The number of milliseconds to prebuffer before starting playback. This member is present on all Properties objects with an object_version of 0 (zero). The size of this member is 32 bits.

**index_offset**

The offset in bytes from the start of the file to the start of the index header object. This value can be 0 (zero), which indicates that no index chunks are present in this file. This member is present on all Properties objects with an object_version of 0 (zero). The size of this member is 32 bits.
**data_offset**

The offset in bytes from the start of the file to the start of the Data Section. This member is present on all Properties objects with an `object_version` of 0 (zero). The size of this member is 32 bits.

*Note:* There can be a number of Data_Chunk_Headers in a RealMedia file. The `data_offset` value specifies the offset in bytes to the first Data_Chunk_Header. The offsets to the other Data_Chunk_Headers can be derived from the `next_data_header` field in a Data_Chunk_Header.

**num_streams**

The total number of media properties headers in the main headers section. This member is present on all Properties objects with an `object_version` of 0 (zero). The size of this member is 16 bits.

**flags**

Bit mask containing information about this file. The following bits carry information—all of the rest should be zero:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Flag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Save_Enabled</td>
<td>If 1, clients are allowed to save this file to disk.</td>
</tr>
<tr>
<td>1</td>
<td>Perfect_Play</td>
<td>If 1, clients are instructed to use extra buffering.</td>
</tr>
<tr>
<td>2</td>
<td>Live</td>
<td>If 1, these streams are from a live broadcast.</td>
</tr>
</tbody>
</table>

The size of this member is 16 bits.

**Media Properties Header**

The Media Properties Header describes the specific media properties of each stream in a RealMedia file. Components of the RealMedia system use this object to configure themselves for handling the media data in each stream. There is one Media Properties Header for each media stream in a RealMedia file. The following pseudo-structure describes the Media Properties header:

```c
Media_Properties
{
    UINT32     object_id;
    UINT32     size;
    UINT16     object_version;

    if (object_version == 0)
    {
        UINT16                      stream_number;
        UINT32                      max_bit_rate;
        UINT32                      avg_bit_rate;
        UINT32                      max_packet_size;
        UINT32                      avg_packet_size;
        UINT32                      start_time;
        UINT32                      preroll;
        UINT32                      duration;
        UINT8                       stream_name_size;
    }
}
```
The Media Properties Header contains the following members:

- **object_id**
  The unique object ID for a Media Properties Header ("MDPR"). The size of this member is 32 bits.

- **size**
  The size of the Media Properties Header in bytes. The size of this member is 32 bits.

- **object_version**
  The version of the Media Properties Header object. The size of this member is 16 bits.

- **stream_number**
  The **stream_number** (synchronization source identifier) is a unique value that identifies a physical stream. Every data packet that belongs to a physical stream contains the same **STREAM_NUMBER**. The **STREAM_NUMBER** enables a receiver of multiple physical streams to distinguish which packets belong to each physical stream. This member is present on all MediaProperties objects with an **object_version** of 0 (zero). The size of this member is 32 bits.

- **max_bit_rate**
  The maximum bit rate required to deliver this stream over a network. This member is present on all MediaProperties objects with an **object_version** of 0 (zero). The size of this member is 32 bits.

- **avg_bit_rate**
  The average bit rate required to deliver this stream over a network. This member is present on all MediaProperties objects with an **object_version** of 0 (zero). The size of this member is 32 bits.

- **max_packet_size**
  The largest packet size (in bytes) in the stream of media data. This member is present on all MediaProperties objects with an **object_version** of 0 (zero). The size of this member is 32 bits.

- **avg_packet_size**
  The average packet size (in bytes) in the stream of media data. This member is present on all MediaProperties objects with an **object_version** of 0 (zero). The size of this member is 32 bits.

- **start_time**
  The time offset in milliseconds to add to the time stamp of each packet in a physical stream. This member is present on all MediaProperties objects with an **object_version** of 0 (zero). The size of this member is 32 bits.

- **preroll**
  The time offset in milliseconds to subtract from the time stamp of each packet in a physical stream. This member is present on all MediaProperties objects with an **object_version** of 0 (zero). The size of this member is 32 bits.
duration
The duration of the stream in milliseconds. This member is present on all MediaProperties objects with an object_version of 0 (zero). The size of this member is 32 bits.

stream_name_size
The length of the following stream_name member in bytes. This member is present on all MediaProperties objects with an object_version of 0 (zero). The size of this member is 8 bits.

stream_name
A nonunique alias or name for the stream. This member is present on all MediaProperties objects with an object_version of 0 (zero). This size of this member is variable.

mime_type_size
The length of the following mime_type field in bytes. This member is present on all MediaProperties objects with an object_version of 0 (zero). This size of this member is 8 bits.

mime_type
A nonunique MIME style type/subtype string for data associated with the stream. This member is present on all MediaProperties objects with an object_version of 0 (zero). This size of this member is variable.

type_specific_len
The length of the following type_specific_data in bytes. The type_specific_data is typically used by the data type renderer to initialize itself in order to process the physical stream. This member is present on all MediaProperties objects with an object_version of 0 (zero). The size of this member is 32 bits.

type_specific_data
The type_specific_data is typically used by the data type renderer to initialize itself in order to process the physical stream. This member is present on all MediaProperties objects with an object_version of 0 (zero). The size of this member is variable.

Logical Stream Organization
A RealMedia file can contain a higher-level grouping of physical streams. This grouping is called a logical stream. Logical streams contain the following information:

• Identifies which physical streams are grouped together into a logical stream.
• Contains name value properties that can be used to idnetify properties of the logical stream. (such as language, packet grouping, and so on.)

A logical stream is represented with a Media Properties Header. The mime type of the physical stream is preceeded with “logical-“. For example, the mime type for an ASM-compatible RealAudio stream is audio/x-pn-multirate-realaudio. A logical stream consisting of a set of RealAudio physical streams would therefore have the mime type logical-audio/x-pn-multirate-realaudio. An example of a logical stream is shown in the following figure.
Logical Stream Organization

In this example there is one logical stream, one low bit rate audio stream and one high bit rate audio stream. This results in a RealMedia file with three Media Property Headers and three data sections. The type_specific_data field of the logical stream’s Media Property Header contains a LogicalStream structure. This structure contains all of the information required to interpret the logical stream and its collection of physical streams. The structure refers to the low bit rate and high bit rate audio streams. The LogicalStream structure also contains the data_offsets to the start of the data section for each physical stream.

The logical stream number assigned to the logical stream is determined from the stream_number field in the Media Properties Header.

There is also one special logical stream of MIME type “logical-fileinfo” containing information about the entire file. There should only be one media header with this type. Behavior of players and editing tools is undefined if you have more than one.

The ASM rules contained in the logical-fileinfo stream are used to define precisely how bandwidth will be divided between the streams in the file. The logical-fileinfo may also contain a name-value pair that specifies which stream combinations should be served to older players.

LogicalStream Structure

The following sample shows the LogicalStream structure:

```
LogicalStream
{
    ULONG32       size;
    UINT16         object_version;

    if (object_version == 0)
    {
        UINT16 num_physical_streams;
        UINT16 physical_stream_numbers[num_physical_streams];
    }
```
The LogicalStream structure contains the following fields:

**size**
The size of the LogicalStream structure in bytes. The size of this structure member is 32 bits.

**object_version**
The version of the LogicalStream structure. The size of this structure member is 16 bits.

**num_physical_streams**
The number of physical streams that make up this logical stream. The physical stream numbers are stored in a list immediately following this field. These physical stream numbers refer to the stream_number field found in the Media Properties Object for each physical stream belonging to this logical stream. The size of this structure member is 16 bits.

**physical_stream_numbers[]**
The list of physical stream numbers that comprise this logical stream. The size of this structure member is variable.

**data_offsets[]**
The list of data offsets indicating the start of the data section for each physical stream. The size of this structure member is variable.

**num_rules**
The number of ASM rules for the logical stream. Each physical stream in the logical stream has at least one ASM rule associated with it or it will never get played. The mapping of ASM rule numbers to physical stream numbers is stored in a list immediately following this member. These physical stream numbers refer to the stream_number field found in the Media Properties Object for each physical stream belonging to this logical stream. The size of this structure member is 16 bits.

**rule_to_physical_stream_map[]**
The list of physical stream numbers that map to each rule. Each entry in the map corresponds to a 0-based rule number. The value in each entry is set to the physical stream number for the rule. For example:

```
rule_to_physical_stream_map[0] = 5
```

This example means physical stream 5 corresponds to rule 0. All of the ASM rules referenced by this array are stored in the first name-value pair of this logical stream which must be called “ASMRuleBook” and be of type “string”. Each rule is separated by a semicolon.

The size of this structure member is variable.

**num_properties**
The number of NameValueProperty structures contained in this structure. These name/value structures can be used to identify properties of this logical stream (for example, language). The size of this structure member is 16 bits.
properties[]

The list of NameValueProperty structures (see NameValueProperty Structure below for more details). As mentioned above, it is required that the first name-value pair be a string named “ASMRuleBook” and contain the ASM rules for this logical stream. The size of this structure member is variable.

NameValueProperty Structure

The following sample shows the NameValueProperty structure:

```c
NameValueProperty
{
    ULONG32    size;
    UINT16     object_version;

    if (object_version == 0)
    {
        UINT8    name_length;
        UINT8    name[name_length];
        INT32    type;
        UINT16   value_length;
        UINT8    value_data[value_length];
    }
}
```

The NameValueProperty structure contains the following fields:

size
The size of the NameValueProperty structure in bytes. The size of this structure member is 32 bits.

object_version
The version of the NameValueProperty structure. The size of this structure member is 16 bits.

name_length
The length of the name data. The size of this structure member is 8 bits.

name[]
The name string data. The size of this structure member is 8 bits.

type
The type of the value data. This member can take on one of three values (any other value is undefined), as shown in the following table:

<table>
<thead>
<tr>
<th>type</th>
<th>Description</th>
<th>value_length</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>32-bit unsigned integer property</td>
<td>4</td>
</tr>
<tr>
<td>1</td>
<td>buffer</td>
<td>variable</td>
</tr>
<tr>
<td>2</td>
<td>string</td>
<td>variable</td>
</tr>
</tbody>
</table>

The size of this structure member is 32 bits.

value_length
The length of the value data. The size of this structure member is 16 bits.
value_data[]

The value data. The size of this structure member is 8 bits.

**Content Description Header**

The Content Description Header contains the title, author, copyright, and comments information for the RealMedia file. All text data is in ASCII format. The following pseudo-structure describes the Content Description Header:

```c
Content_Description
{
    UINT32    object_id;
    UINT32    size;
    UINT16    object_version;

    if (object_version == 0)
    {
        UINT16    title_len;
        UINT8[title_len]  title;
        UINT16    author_len;
        UINT8[author_len]  author;
        UINT16    copyright_len;
        UINT8[copyright_len]  copyright;
        UINT16    comment_len;
        UINT8[comment_len]  comment;
    }
}
```

The Content Description Header contains the following fields:

**object_id**

The unique object ID for the Content Description Header (“CONT”). The size of this member is 32 bits.

**size**

The size of the Content Description Header in bytes. The size of this member is 32 bits.

**object_version**

The version of the Content Description Header object. The size of this member is 16 bits.

**title_len**

The length of the title data in bytes. Note that the title data is not null-terminated. This member is present on all Content Description Header objects with an object_version of 0 (zero). The size of this member is 16 bits.

**title**

An array of ASCII characters that represents the title information for the RealMedia file. This member is present on all Content Description Header objects with an object_version of 0 (zero). The size of this member is variable.
**author_len**
The length of the author data in bytes. Note that the author data is not null-terminated. This member is present on all Content Description Header objects with an `object_version` of 0 (zero). The size of this member is 16 bits.

**author**
An array of ASCII characters that represents the author information for the RealMedia file. This member is present on all Content Description Header objects with an `object_version` of 0 (zero). The size of this member is variable.

**copyright_len**
The length of the copyright data in bytes. Note that the copyright data is not null-terminated. This member is present on all Content Description Header objects with an `object_version` of 0 (zero). The size of this member is 16 bits.

**copyright**
An array of ASCII characters that represents the copyright information for the RealMedia file. This member is present on all Content Description Header objects with an `object_version` of 0 (zero). The size of this member is variable.

**comment_len**
The length of the comment data in bytes. Note that the comment data is not null-terminated. This member is present on all Content Description Header objects with an `object_version` of 0 (zero). The size of this member is 16 bits.

**comment**
An array of ASCII characters that represents the comment information for the RealMedia file. This member is present on all Content Description Header objects with an `object_version` of 0 (zero). The size of this member is variable.

### Data Section

The data section of the RealMedia file consists of a Data Section Header that describes the contents of the data section, followed by a series of interleaved media data packets. Note that the size field of the Data Chunk Header is the size of the entire data chunk, including the media data packets.

#### Data Chunk Header

The Data Chunk Header marks the start of the data chunk. There is usually only one data chunk in a RealMedia file; however, for extremely large files, there may be multiple data chunks. The following pseudostructure describes the Data chunk header:

```
Data_Chunk_Header
{
    UINT32     object_id;
    UINT32     size;
    UINT16      object_version;

    if (object_version == 0)
    {
```
APPENDIX E: RealMedia File Format (RMFF) Reference

```c
UINT32 num_packets;
UINT32 next_data_header;
}
}
```

The Data Chunk Header contains the following fields:

- **object_id**
  The unique object ID for the Data Chunk Header (‘DATA’). The size of this member is 32 bits.

- **size**
  The size of the Data Chunk in bytes. The size includes the size of the header plus the size of all the packets in the data chunk. The size of this member is 32 bits.

- **object_version**
  The version of the Data Chunk Header object. The size of this member is 16 bits.

- **num_packets**
  Number of packets in the data chunk. This member is present on all Data Chunk Header objects with an `object_version` of 0 (zero). The size of this member is 32 bits.

- **next_data_header**
  Offset from start of file to the next data chunk. A non-zero value refers to the file offset of the next data chunk. A value of zero means there are no more data chunks in this file. This field is not typically used. This member is present on all Data Chunk Header objects with an `object_version` of 0 (zero). The size of this member is 32 bits.

### Data Packet Header

Following a data chunk header is `num_packets` data packets. These packets can all be from the same stream, or packets from different streams can follow one another. These packets, whether from the same stream or from different streams, should have an increasing value of timestamp. That is, the timestamp of a packet should be greater than or equal to the timestamp of the previous packet in the file.

The following pseudo-structure describes the details of the packet:

```c
Media_Packet_Header
{
    UINT16 object_version;
    
    if ((object_version == 0) || (object_version == 1))
    {
        UINT16 length;
        UINT16 stream_number;
        UINT32 timestamp;
        if (object_version == 0)
        {
            UINT8 packet_group;
            UINT8 flags;
        }
        else if (object_version == 1)
        {
```


```c
UINT16        asm_rule;
UINT8          asm_flags;
}

UINT8[length]        data;
}
else
{
    StreamDone();
}
}

The Media Packet Header contains the following fields:

**object_version**

The version of the Media Packet Header object. The size of this member is 16 bits.

**length**

The length of the packet in bytes. This member is present on all Media Packet Header objects with an object_version of 0 (zero) or 1. The size of this member is 16 bits.

**stream_number**

The 16-bit alias used to associate data packets with their associated Media Properties Header. This member is present on all Media Packet Header objects with an object_version of 0 (zero) or 1. The size of this member is 16 bits.

**timeStamp**

The time stamp of the packet in milliseconds This member is present on all Media Packet Header objects with an object_version of 0 (zero) or 1. The size of this member is 32 bits.

**packet_group**

The packet group to which the packet belongs. If packet grouping is not used, set this field to 0 (zero). This member is present on all Media Packet Header objects with an object_version of 0 (zero). The size of this member is 8 bits.

**flags**

Flags describing the properties of the packet. The following flags are defined:

- HX_RELIABLE_FLAG
  
  If this flag is set, the packet is delivered reliably.

- HX_KEYFRAME_FLAG
  
  If this flag is set, the packet is part of a key frame or in some way marks a boundary in your data stream.

This member is present on all Media Packet Header objects with an object_version of 0 (zero). The size of this member is 8 bits.

**asm_rule**

The ASM rule assigned to this packet. Only present if object_version equals 1. The size of this member is 16 bits.
asm_flags
Contains HX_flags that dictate stream switching points. Only present if object_version equals 1. The size of this member is 8 bits.

data
The application-specific media data. This member is present on all Media Packet Header objects with an object_version of 0 (zero) or 1. The size of this member is variable.

Index Section
The index section of the RealMedia file consists of an Index Chunk Header that describes the contents of the index section, followed by a series of index records. Note that the size field of the Index Chunk Header is the size of the entire index chunk, including the index records.

Index Section Header
The Index Chunk Header marks the start of the index chunk. There is usually one index chunk per stream in a RealMedia file. The following pseudo-structure describes the Index chunk header.

Index_Chunk_Header
{
    u_int32 object_id;
    u_int32 size;
    u_int16 object_version;

    if (object_version == 0)
    {
        u_int32 num_indices;
        u_int16 stream_number;
        u_int32 next_index_header;
    }
}

The Index Chunk Header contains the following fields:

object_id
The unique object ID for the Index Chunk Header (“INDX”). The size of this member is 32 bits.

size
The size of the Index Chunk in bytes. The size of this member is 32 bits.

object_version
The version of the Index Chunk Header object. The size of this member is 16 bits.

num_indices
Number of index records in the index chunk. This member is present on all Index Chunk Header objects with an object_version of 0 (zero). The size of this member is 32 bits.

stream_number
The stream number for which the index records in this index chunk are associated. This member is present on all Index Chunk Header objects with an object_version of 0 (zero). The size of this member is 16 bits.
**next_index_header**
- Offset from start of file to the next index chunk. This member enables RealMedia file format readers to find all the index chunks quickly. A value of zero for this member indicates there are no more index headers in this file. This member is present on all Index Chunk Header objects with an object_version of 0 (zero). The size of this member is 32 bits.

**Index Record**
- The index section of a RealMedia file consists of a series of index record objects. Each index record contains information for quickly finding a packet of a particular time stamp for a physical stream. The following pseudo-structure describes the details of each index record:

```
IndexRecord
{
    UINT16    object_version;
    if (object_version == 0)
    {
        u_int32  timestamp;
        u_int32  offset;
        u_int32   packet_count_for_this_packet;
    }
}
```

An Index Record contains the following fields:

- **object_version**
  - The version of the Index Record object. The size of this member is 16 bits.

- **timestamp**
  - The time stamp (in milliseconds) associated with this record. This member is present on all Index Record objects with an object_version of 0 (zero). The size of this member is 32 bits.

- **offset**
  - The offset from the start of the file at which this packet can be found. This member is present on all Index Record objects with an object_version of 0 (zero). The size of this member is 32 bits.

- **packet_count_for_this_packet**
  - The packet number of the packet for this record. This is the same number of packets that would have been seen had the file been played from the beginning to this point. This member is present on all Index Record objects with an object_version of 0 (zero). The size of this member is 32 bits.

**Metadata Section**
- The metadata section of the RealMedia file consists of a tag containing a set of named metadata properties that describe the media file. These properties can be text, integers, or any binary data. The tag is preceded by a header that identifies the size of the entire metadata section. Following the tag, the footer identifies the size of the tag. Since the metadata section is found at the end of the file, the footer can be used to expedite seeking backwards. At the end of the metadata section, and the file itself, is an ID3v1 tag.
**Metadata Section Header**

The Metadata Section Header marks the start of the metadata section. There is one metadata section in a RealMedia file and it is expected to be at the end of the file. The following structure describes the Metadata section header:

```
MetadataSectionHeader
{
    u_int32 object_id;
    u_int32 size;
}
```

The Metadata Section Header contains the following fields:

**object_id**

The unique object ID for the Metadata Section Header (“RMMD”). The size of this member is 32 bits.

**size**

The size of the full metadata section in bytes. The size of this member is 32 bits.

**Metadata Tag**

The metadata tag of a RealMedia file consists of a series of properties. The properties are represented as a tree hierarchy with one unnamed root property. Each property contains a type and value, as well as multiple (optional) sub-properties. The following structure describes the details of the metadata tag:

```
MetadataTag
{
    u_int32 object_id;
    u_int32 object_version;
    u_int8[] properties;
}
```

The Metadata Tag contains the following fields:

**object_id**

The unique object ID for the Metadata Tag (“RJMD”). The size of this member is 32 bits.

**object_version**

The version of the Metadata Tag. The size of this member is 32 bits.

**properties[]**

The `MetadataProperty` structure that makes up the metadata tag (see “Metadata Property Structure” on page 543 for more details). As mentioned above, the properties will be represented as one unnamed root metadata property with multiple sub-properties, each with their own optional sub-properties. These will be nested, as in a tree.

**Metadata Property Structure**

The following sample describes the details of the `MetadataProperty` structure:
The `MetadataProperty` structure contains the following fields:

- `size`:
  The size of the `MetadataProperty` structure in bytes. The size of this member is 32 bits.

- `type`:
  The type of the value data. The data in the value array can be one of the following types:
  - `MPT_TEXT`
    The value is string data.
  - `MPT_TEXTLIST`
    The value is a separated list of strings, separator specified as sub-property/type descriptor.
  - `MPT_FLAG`
    The value is a boolean flag—either 1 byte or 4 bytes, check size value.
  - `MPT_ULONG`
    The value is a four-byte integer.
  - `MPT_BINARY`
    The value is a byte stream.
  - `MPT_URL`
    The value is string data.
  - `MPT_DATE`
    The value is a string representation of the date in the form: YYYYmmDDHHMMSS (m = month, M = minutes).
  - `MPT_FILENAME`
    The value is string data.
  - `MPT_GROUPING`
    This property has subproperties, but its own value is empty.
  - `MPT_REFERENCE`
The value is a large buffer of data, use sub-properties/type descriptors to identify mime-type.

The size of this member is 32 bits.

(flags)
Flags describing the property. The following flags are defined these can be used in combination:

- **MPT_READONLY**
  Read only, cannot be modified.
- **MPT_PRIVATE**
  Private, do not expose to users.
- **MPT_TYPE_DESCRIPTOR**
  Type descriptor used to further define type of value.

The size of this member is 32 bits.

(value_offset)
The offset to the value_length, relative to the beginning of the MetadataProperty structure. The size of this member is 32 bits.

(subproperties_offset)
The offset to the subproperties_list, relative to the beginning of the MetadataProperty structure. The size of this member is 32 bits.

(num_subproperties)
The number of subproperties for this MetadataProperty structure. The size of this member is 32 bits.

(name_length)
The length of the name data, including the null-terminator. The size of this member is 32 bits.

(name[])
The name of the property (string data). The size of this member is designated by name_length.

(value_length)
The length of the value data. The size of this member is 32 bits.

(value[])
The value of the property (data depends on the type specified for the property). The size of this member is designated by value_length.

(subproperties_list[])
The list of PropListEntry structures. The PropListEntry structure identifies the offset for each property (see "PropListEntry Structure on page 545 for more details. The size of this member is num_subproperties * sizeof(PropListEntry).

(subproperties[])
The sub-properties. Each sub-property is a MetadataProperty structure with its own size, name, value, sub-properties, and so on. The size of this member is variable.

**PropListEntry Structure**
The following sample describes the details of the PropListEntry structure:
The PropListEntry structure contains the following fields:

```c
PropListEntry
{
    u_int32    offset;
    u_int32    num_props_for_name;
}
```

- **offset**
  The offset for this indexed sub-property, relative to the beginning of the containing MetadataProperty. The size of this member is 32 bits.

- **num_props_for_name**
  The number of sub-properties that share the same name. For example, a lyrics property could have multiple versions as differentiated by the language sub-property type descriptor. The size of this member is 32 bits.

### Metadata Section Footer

The metadata section footer marks the end of the metadata section of a RealMedia file. The metadata section footer contains the size of the metadata tag. Since the metadata section is at the end of the file, the section footer lies a fixed offset of 140 bytes from the end of the file. The size of the metadata tag enables a file reader to quickly locate the beginning of the metadata tag relative to the end of the file. The following structure describes the Metadata section footer.

```c
MetadataSectionFooter
{
    u_int32    object_id;
    u_int32    object_version;
    u_int32    size;
}
```

- **object_id**
  The unique object ID for the Metadata Section Footer ("RMJE"). The size of this member is 32 bits.

- **object_version**
  The version of the metadata tag. The size of this member is 32 bits.

- **size**
  The size of the preceding metadata tag. The size of this member is 32 bits.

### ID3v1 Tag

The ID3v1 Tag is at the end of the metadata section and is expected to be at the end of the entire file. It is a fixed size—128 bytes—and begins with the characters “TAG”. Further information about the informal ID3v1 standard can be found at [http://id3.org/id3v1.html](http://id3.org/id3v1.html).
This chapter lists the header files and sample files you can use to create Helix components.

**Header Files**

The header files in the SDK download describe Helix’s public interfaces. These header files are found in three locations in the SDK: /source/client/include, /source/common/include, and /source/server/include. For descriptions of the interfaces defined in these header files, see “Appendix A: Interface List” beginning in Volume 2, on page 1.

<table>
<thead>
<tr>
<th>File</th>
<th>Defines</th>
</tr>
</thead>
<tbody>
<tr>
<td>hxbastd.h</td>
<td>Definitions used by hxtypes.h that correctly define the basic size types.</td>
</tr>
<tr>
<td>hxcom.h</td>
<td>Definitions for items required for Component Object Model (COM) interfaces in Helix.</td>
</tr>
<tr>
<td>hxresult.h</td>
<td>Definitions for Helix status codes.</td>
</tr>
<tr>
<td>hxtypes.h</td>
<td>Definitions for several types used in Helix.</td>
</tr>
<tr>
<td>hwxintyp.h</td>
<td>Definitions for several types related to Windows, Macintosh, and X-Windows graphical user interface (GUI) layers, as used in Helix.</td>
</tr>
<tr>
<td>hxallow.h</td>
<td>Allowance plug-in interfaces.</td>
</tr>
<tr>
<td>hxasm.h</td>
<td>Adaptive Stream Management (ASM) and BackChannel interfaces.</td>
</tr>
<tr>
<td>hxausvc.h</td>
<td>Audio Services interfaces.</td>
</tr>
<tr>
<td>hxauth.h</td>
<td>Authentication and password-handling interfaces.</td>
</tr>
<tr>
<td>hxauthn.h</td>
<td>Validation interfaces.</td>
</tr>
<tr>
<td>hxcfg.h</td>
<td>Server configuration interfaces.</td>
</tr>
<tr>
<td>hxclsnk.h</td>
<td>Client advise sink interfaces.</td>
</tr>
<tr>
<td>hxcmenu.h</td>
<td>Context menu interfaces.</td>
</tr>
<tr>
<td>hxcomm.h</td>
<td>Common utility interfaces.</td>
</tr>
<tr>
<td>hxcore.h</td>
<td>Client core interfaces.</td>
</tr>
<tr>
<td>hxdb.h</td>
<td>Database interfaces.</td>
</tr>
<tr>
<td>hxdtcvt.h</td>
<td>Data conversion and reversion interfaces.</td>
</tr>
<tr>
<td>hxencod.h</td>
<td>Remote broadcasting interfaces.</td>
</tr>
<tr>
<td>hxengin.h</td>
<td>Callback, networking, and scheduling interfaces.</td>
</tr>
<tr>
<td>hxerror.h</td>
<td>Error-reporting interfaces.</td>
</tr>
<tr>
<td>hxevent.h</td>
<td>Definitions for classes of events.</td>
</tr>
<tr>
<td>hxfiles.h</td>
<td>File system plug-in interfaces.</td>
</tr>
</tbody>
</table>

(Table Page 1 of 2)
Sample Files

The /source/samples directory contains sample C++ files, as well as header files and make files, or project files, for making Helix components. Use the sample files to learn about the Helix architecture and to build your own plug-ins. The /source/samples directory contains subdirectories for audio, client, client applications, common, datatype, server, and video samples. Many of the samples include a directory of test data you can use for testing your compiled plug-ins.

<table>
<thead>
<tr>
<th>File</th>
<th>Defines</th>
</tr>
</thead>
<tbody>
<tr>
<td>hxformt.h</td>
<td>File format and broadcast plug-in interfaces.</td>
</tr>
<tr>
<td>hxgroup.h</td>
<td>Header file for declaration of various group interfaces.</td>
</tr>
<tr>
<td>hxyper.h</td>
<td>Hypernavigation interfaces.</td>
</tr>
<tr>
<td>hxiiids.h</td>
<td>All interface IDs (IIDs) used in Helix interfaces.</td>
</tr>
<tr>
<td>hxlvpix.h</td>
<td>Header file for RealPix live-broadcast interfaces (IHXLiveRealPix and IHXLiveRealPixResponse) and structures for library initialization, image initialization, and effects parameters.</td>
</tr>
<tr>
<td>hxlvtxt.h</td>
<td>Header file for RealText live broadcast sample application.</td>
</tr>
<tr>
<td>hxmon.h</td>
<td>System-monitoring interfaces.</td>
</tr>
<tr>
<td>ihxpcks.ts.h</td>
<td>Packet, buffer, and stream interfaces.</td>
</tr>
<tr>
<td>hxpend.h</td>
<td>Get pending status interface.</td>
</tr>
<tr>
<td>hxhook.h</td>
<td>Client hook interface for recording presentations.</td>
</tr>
<tr>
<td>hxplugns.h</td>
<td>Additional plug-in interfaces.</td>
</tr>
<tr>
<td>hxpugn.h</td>
<td>Plug-in inspector interface.</td>
</tr>
<tr>
<td>hxppv.h</td>
<td>Pay-per-view interfaces.</td>
</tr>
<tr>
<td>hxprefs.h</td>
<td>Persistent preferences interfaces.</td>
</tr>
<tr>
<td>hxpsink.h</td>
<td>Player creation interfaces.</td>
</tr>
<tr>
<td>hxrendr.h</td>
<td>Rendering plug-in interfaces.</td>
</tr>
<tr>
<td>hxsit2.h</td>
<td>Windowing interfaces complementing those in hxwin.h.</td>
</tr>
<tr>
<td>hxsitla.h</td>
<td>Simulated live transfer agent (SLTA) interfaces.</td>
</tr>
<tr>
<td>hxsr.h</td>
<td>Packet source interfaces.</td>
</tr>
<tr>
<td>hxupgrd.h</td>
<td>Automatic upgrade interfaces.</td>
</tr>
<tr>
<td>hxvalue.h</td>
<td>Option and key value interfaces.</td>
</tr>
<tr>
<td>hxvsurf.h</td>
<td>Platform-specific windowing functions.</td>
</tr>
<tr>
<td>hxwin.h</td>
<td>Windowing interfaces.</td>
</tr>
<tr>
<td>hxxml.h</td>
<td>XML parser interfaces.</td>
</tr>
<tr>
<td>hxxmltg.h</td>
<td>XMLTag object interfaces.</td>
</tr>
</tbody>
</table>
### Introductory Samples

The subdirectories in the following table contain introductory sample files. The sample file code in these subdirectories introduces basic Helix concepts and functionality.

<table>
<thead>
<tr>
<th>Directory</th>
<th>Sample</th>
</tr>
</thead>
<tbody>
<tr>
<td>/server/authfilesys</td>
<td>Authenticating file system plug-in.</td>
</tr>
<tr>
<td>/server/converter</td>
<td>Data conversion plug-in for the server. Running the data reversion sample restores this information. (This sample requires RealServer 8 and later, and either RealPlayer 8 and later for Win32 or MacIntosh systems, or RealPlayer 7 and later for UNIX.)</td>
</tr>
<tr>
<td>/server/allowance</td>
<td>Basic allowance plug-in.</td>
</tr>
<tr>
<td>/datatype/fileformat1_renderer</td>
<td>Basic rendering plug-in.</td>
</tr>
<tr>
<td>/server/fileformat2</td>
<td>Basic file-format plug-in.</td>
</tr>
<tr>
<td>/server/filesystem</td>
<td>Basic file-system plug-in.</td>
</tr>
<tr>
<td>/client/hello_world</td>
<td>“Hello world” plug-in that introduces basic features of Helix.</td>
</tr>
<tr>
<td>/client/reverter</td>
<td>Data reversion plug-in for the client. Running this sample restores the information converted by the data conversion sample. (This sample requires RealServer 8 and later, and either RealPlayer 8 and later for Win32 or the MacIntosh, or RealPlayer 7 and later for UNIX.)</td>
</tr>
</tbody>
</table>

### Intermediate Samples

The subdirectories listed in the following table contain intermediate sample files. The sample code in these subdirectories provides more advanced functionality than do the introductory samples.

<table>
<thead>
<tr>
<th>Directory</th>
<th>Sample</th>
</tr>
</thead>
<tbody>
<tr>
<td>/audio/audio_device</td>
<td>Example audio device application.</td>
</tr>
<tr>
<td>/datatype/audio_renderer</td>
<td>Example midstream playback audio-rendering plug-in.</td>
</tr>
<tr>
<td>/server/authbasic</td>
<td>Example authentication plug-in.</td>
</tr>
<tr>
<td>/server/fileformat</td>
<td>Example file format plug-in that works with exrender.</td>
</tr>
<tr>
<td>/datatype/live_slideshow</td>
<td>Example RealPix slideshow application.</td>
</tr>
<tr>
<td>/datatype/live_realpix</td>
<td>Example RealPix broadcast application.</td>
</tr>
<tr>
<td>/server/broadcast</td>
<td>Example broadcast plug-in.</td>
</tr>
<tr>
<td>/server/logging</td>
<td>Example logging plug-in.</td>
</tr>
<tr>
<td>/datatype/live_realtex</td>
<td>Example RealText broadcast application.</td>
</tr>
<tr>
<td>/server/monitor</td>
<td>Example monitor plug-in.</td>
</tr>
<tr>
<td>/server/payperview</td>
<td>Example pay-per-view plug-in.</td>
</tr>
<tr>
<td>/datatype/example_renderer</td>
<td>Example rendering plug-in that works with exffplin.</td>
</tr>
</tbody>
</table>

(Table Page 1 of 2)
Intermediate Samples (continued)

<table>
<thead>
<tr>
<th>Directory</th>
<th>Sample</th>
</tr>
</thead>
<tbody>
<tr>
<td>/datatype/videosurface_renderer</td>
<td>Example rendering plug-in using video surface interfaces.</td>
</tr>
<tr>
<td>/datatype/singlewindow_renderer</td>
<td>Example rendering plug-in for single-window display.</td>
</tr>
<tr>
<td>/clientapps/motif_testclient</td>
<td>Example Motif-based test client running on UNIX only.</td>
</tr>
<tr>
<td>/datatype/pcm_renderer</td>
<td>Example implementation of a basic PCM audio rendering plug-in.</td>
</tr>
<tr>
<td>/datatype/pre_post_file_format</td>
<td>Example postprocessed audio file format plug-in.</td>
</tr>
<tr>
<td>/datatype/pre_post_renderer</td>
<td>Example postprocessed audio-rendering plug-in.</td>
</tr>
<tr>
<td>/clientapps/testplay</td>
<td>Simplified Helix client application for testing plug-ins.</td>
</tr>
</tbody>
</table>

Advanced Samples

The subdirectories listed in the following table contain advanced sample files. The sample file code in these subdirectories illustrates more advanced Helix functionality than do the intermediate samples.

<table>
<thead>
<tr>
<th>Directory</th>
<th>Sample</th>
</tr>
</thead>
<tbody>
<tr>
<td>datatype/audio_renderer2</td>
<td>Audio Services rendering plug-in that shows how to create a new client to start a new timeline and use it for instantaneous sound.</td>
</tr>
<tr>
<td>server/livepacketsink</td>
<td>Monitor plug-in that receives notifications about and examines the headers and packets sent to the server from a live source.</td>
</tr>
<tr>
<td>/datatype/example_renderer2</td>
<td>Rendering plug-in that shows how to create a new client to start a new timeline and play continuous background music, or how to open a different, unrelated URL, such as a stream for an advertisement.</td>
</tr>
<tr>
<td>/video/capture_vframes</td>
<td>Top-level client that shows how to capture or intercept video frames.</td>
</tr>
<tr>
<td>/common/xml_parser</td>
<td>XML parser application that shows how to parse an XML document into its individual components.</td>
</tr>
<tr>
<td>/datatype/pcm_renderer2</td>
<td>Audio Services rendering plug-in sample for PCM data that shows how to get notifications when an audio stream is running dry.</td>
</tr>
</tbody>
</table>
Helix Universal Server Binary Files

You can use the Helix Universal Server utilities listed in the following table with the Helix SDK. These utilities are installed with Helix Universal Server.

<table>
<thead>
<tr>
<th>Windows 32-bit</th>
<th>UNIX</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>encnet.dll</td>
<td>encnet.so</td>
<td>Library for remote broadcasting (used with RealText and RealPix).</td>
</tr>
<tr>
<td>iqsltalib.dll</td>
<td>iqsltalib.so</td>
<td>Library for the simulated live transfer agent.</td>
</tr>
<tr>
<td>pxlive.dll</td>
<td>pxlive.so</td>
<td>Library for RealPix broadcasting.</td>
</tr>
<tr>
<td>rembrdcst.dll</td>
<td>rembrdcst.so</td>
<td>Library for remote broadcasting.</td>
</tr>
<tr>
<td>rtlive.dll</td>
<td>rtlive.so</td>
<td>Library for RealText broadcasting.</td>
</tr>
<tr>
<td>makepass.exe</td>
<td>makepass</td>
<td>Program that adds user names and passwords to the system password file for authentication.</td>
</tr>
<tr>
<td>slta.exe</td>
<td>slta</td>
<td>Program that simulates a live broadcast.</td>
</tr>
</tbody>
</table>

RealPlayer Binary Files

RealPlayer executable files and libraries are automatically placed in a variety of subdirectories when you install RealPlayer. Your license agreement determines which components you can distribute with any application that includes RealPlayer functionality. Please refer to your license agreement to determine the specific components that you are able to distribute.

Free downloads of RealPlayer are available at [http://www.real.com](http://www.real.com).
INDEX FOR VOLUME 2

A
Abort, IHXAutoConfig, 57
AbsoluteEnter, IHXOptimizedScheduler, 239
AbsoluteEnter, IHXScheduler, 373
AbsoluteEnter, IHXTreadSafeScheduler, 432
ActivateFocus, IHXFocusNavigation, 164
Add, IHXAsyncIOSelection, 19
Add, IHXUpgradeCollection, 447
AddAdviseSink, IHXPlayer, 262
AddAdviseSink, IHXVolume, 476
AddBuf, IHXRegistry, 339
AddChildMenu, IHXContextMenu, 88
AddChildPlayer, IHXPlayerNavigator, 279
AddComp, IHXRegistry, 339
AddData, IHXLiveText, 219
AddDryNotification, IHXAudioStream, 37
AddProp, IHXPropWatchResponse, 323
AddErrorSink, IHXErrorSinkControl, 127
AddGroup, IHXGroupManager, 177
AddHook, IHXEventHookMgr, 131
AddInt, IHXRegistry, 339
AddIntRef, IHXRegistry, 340
AddKeyVal, IHXKeyValueList, 200
AddMultic和平ControlConverter, IHXDataConvert, 96
AddPassiveSiteWatcher, IHXSite2, 387
AddPrefetchSink, IHXPrefetch, 314
AddPrefetchTrack, IHXPrefetch, 314
AddPreMixHook, IHXAudioStream, 37
AddPrincipal, IHXAuthenticationDBManager, 46
AddPrincipalDone, IHXAuthenticationDBManagerResponse, 48
AddRedirect, IHXRedirectDBManager, 331
AddRedirectDone, IHXRedirectDBManagerResponse, 333
Addr、IHUnknown, 494
AddRepeat, IHXTrack, 434
AddSeparator, IHXContextMenu, 89
AddSink, IHXGroupManager, 177
AddSink, IHXPlayerSinkControl, 281
AddSite, IHXSiteManager, 394
AddStr, IHXRegistry, 340

AddStreamSink, IHXASMSStream, 12
AddTickerItem, IHXLiveText, 220
AddTrack, IHXGroup, 172
AdjustRecordTimeline, IHXRecordTimeline, 330
Advise, IHXFileObject, 143
AlertAndDisconnect, IHXPlayerController, 275
AllGroupsRemoved, IHXGroupSink, 181
AppendAllListItems, IHXKeyValueList, 201
AtInterruptTime, IHXInterruptState, 194
AttachSite, IHXSiteUser, 397
AttachSite, IHXSiteWatcher, 401
AttachUser, IHXSite, 383
AttachWatcher, IHXSite, 384
AttachWindow, IHXSiteWindowed, 403
Authenticated, IHXClientAuthConversation, 74
AuthenticationRequestDone, IHXAuthenticationManagerResponse, 52

B
Begin, IHXPlayer, 263
Begin, IHXTrack, 434
BeginChangeLayout, IHXSiteSupplier, 395
BeginDone, IHXTrackSink, 437
BeginOptimizedBlt, IHXVideoSurface, 468
BeginSoundLevelAnimation, IHXTrack, 435
BeginSubPref, IHXPreferenceEnumerator, 309
BeginTransmission, IHXioQSLTA, 196
Bind, IHXTCPsocket, 427
Bind, IHXUDPsocket, 443
Blt, IHXVideoSurface, 468
BroadcastTypeFound, IHXBroadcastMapperResponse, 64

C
CanViewRights, IHXViewDRMRights, 472
CanViewSource, IHXViewSourceCommand, 473
ChallengeReady, IHXServerAuthResponse, 377
ChangeMenuitem, IHXContextMenu, 89
ChangingPosition, IHXSiteWatcher, 401
ChangingSize, IHXSiteWatcher, 402
CheckExistence, IHXAuthenticationDBAccess, 42
CheckFormat, IHXAudioDevice, 22
ClearFocus, IHXDrawFocus, 112
ClearFocus, IHXFocusNavigation, 164
ClearWatchByIdx, IHXPropWatch, 320
ClearWatchByName, IHXPropWatch, 320
ClearWatchOnRoot, IHXPropWatch, 321
Clone, IHXAsyncEnumAuthenticationDB, 15
Close, IHXAudioDevice, 22
Close, IHXAutoConfig, 57
Close, IHXFileFormatObject, 137
Close, IHXFileObject, 143
Close, IHXFileViewSource, 160
Close, IHXQLSTLA, 197
Close, IHXPlugin2Handler, 284
Close, IHXSyncFileFormatObject, 422
Close, IHXXMLTagObject, 491
Closed, IHXTCPResponse, 425
CloseDirHandler, IHXDirHandler, 108
CloseDirHandlerDone, IHXDirHandlerResponse, 110
CloseDone, IHXFileResponse, 148
CloseDone, IHXFileViewSourceResponse, 162
CloseEngine, 515
CloseHook, IHXPacketHookManager, 251
ClosePlayer, IHXClientEngine, 77
Connect, IHXSLTA, 406
Connect, IHXTCPResponse, 428
ConnectDone, IHXTCPResponse, 425
ConnectionCheckFailed, IHXConnectionlessControl, 86
ControlBufferReady, IHXDataConvert, 97
ControlBufferReady, IHXDataRevert, 104
ConvertData
   IHXDataConvert, 97
   ConvertedDataReady, IHXDataConvertResponse, 99
ConvertDone, IHXFileResponse, 148
ConvertStreamHeaderReady, IHXDataConvertResponse, 99
ConvertStreamHeaderReady, IHXDataConvertResponse, 100
DataConvertInit, IHXDataConvert, 98
DataConvertInitDone, IHXDataConvertResponse, 100
DataConvertInitDone, IHXDataConvertResponse, 106
DataRevertInit, IHXDataRevert, 104
DataRevertInitDone, IHXDataRevertResponse, 106
DataRevertInitDone, IHXDataRevertResponse, 106
 DECLARE AverageBitrate, IHXLiveText2, 226
 DECLARE MaximumBitrate, IHXLiveText2, 226
 DeductTime, IHXPPVDatabase, 296
 DeleteActiveProp, IHXActivePropUser, 2
 DeleteActiveProp, IHXActiveRegistry, 7
 DeleteActivePropDone, IHXActivePropUserResponse, 4
 DeleteByld, IHXRegistry, 341
 DeleteByName, IHXRegistry, 341
 DeleteProps, IHXPropWatchManager, 327
 Destroy, IHXSiteWindowed, 404
 DestroyChild, IHXSite, 384
 DestroySiteUser, IHXSiteUserSupplier, 399
 DetachSite, IHXSiteUser, 397
 DetachSite, IHXSiteWatcher, 402
 DetachSiteWatcher, IHXSite, 385
 DetachWindow, IHXSiteWindowed, 404
 DirObjectReady, IHXFileSystemManagerResponse, 156
 Disconnect, IHXPlayerController, 275
 Disconnect, IHXSLTA, 407
 DoAutoConfig, IHXAutoConfig, 57
 DoesExist, IHXFileExists, 135
 DoesExistDone, IHXFileExistsResponse, 136
 Done, IHXDataConvert, 98
 Done, IHXRawSourceObject, 327
 Done, IHXSourceFinderObject, 412
 DoneChangeLayout, IHXSiteSupplier, 395
 DoViewSource, IHXViewSourceCommand, 473
 Drain, IHXAudioDevice, 23
Index for Volume 2

E  EffectSent, IHXLiveRealPixResponse, 215
EnableConnectionlessControl, IHXConnectionlessControl, 86
EnableInterrupt, IHXInterruptState, 194
Encode, IHXSLTA, 407
EncoderDone, IHXEncoderCompletion, 118
EncoderIsDone, IHXLiveText, 220
EncoderIsInitialized, IHXLiveText, 221
EncoderResponseDone, IHXEncoderResponseCompletion, 122
EncoderStarted, IHXLiveRealPixResponse, 215
EncoderStopped, IHXLiveRealPixResponse, 216
EndChildMenu, IHXContextMenu, 90
EndOptimizedBlt, IHXVideoSurface, 469
EndSoundLevelAnimation, IHXTrack, 435
EndStream, IHXRenderer, 358
EndSubPref, IHXPreferenceEnumerator, 309
EndTransmission, IHXQSLTA, 198
EnterFullScreen, IHXSiteFullScreen, 392
EndInterruptState, IHXInterruptState, 195
EnumerateInterfaces, IHXNetworkInterfaceEnumerator, 233
ErrorOccurred, IHXErrorSink, 125
ErrorOccurred, IHXLiveRealPixResponse, 216
EventOccurred, IHXClientEngine, 78
EventOccurred, IHXSiteWindowless, 405
ExistenceCheckDone, IHXAuthenticationDBAccessResponse, 44
ExitFullScreen, IHXSiteFullScreen, 392

F  FileHeaderReady, IHXEncoderResponse, 119
FileHeaderReady, IHXFormatResponse, 166
FileHeaderReady, IHXRawSinkObject, 325
FileObjectReady, IHXFileSystemManagerResponse, 156
FileObjectReady, IHXGetFileFromSamePoolResponse, 171
Find, IHXSourceFinderObject, 412
FindBroadcastType, IHXBroadcastMapper, 63
FindDone, IHXSourceFinderResponse, 413
FindImplementationFromClassID, IHXPlugin2Handler, 285
FindIndexUsingStrings, IHXPlugin2Handler, 285
FindIndexUsingValues, IHXPlugin2Handler, 286
FindMimeType, IHXFileMimeMapper, 140
FindParentIdById, IHXRegistry, 341
FindParentIdByName, IHXRegistry, 342
FindPluginUsingStrings, IHXPlugin2Handler, 286
FindPluginUsingValues, IHXPlugin2Handler, 287
flush, IHXLiveText, 221
FlushCache, IHXPlugin2Handler, 287
ForceRedraw, IHXSite, 385
Fork, IHXServerFork, 380
Func, IHXCallback, 67
Func, IHXProcessEntryPoint, 319

G  Get, IHXBuffer, 65
Get, IHXPacket, 244
Get, IHXTQTPacket, 368
GetASMFlags, IHXPacket, 245
GetASMFlags, IHXRTQTPacket, 369
GetASMRuleNumber, IHXPacket, 245
GetASMRuleNumber, IHXRTQTPacket, 369
GetAt, IHXUpgradeCollection, 448
GetAudioFormat, IHXAudioDeviceManager, 26
GetAudioFormat, IHXAudioStream2, 40
GetAudioStream, IHXAudioPlayer, 33
GetAudioStreamCount, IHXAudioPlayer, 33
GetAudioVolume, IHXAudioPlayer, 33
GetAudioVolume, IHXAudioStream, 38
GetAuthorityName, IHXUserProperties, 454
GetBroadcastFormatInfo, IHXBroadcastFormatObject, 60
GetBufByld, IHXRegistry, 341
GetBufByName, IHXRegistry, 342
GetBuffer, IHXBuffer, 65
GetBuffer, IHXPacket, 245
GetBuffer, IHXRTQTPacket, 369
GetChildPlayer, IHXPlayerNavigator, 279
GetClientContext, IHXPlayer, 263
GetClientEngine, IHXPlayer, 263
GetConversionMimeType, IHXDataConvert, 98
GetCount, IHXUpgradeCollection, 448
GetCredentials, IHXAuthenticationDBAccess, 43
GetCredentials, IHXCredentialRequest, 93
GetCredentialsDone, IHXAuthenticationDBAccessResponse, 44
GetCurrentAudioTime, IHXAudioDevice, 23
GetCurrentGroup, IHXGroupManager, 178
GetCurrentPlayTime, IHXPlayer, 263
GetCurrentSchedulerTime, IHXOptimizedScheduler, 239
GetCurrentSchedulerTime, IHXScheduler, 373
GetDataConvertInfo, IHXDataConvertSystemObject, 102
GetDataRevertInfo, IHXDataRevert, 105
GetDeviceVolume, IHXAudioPlayer, 33
GetDirObjectFromURL, IHXFileSystemManager, 153
GetDisplayType, IHXRenderer, 359
GetErrorText, IHXErrorMessages, 123
GetFileFormatInfo, IHXFileFormatObject, 137
GetFileFormatInfo, IHXSyncFileFormatObject, 422
GetFileHeader, IHXBroadcastFormatObject, 60
GetFileHeader, IHXEncoder, 116
GetFileHeader, IHXFileFormatObject, 138
GetFileHeader, IHXRawSourceObject, 327
GetFileHeader, IHXSyncFileFormatObject, 423
GetFilename, IHXConfigFile, 84
GetFilename, IHXFileObject, 144
GetFileObject, IHXFileSystemManager, 153
GetFileObjectFromPool, IHXGetFileFromSamePool, 170
GetFileInfo, IHXFileFormatObject, 138
GetFileInfo, IHXSyncFileFormatObject, 423
GetFocusState, IHXFocusNavigation, 164
GetForeignAddress, IHXTCPSocket, 428
GetForeignPort, IHXTCPSocket, 428
GetGroup, IHXGroupManager, 178
GetGroupCount, IHXGroupManager, 178
GetGroupProperties, IHXGroup, 172
GetHeader, IHXStream, 416
GetHostByName, IHXResolver, 366
GetHostByNameDone, IHXResolverResponse, 367
GetId, IHXRegistry, 342
GetID, IHXRegistryID, 349
GetInstance, IHXPlugin2Handler, 287
GetInstanceFromID, IHXDatabaseManager, 95
GetIntByld, IHXRegistry, 342
GetIntByName, IHXRegistry, 343
GetItem, IHXKeyValueList, 201
GetItemOneKey, IHXKeyValueList, 201
GetKeyboardFocus, IHXKeyboardFocus, 199
GetLastSentTextEndTime, IHXLiveText2, 227
GetLocalAddress, IHXTCPSocket, 428
GetLocalPort, IHXTCPSocket, 429
GetLocalPort, IHXUPDSocket, 444
GetMinimumPreroll, IHXPlayer2, 266
GetMute, IHXVolume, 476
GetNewFileObject, IHXFileSystemManager, 154
GetNextPair, IHXKeyValueList, 203
GetNextPropertyBuffer, IHXValues, 455
GetNextPropertyCString, IHXValues, 456
GetNextPropertyULONG32, IHXValues, 456
GetFirstSite, IHXSiteEnumerator, 390
GetFirstPropertyBuffer, IHXValues, 455
GetFirstPropertyCString, IHXValues, 456
GetFirstPropertyULONG32, IHXValues, 456
GetNextPropertyBuffer, IHXValues, 456
GetNextPropertyCString, IHXValues, 457
GetNextPropertyULONG32, IHXValues, 457
GetnextSite, IHXSiteEnumerator, 390
GetNextString, IHXKeyValueListIterOneKey, 201
GetNextString, IHXKeyValueListIterOneKey, 201
GetNextString, IHXKeyValueList, 201
GetNumberofChildPlayers, IHXSite2, 387
GetNumChildPlayer, IHXPlayerNavigator, 280
GetNumOPlugins, IHXPlugin GetEnumerator, 290
GetNumOPlugins, IHXPluginGroupEnumerator, 292
GetNumPlugins, IHXPluginFactory, 291
GetNumPlugins2, IHXPlugin2Handler, 288
GetNumPropsAtRoot, IHXRegistry, 343
GetNumPropsByld, IHXRegistry, 343
GetNumPropsByld, IHXRegistry, 343
GetOptimizedFormat, IHXVideoSurface, 469
GetOptions, IHXOptions, 241
GetPacket, IHXFileFormatObject, 138
GetPacket, IHXSyncFileFormatObject, 423
GetParentPlayer, IHXPlayerNavigator, 280
GetParentWindow, IHXSiteWindowless, 405
GetPasswordFromUserld, IHXPPVDatabase, 297
GetPermissions, IHXPPVDBManager, 303
GetPermissionsDone, IHXPPVDBManagerResponse, 307
GetPlayer, IHXClientEngine, 78
GetPlayer, IHXStreamSource, 420
GetPlayerCount, IHXClientEngine, 78
GetPlugin, IHXPluginFactory, 291
GetPlugin, IHXPluginFactory, 291
GetPlugin, IHXPluginGroupEnumerator, 292
GetPluginInfo, IHXPlugin, 282
GetPluginInfo, IHXPlugin2Handler, 288
GetPosition, IHXSite, 385
GetPPVDBInfo, IHXPPVDatabase, 297
GetPreference Enumerator, IHXPreference, 312
GetPreferredFormat, IHXVideoSurface, 470
GetPrefetchTrack, IHXPrefetch, 314
GetPrefetchTrackCount, IHXPrefetch, 315
GetPrefKey, IHXPreference Enumerator, 309
GetPresentationProperties, IHXGroupManager, 179
GetPrincipalID, IHXUserProperties, 454
GetPrincipalIDFromGUID, IHXGUIDDBManager, 187
GetPrincipalIDFromGUIDDone, IHXGUIDDBManagerResponse, 189
GetProperties, IHXPluginProperties, 294
GetPropertyBuffer, IHXValues, 457
GetPropertyCString, IHXValues, 457
GetPropertyULONG32, IHXValues, 458
GetPropListByld, IHXRegistry, 344
GetPropListByName, IHXRegistry, 344
GetPropListORoot, IHXRegistry, 344
GetPropName, IHXRegistry, 344
GetRecordPos, IHXRecordTimeline, 330
GetRedirect, IHXPPVDatabase, 297
GetRedirect, IHXRedirectDBManager, 331
GetRedirectdone, IHXRedirectDBManagerResponse, 333
GetRelativeFileObject, IHXFileSystemManager, 154
GetRenderer, IHXStream, 416
GetRendererCount, IHXStream, 417
GetRendererInfo, IHXRenderer, 359
GetRequest, IHXPlayer2, 266
GetRequest, IHXRequestHandler, 365
GetResponseHeaders, IHXRequest, 363
GetSize, IHXBuffer, 66
GetSize, IHXSite, 385
GetSoundLevel, IHXTrack, 435
GetSource, IHXFileViewSource, 160
GetSource, IHXPlayer, 263
GetSource, IHXStream, 417
GetSource, IHXTrack, 435
GetSourceCount, IHXPlayer, 264
GetStatus, IHXPendingStatus, 260
GetStrByld, IHXRegistry, 345
GetStrByName, IHXRegistry, 345
GetStream, IHXStreamSource, 420
GetStreamCount, IHXStreamSource, 421
GetStreamHeader, IHXBroadcastFormatObject, 61
GetStreamHeader, IHXEncoder, 116
GetStreamHeader, IHXFileFormatObject, 138
GetStreamHeader, IHXRawSourceObject, 327
GetStreamHeader, IHXSyncFileFormatObject, 423
GetStreamInfo, IHXAudioStream, 38
GetStreamNumber, IHXPacket, 245
GetStreamNumber, IHXRTPPacket, 370
GetStreamNumber, IHXStream, 417
GetStreamType, IHXStream, 417
GetSupportedPacketFormats, IHXPacketFormat, 247
GetTagHandlerInfo, IHXXMLTagHandler, 489
GetTime, IHXEncoderResponse, 120
GetTime, IHXLiveRealPix, 210
GetTime, IHXLiveText, 221
GetTime, IHXPacket, 245
GetTime, IHXRTPPacket, 370
GetTrack, IHXGroup, 173
GetTrackCount, IHXGroup, 173
GetTrackProperties, IHXTrack, 435
GetTypeByld, IHXRegistry, 345
GetTypeByName, IHXRegistry, 345
GetURL, IHXRequest, 364
GetURL, IHXStreamSource, 421
GetUser, IHXSite, 386
GetUserContext, IHXServerAuthConversation, 375
GetUseridFromGUID, IHXPVDatabase, 298
GetVideoSurface, IHXSite2, 388
GetViewSourceURL, IHXViewSourceCommand, 474
GetVolume, IHXAudioDevice, 23
GetVolume, IHXVolume, 477
GetWindow, IHXSiteWindowed, 404
GetZOrder, IHXSite2, 388
GoToURL, IHXHyperNavigate, 191
GrantPermission, IHXPPVDatabase, 298
GrantTime, IHXPPVDatabase, 298
GroupAdded, IHXGroupSink, 182
GroupInsertedAfter, IHXGroupSink2, 185
GroupInsertedBefore, IHXGroupSink2, 185
GroupRemoved, IHXGroupSink, 182
GroupReplaced, IHXGroupSink2, 186

H
HandleAuthenticationRequest, IHXAuthenticationManager, 50
HandleAuthenticationRequest2, IHXAuthenticationManager2, 51
HandleEvent, IHXEventHook, 129
HandleEvent, IHXSiteUser, 398
HandlePacket, IHXPacketTimeOffsetHandler, 253
HasComponents, IHXUpgradeHandler, 450
HostRedirect, IHXPlayerController, 276
HXAudioData, 497
HXAudioFormat, 497
HXBitmapInfo, 498
HXBitmapInfoHeader, 500
HXCreateInstance, 516
HXShutdown, 516
HXTimeval, 502
HXxEvent, 502
HXxPoint, 503
HXxRect, 503
HXxSize, 503
HXxWindow, 504

I
IgnoreExtraSpaces, IHXLiveText2, 227
IHXActivePropUser, 2
DeleteActiveProp, 2
SetActiveBuf, 2
SetActiveInt, 3
SetActiveStr, 3
IHXActivePropUserResponse, 4
DeleteActivePropDone, 4
SetActiveBufDone, 4
SetActiveIntDone, 5
SetActiveStrDone, 6
IHXActiveRegistry, 7
DeleteActiveProp, 7
isActive, 7
SetActiveBuf, 8
SetActiveInt, 8
Abort, 57
Close, 57
DoAutoConfig, 57
Init, 57
IHXAutoConfigResponse, 58
  OnBegin, 58
  OnComplete, 58
  OnProgress, 58
IHXBackChannel, 59
  PacketReady, 59
IHXBroadcastFormatObject, 60
  GetBroadcastFormatInfo, 60
  GetFileHeader, 60
  GetStreamHeader, 61
  InitBroadcastFormat, 61
  StartPackets, 61
  StopPackets, 61
IHXBroadcastMapper, 63
  FindBroadcastType, 63
IHXBroadcastMapperResponse, 64
  BroadcastTypeFound, 64
IHXBuffer, 65
  Get, 65
  GetBuffer, 65
  GetSize, 66
  Set, 66
  SetSize, 66
IHXCallback, 67
  Func, 67
IHXChallenge, 68
  SendChallenge, 68
IHXChallengeResponse, 68
  ResponseReady, 69
IHXClientAdviseSink, 70
  OnBegin, 70
  OnBuffering, 71
  OnContacting, 71
  OnPause, 71
  OnPosLength, 72
  OnPostSeek, 72
  OnPreSeek, 72
  OnPresentationClosed, 72
  OnPresentationOpened, 73
  OnStatisticsChanged, 73
  OnStop, 73
IHXClientAuthConversation, 74
  Authenticated, 74
  IsDone, 74
  MakeResponse, 74
IHXClientAuthResponse, 76
  ResponseReady, 76
IHXClientEngine, 77
  ClosePlayer, 77
  CreatePlayer, 77
  EventOccurred, 78
  GetPlayer, 78
  GetPlayerCount, 78
IHXClientEngineSelector, 79
  Select, 79
IHXClientEngineSetup, 81
  Setup, 81
IHXCommonClassFactory, 82
  CreateInstance, 82
  CreateInstanceAggregatable, 83
IHXConfigFile, 84
  GetFilename, 84
  LoadFrom, 84
  Reload, 84
  Save, 85
  SaveAs, 85
  SetFilename, 85
IHXConnectionlessControl, 86
  ConnectionCheckFailed, 86
  EnableConnectionlessControl, 86
  SetConnectionTimeout, 86
IHXContextMenu, 88
  AddChildMenu, 88
  AddMenuItem, 88
  AddSeparator, 89
  ChangeMenuItem, 89
  EndChildMenu, 90
  InitContextMenu, 90
  ShowMenu, 90
IHXContextMenuResponse, 91
  OnCanceled, 91
  OnCommand, 91
IHXCopyRegistry, 92
  CopyByName, 92
IHXCredRequest, 93
  GetCredentials, 93
IHXCredRequestResponse, 94
  CredentialsReady, 94
IHXDatabaseManager, 95
  GetInstanceFromID, 95
IHXDataConvert, 96
  GetConversionMimeType, 98
  AddMulticastControlConverter, 96
  ControlBufferReady, 97
  ConvertData, 97
  ConvertFileHeader, 97
  ConvertStreamHeader, 97
  DataConvertInit, 98
  Done, 98
IHXFileRemove, 146
Remove, 146
IHXFileRename, 147
Rename, 147
IHXFileResponse, 148
CloseDone, 148
InitDone, 148
ReadDone, 149
SeekDone, 149
WriteDone, 149
IHXFileStat, 150
Stat, 150
IHXFileStatResponse, 151
StatDone, 151
IHXFileSystemManager, 153
GetDirObjectFromURL, 153
GetFileObject, 153
GetNewFileObject, 154
GetRelativeFileObject, 154
Init, 154
IHXFileSystemManagerResponse, 156
DirObjectReady, 156
FileObjectReady, 156
InitDone, 157
IHXFileSystemObject, 158
CreateDir, 158
CreateFile, 158
GetFileSystemInfo, 159
InitFileSystem, 159
IHXFileViewSource, 160
Close, 160
GetSource, 160
InitViewSource, 160
IHXFileViewSourceResponse, 162
CloseDone, 162
InitDone, 162
SourceReady, 162
IHXFocusNavigation, 164
ActivateFocus, 164
ClearFocus, 164
GetFocusState, 164
SetFocus, 165
IHXFormatResponse, 166
FileHeaderReady, 166
InitDone, 166
PacketReady, 167
SeekDone, 167
StreamDone, 167
StreamHeaderReady, 167
IHXGenericPlugin, 169
IsGeneric, 169
IHXGetFileFromSamePool, 170
GetFileObjectFromPool, 170
IHXGetFileFromSamePoolResponse, 171
FileObjectReady, 171
IHXGroup, 172
AddTrack, 172
GetGroupProperties, 172
GetTrack, 173
GetTrackCount, 173
RemoveTrack, 173
SetGroupProperties, 173
IHXGroupManager, 177
AddGroup, 177
AddSink, 177
CreateGroup, 178
GetCurrentGroup, 178
GetGroup, 178
GetGroupCount, 178
GetPresentationProperties, 179
RemoveGroup, 179
RemoveSink, 179
SetCurrentGroup, 179
SetPresentationProperties, 179
IHXGroupSink, 181
AllGroupsRemoved, 181
CurrentGroupSet, 181
GroupAdded, 182
GroupRemoved, 182
TrackAdded, 182
TrackRemoved, 183
TrackStarted, 183
TrackStopped, 183
IHXGroupSink2, 185
GroupInsertedAfter, 185
GroupInsertedBefore, 185
GroupReplaced, 186
IHXGUIDDBManager, 187
GetPrincipalIDFromGUID, 187
SetGUIDForPrincipalID, 187
IHXGUIDDBManagerResponse, 189
GetPrincipalIDFromGUIDDone, 189
SetGUIDForPrincipalIDDone, 189
IHXHyperNavigate, 191
GoToURL, 191
IHXInfoLogger, 192
LogInformation, 192
IHXInterruptSafe, 193
IsInterruptSafe, 193
IHXInterruptState, 194
AtInterruptTime, 194
EnableInterrupt, 194
EnterInterruptState, 195
IsInterruptEnabled, 195
LeaveInterruptState, 195
IHXiQSLTA, 196
   BeginTransmission, 196
   Close, 197
   EndTransmission, 198
   Init, 197
   SetTAC, 198
   SetTargetBandwidth, 198
IHXKeyboardFocus, 199
   GetKeyboardFocus, 199
   SetKeyboardFocus, 199
IHXKeyValueList, 200
   AddKeyValue, 200
   AppendAllListItems, 201
   CreateObject, 201
   GetIterator, 201
   GetIteratorOneKey, 201
   ImportValues, 202
   KeyExists, 202
IHXKeyValueListIterator
   GetNextPair, 203
   ReplaceCurrent, 203
IHXKeyValueListIteratorOneKey
   GetNextString, 204
   ReplaceCurrent, 204
IHXListenResponse, 205
   NewConnection, 205
IHXListenSocket, 206
   Init, 206
IHXLiveFileFormatInfo, 207
   FormResendPacket, 207
   GetResendBitrate, 207
   GetResendDuration, 208
   IsLiveResendRequired, 208
   VerifyFileCompatibility, 208
   VerifyStreamCompatibility, 209
IHXLiveRealPix, 210
   GetTime, 210
   InitImage, 210
   Process, 211
   SendEffect, 211
   SendImage, 211
   StartEncoder, 211
   StopEncoder, 212
IHXLiveRealPixResponse, 215
   EffectSent, 215
   EncoderStarted, 215
   EncoderStopped, 216
   ErrorOccurred, 216
   ImageSent, 217
IHXLiveText, 219
   AddData, 219
   AddTickerItem, 220
   EncoderIsDone, 220
   EncoderIsInitialized, 221
   flush, 221
   GetTime, 221
   InitLiveText, 221
   PacketsHaveStarted, 222
   Process, 222
   SetBackgroundColor, 222
   SetDoLooping, 223
   SetEncoderDone, 223
   SetHyperlinkInfo, 223
   SetTextMotion, 223
   SetType, 224
   SetWindowDimensions, 224
   UseWordwrap, 225
IHXLiveText2, 226
   DeclareAverageBitrate, 226
   DeclareMaximumBitrate, 226
   GetLastSentTextEndTime, 227
   IgnoreExtraSpaces, 227
   SetFlags, 227
   SetTimeBetweenDryStreamResends, 227
IHXLoadBalancedListen, 229
   SetID, 229
   SetReserveLimit, 229
IHXMultiInstanceSiteUserSupplier, 231
   ReleaseSingleSiteUser, 231
   SetSingleSiteUser, 231
IHXNetworkInterfaceEnumerator, 233
   EnumerateInterfaces, 233
IHXNetworkServices, 234
   CreateListenSocket, 234
   CreateResolver, 234
   CreateTCPSocket, 235
   CreateUDPSocket, 235
IHXNetworkServices2, 236
   CreateLBoundTCPSocket, 236
IHXObjectConfiguration, 237
   SetConfiguration, 237
   SetContext, 238
IHXOptimizedScheduler, 239
   AbsoluteEnter, 239
   GetCurrentSchedulerTime, 239
   RelativeEnter, 240
   Remove, 240
IHXOptions, 241
   GetOptions, 241
   SetOptionBuffer, 241
   SetOptionCString, 242
   SetOptionULONG32, 242
IHXOverrideDefaultServices, 243
Index for Volume 2

OverrideServices, 243
IHXPacket, 244
  Get, 244
  GetASMFlags, 245
  GetASMRuleNumber, 245
  GetBuffer, 245
  GetStreamNumber, 245
  GetTime, 245
  IsLost, 246
  Set, 246
  SetAsLost, 246
IHXPacketFormat, 247
  GetSupportedPacketFormats, 247
  SetPacketFormat, 247
IHXPacketHook, 248
  OnEnd, 248
  OnFileHeader, 248
  OnPacket, 248
  OnStart, 248
  OnStreamHeader, 248
IHXPacketHookHelper, 249
  StartHook, 249
  StopHook, 249
IHXPacketHookHelperResponse, 250
  OnEndOfPackets, 250
  OnPacket, 250
IHXPacketHookManager, 251
  CloseHook, 251
  InitHook, 251
  StartHook, 251
  StopHook, 251
IHXPacketTimeOffsetHandler, 253
  HandlePacket, 253
  Init, 253
  SetTimeOffset, 254
IHXPacketTimeOffsetHandlerResponse, 255
  PacketReady, 255
IHXPassiveSiteWatcher, 256
  PositionChanged, 256
  SizeChanged, 256
IHXPassword, 257
IHXPendingStatus, 260
  GetStatus, 260
IHXPlayer, 262
  AddAdviseSink, 262
  Begin, 263
  GetClientContext, 263
  GetClientEngine, 263
  GetCurrentPlayTime, 263
  GetSource, 263
  GetSourceCount, 264
  IsDone, 264
  IsLive, 264
  OpenURL, 264
  Pause, 264
  RemoveAdviseSink, 265
  Seek, 265
  SetClientContext, 265
  Stop, 265
IHXPlayer2, 266
  GetMinimumPreroll, 266
  GetRequest, 266
  OpenRequest, 267
  SetMinimumPreroll, 267
IHXPlayerConnectionAdviseSink, 269
  OnBegin, 269
  OnConnection, 269
  OnDone, 270
  OnPause, 270
  OnStop, 270
  OnURL, 270
  SetPlayerController, 270
  SetRegistryID, 271
IHXPlayerConnectionAdviseSinkManager, 272
  CreatePlayerConnectionAdviseSink, 272
IHXPlayerConnectionResponse, 273
  OnBeginDone, 273
  OnConnectionDone, 273
  OnPauseDone, 274
  OnStopDone, 274
  OnURLDone, 274
IHXPlayerController, 275
  AlertAndDisconnect, 275
  Disconnect, 275
  HostRedirect, 276
  NetworkRedirect, 276
  Pause, 276
  Redirect, 276
  Resume, 277
IHXPlayerCreationSink, 268
  PlayerClosed, 268
  PlayerCreated, 268
IHXPlayerNavigator, 279
  AddChildPlayer, 279
  GetChildPlayer, 279
  GetNumChildPlayer, 280
  GetParentPlayer, 280
  RemoveChildPlayer, 280
  RemoveParentPlayer, 280
  SetParentPlayer, 280
IHXPlayerSinkControl, 281
  AddSink, 281
  RemoveSink, 281
IHXPlugin, 282
<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>GetPluginInfo</td>
<td>282</td>
</tr>
<tr>
<td>InitPlugin</td>
<td>283</td>
</tr>
<tr>
<td>IHXPlugin2Handler</td>
<td>284</td>
</tr>
<tr>
<td>Close</td>
<td>284</td>
</tr>
<tr>
<td>FindImplementationFromClassID</td>
<td>285</td>
</tr>
<tr>
<td>FindIndexUsingStrings</td>
<td>285</td>
</tr>
<tr>
<td>FindIndexUsingValues</td>
<td>286</td>
</tr>
<tr>
<td>FindPluginUsingStrings</td>
<td>286</td>
</tr>
<tr>
<td>FindPluginUsingValues</td>
<td>287</td>
</tr>
<tr>
<td>FlushCache</td>
<td>287</td>
</tr>
<tr>
<td>GetInstance</td>
<td>287</td>
</tr>
<tr>
<td>GetNumPlugins2</td>
<td>288</td>
</tr>
<tr>
<td>GetPluginInfo</td>
<td>288</td>
</tr>
<tr>
<td>Init</td>
<td>288</td>
</tr>
<tr>
<td>SetCacheSize</td>
<td>288</td>
</tr>
<tr>
<td>SetRequiredPlugins</td>
<td>289</td>
</tr>
<tr>
<td>IHXPluginEnumerator</td>
<td>290</td>
</tr>
<tr>
<td>GetNumOfPlugins</td>
<td>290</td>
</tr>
<tr>
<td>GetPlugIn</td>
<td>290</td>
</tr>
<tr>
<td>IHXPluginFactory</td>
<td>291</td>
</tr>
<tr>
<td>GetNumPlugins</td>
<td>291</td>
</tr>
<tr>
<td>GetPlugIn</td>
<td>291</td>
</tr>
<tr>
<td>IHXPluginGroupEnumerator</td>
<td>292</td>
</tr>
<tr>
<td>GetNumOfPlugins</td>
<td>292</td>
</tr>
<tr>
<td>GetPlugIn</td>
<td>292</td>
</tr>
<tr>
<td>Init</td>
<td>293</td>
</tr>
<tr>
<td>IHXPluginProperties</td>
<td>294</td>
</tr>
<tr>
<td>GetProperties</td>
<td>294</td>
</tr>
<tr>
<td>IHXPluginReloader</td>
<td>295</td>
</tr>
<tr>
<td>ReloadPlugins</td>
<td>295</td>
</tr>
<tr>
<td>IHXPPVDatabase</td>
<td>296</td>
</tr>
<tr>
<td>DeductTime</td>
<td>296</td>
</tr>
<tr>
<td>GetPasswordFromUserid</td>
<td>297</td>
</tr>
<tr>
<td>GetPVDBInfo</td>
<td>297</td>
</tr>
<tr>
<td>GetRedirect</td>
<td>297</td>
</tr>
<tr>
<td>GetUserIdFromGUID</td>
<td>298</td>
</tr>
<tr>
<td>GrantPermission</td>
<td>298</td>
</tr>
<tr>
<td>GrantTime</td>
<td>298</td>
</tr>
<tr>
<td>InitPPVDB</td>
<td>299</td>
</tr>
<tr>
<td>InsertUser</td>
<td>299</td>
</tr>
<tr>
<td>LogAccess</td>
<td>299</td>
</tr>
<tr>
<td>LogReg</td>
<td>300</td>
</tr>
<tr>
<td>PutRedirect</td>
<td>300</td>
</tr>
<tr>
<td>RegisterGUID</td>
<td>300</td>
</tr>
<tr>
<td>RemoveUser</td>
<td>301</td>
</tr>
<tr>
<td>RevokeAllPermissions</td>
<td>301</td>
</tr>
<tr>
<td>RevokePermission</td>
<td>301</td>
</tr>
<tr>
<td>SetPassword</td>
<td>301</td>
</tr>
<tr>
<td>ValidateUser</td>
<td>302</td>
</tr>
<tr>
<td>IHXPPVDBManager</td>
<td>303</td>
</tr>
<tr>
<td>GetPermissions</td>
<td>303</td>
</tr>
<tr>
<td>LogAccessAttempt</td>
<td>303</td>
</tr>
<tr>
<td>RevokeAllPermissions</td>
<td>304</td>
</tr>
<tr>
<td>RevokePermissions</td>
<td>305</td>
</tr>
<tr>
<td>SetPermissions</td>
<td>305</td>
</tr>
<tr>
<td>IHXPPVDBManagerResponse</td>
<td>307</td>
</tr>
<tr>
<td>GetPermissionsDone</td>
<td>307</td>
</tr>
<tr>
<td>RevokeAllPermissionsDone</td>
<td>308</td>
</tr>
<tr>
<td>RevokePermissionsDone</td>
<td>308</td>
</tr>
<tr>
<td>SetPermissionsDone</td>
<td>308</td>
</tr>
<tr>
<td>IHXPreferenceEnumerator</td>
<td>309</td>
</tr>
<tr>
<td>BeginSubPref</td>
<td>309</td>
</tr>
<tr>
<td>EndSubPref</td>
<td>309</td>
</tr>
<tr>
<td>GetPrefKey</td>
<td>309</td>
</tr>
<tr>
<td>ReadPref</td>
<td>310</td>
</tr>
<tr>
<td>IHXPreferences</td>
<td>311</td>
</tr>
<tr>
<td>ReadPref</td>
<td>311</td>
</tr>
<tr>
<td>WritePref</td>
<td>311</td>
</tr>
<tr>
<td>IHXPreferences2</td>
<td>312</td>
</tr>
<tr>
<td>GetPreferenceEnumerator</td>
<td>312</td>
</tr>
<tr>
<td>ResetRoot</td>
<td>312</td>
</tr>
<tr>
<td>IHXPrefetch</td>
<td>314</td>
</tr>
<tr>
<td>AddPrefetchSink</td>
<td>314</td>
</tr>
<tr>
<td>AddPrefetchTrack</td>
<td>314</td>
</tr>
<tr>
<td>GetPrefetchTrack</td>
<td>314</td>
</tr>
<tr>
<td>GetPrefetchTrackCount</td>
<td>315</td>
</tr>
<tr>
<td>RemovePrefetchTrack</td>
<td>315</td>
</tr>
<tr>
<td>RemovePrefetchTrack</td>
<td>315</td>
</tr>
<tr>
<td>IHXPrefetchSink</td>
<td>316</td>
</tr>
<tr>
<td>PrefetchDone</td>
<td>316</td>
</tr>
<tr>
<td>PrefetchTrackAdded</td>
<td>316</td>
</tr>
<tr>
<td>PrefetchTrackRemoved</td>
<td>317</td>
</tr>
<tr>
<td>IHXProcess</td>
<td>318</td>
</tr>
<tr>
<td>Start</td>
<td>318</td>
</tr>
<tr>
<td>IHXProcessEntryPoint</td>
<td>319</td>
</tr>
<tr>
<td>Func</td>
<td>319</td>
</tr>
<tr>
<td>IHXPropWatch</td>
<td>320</td>
</tr>
<tr>
<td>ClearWatchByIId</td>
<td>320</td>
</tr>
<tr>
<td>ClearWatchByName</td>
<td>320</td>
</tr>
<tr>
<td>ClearWatchOnRoot</td>
<td>321</td>
</tr>
<tr>
<td>Init</td>
<td>321</td>
</tr>
<tr>
<td>SetWatchByIId</td>
<td>321</td>
</tr>
<tr>
<td>SetWatchByName</td>
<td>321</td>
</tr>
<tr>
<td>SetWatchOnRoot</td>
<td>322</td>
</tr>
<tr>
<td>IHXPropWatchResponse</td>
<td>323</td>
</tr>
<tr>
<td>AddedProp</td>
<td>323</td>
</tr>
<tr>
<td>DeletedProp</td>
<td>323</td>
</tr>
<tr>
<td>ModifiedProp</td>
<td>324</td>
</tr>
<tr>
<td>IHXRawSinkObject</td>
<td>325</td>
</tr>
<tr>
<td>FileHeaderReady</td>
<td>325</td>
</tr>
<tr>
<td>InitDone</td>
<td>325</td>
</tr>
<tr>
<td>PacketReady</td>
<td>326</td>
</tr>
<tr>
<td>StreamDone</td>
<td>326</td>
</tr>
<tr>
<td>StreamHeaderReady</td>
<td>326</td>
</tr>
</tbody>
</table>
IHXRawSourceObject, 327
   Done, 327
   GetFileHeader, 327
   GetStreamHeader, 327
   Init, 328
   StartPackets, 328
   StopPackets, 328
IHXReconfigServerResponse, 329
   ReconfigServerDone, 329
IHXRecordTimeline, 330
   AdjustRecordTimeline, 330
   GetRecordPos, 330
   IsRecordStartPosSet, 330
   OnPlaybackPos, 330
   SetRecordStartPos, 330
IHXRedirectDBManager, 331
   AddRedirect, 331
   GetRedirect, 331
   RemoveRedirect, 332
IHXRedirectDBManagerResponse, 333
   AddRedirectDone, 333
   GetRedirectDone, 333
   RemoveRedirectDone, 334
IHXRegConfig, 335
   WriteKey, 335
IHXRegistrationLogger, 336
   LogRegistrationAttempt, 336
IHXRegistry, 338
   AddBuf, 339
   AddComp, 339
   AddInt, 339
   AddIntRef, 340
   AddStr, 340
   CreatPropWatch, 340
   DeleteByld, 341
   DeleteByName, 341
   FindParentIdByld, 341
   FindParentIdByName, 341
   GenNumPropsByName, 343
   GetBufByld, 341
   GetBufByName, 342
   GetId, 342
   GetIntByld, 342
   GetIntByName, 343
   GetNumPropsAtRoot, 343
   GetNumPropsByld, 343
   GetPropListByld, 344
   GetPropListByName, 344
   GetPropListOfRoot, 344
   GetPropName, 344
   GetStrByld, 345
   GetStrByName, 345
   GetTypeByld, 345
   GetTypeByName, 345
   SetBufByld, 346
   SetBufByName, 346
   SetIntByld, 346
   SetIntByName, 347
   SetStrByld, 347
   SetStrByName, 347
IHXRegistryAltStringHandling, 348
   SetStringAccessAsBufferByld, 348
IHXRegistryID, 349
   GetID, 349
IHXRenderer, 358
   EndStream, 358
   GetDisplayType, 359
   GetRendererInfo, 359
   OnBegin, 359
   OnBuffering, 360
   OnEndOfPackets, 360
   OnHeader, 360
   OnPacket, 361
   OnPause, 361
   OnPostSeek, 361
   OnPreSeek, 361
   OnTimeSync, 362
   StartStream, 362
IHXRequest, 363
   GetRequestHeaders, 363
   GetResponseHeaders, 363
   GetURL, 364
   SetRequestHeaders, 364
   SetResponseHeaders, 364
   SetURL, 364
IHXRequestHandler, 365
   GetRequest, 365
   SetRequest, 365
IHXResolver, 366
   GetHostByName, 366
   Init, 366
IHXResolverResponse, 367
   GetHostByNameDone, 367
IHXRevert, 104
IHXRTPPacket, 368
   Get, 368
   GetASMFlags, 369
   GetASMRuleNumber, 369
   GetBuffer, 369
   GetRTP, 369
   GetRTPTime, 370
   GetStreamNumber, 370
   GetTime, 370
   IsLost, 370
Set, 371  
SetAsLost, 371  
SetRTP, 371  
IHXScheduler, 373  
    AbsoluteEnter, 373  
    GetCurrentSchedulerTime, 373  
    RelativeEnter, 374  
    Remove, 374  
IHXServerAuthConversation, 375  
    GetUserContext, 375  
    IsAuthenticated, 375  
    MakeChallenge, 376  
IHXServerAuthResponse, 377  
    ChallengeReady, 377  
IHXServerControl, 378  
    ShutdownServer, 378  
IHXServerControl2, 379  
    ReconfigServer, 379  
    RestartServer, 379  
IHXServerFork, 380  
    Fork, 380  
IHXSetSocketOption, 382  
    SetOption, 382  
IHXSite, 383  
    AttachUser, 383  
    AttachWatcher, 384  
    CreateChild, 384  
    DamageRect, 384  
    DamageRegion, 384  
    DestroyChild, 384  
    DetachUser, 385  
    DetachWatcher, 385  
    ForceRedraw, 385  
    GetPosition, 385  
    GetSize, 385  
    GetUser, 386  
    SetPosition, 386  
    SetSize, 386  
IHXSite2, 387  
    AddPassiveSiteWatcher, 387  
    GetNumberOfChildSites, 387  
    GetVideoSurface, 388  
    GetZOrder, 388  
    IsSiteVisible, 388  
    MoveSiteToTop, 388  
    RemovePassiveSiteWatcher, 388  
    SetCursor, 388  
    SetZOrder, 389  
    ShowSite, 389  
    UpdateSiteWindow, 389  
IHXSiteEnumerator, 390  
    GetNextSite, 390  
IHXSiteFullScreen, 392  
    EnterFullScreen, 392  
    ExitFullScreen, 392  
    IsFullScreen, 392  
    TestFullScreen, 393  
IHXSiteManager, 394  
    AddSite, 394  
    RemoveSite, 394  
IHXSiteSupplier, 395  
    BeginChangeLayout, 395  
    DoneChangeLayout, 395  
    SitesNeeded, 395  
    SitesNotNeeded, 396  
IHXSiteUser, 397  
    AttachSite, 397  
    DetachSite, 397  
    HandleEvent, 398  
    NeedsWindowedSites, 398  
IHXSiteUserSupplier, 399  
    CreateSiteUser, 399  
    DestroySiteUser, 399  
    NeedsWindowedSites, 400  
IHXSiteWatcher, 401  
    AttachSite, 401  
    ChangingPosition, 401  
    ChangingSize, 402  
    DetachSite, 402  
IHXSiteWindowed, 403  
    AttachWindow, 403  
    Create, 403  
    Destroy, 404  
    DetachWindow, 404  
    GetWindow, 404  
IHXSiteWindowless, 405  
    EventOccurred, 405  
    GetParentWindow, 405  
IHXSLTA, 406  
    Connect, 406  
    Disconnect, 407  
    Encode, 407  
    SetTAC, 407  
    SetTargetBandwidth, 408  
IHXSLTAEvent, 409  
    SetEvent, 409  
    SetRepeatedEvent, 410  
IHXSourceFinderObject, 412  
    Done, 412  
    Find, 412  
    Init, 412  
IHXSourceFinderResponse, 413  
    FindDone, 413  
    InitDone, 413
IHXStatistics, 414
  InitializeStatistics, 414
  Update, 414
IHXStatusMessage, 415
  SetStatus, 415
IHXStream, 416
  GetHeader, 416
  GetRenderer, 416
  GetRendererCount, 417
  GetSource, 417
  GetStreamNumber, 417
  GetStreamType, 417
  ReportQualityOfService, 418
  ReportRebufferStatus, 418
  SetGranularity, 418
IHXStreamSource, 420
  GetPlayer, 420
  GetStream, 420
  GetStreamCount, 421
  GetURL, 421
  IsLive, 421
IHXSyncFileFormatObject
  GetPacket, 423
  InitFileFormat, 423
IHXSyncFileFormatObject, 422
  Close, 422
  GetFileFormatInfo, 422
  GetFileHeader, 423
  GetStreamHeader, 423
  Seek, 424
IHXTCPResponse, 425
  Closed, 425
  ConnectDone, 425
  ReadDone, 426
  WriteReady, 426
IHXTCPSocket, 427
  Bind, 427
  Connect, 428
  GetForeignAddress, 428
  GetForeignPort, 428
  GetLocalAddress, 428
  GetLocalPort, 429
  Init, 429
  Read, 429
  SetResponse, 429
  WantWrite, 430
  Write, 430
IHXThreadSafeScheduler, 432
  AbsoluteEnter, 432
  RelativeEnter, 432
  Remove, 433
IHXTrack, 434
  AddRepeat, 434
  Begin, 434
  BeginSoundLevelAnimation, 435
  EndSoundLevelAnimation, 435
  GetSoundLevel, 435
  GetSource, 435
  GetTrackProperties, 435
  Pause, 436
  Seek, 436
  SetSoundLevel, 436
  Stop, 436
IHXTrackSink, 437
  BeginDone, 437
  OnSoundLevelAnimation, 437
  PauseDone, 438
  SeekDone, 438
  StopDone, 438
IHXTransportControl, 440
  SetTransportType, 440
IHXUDPInitMulticast, 441
  InitMulticast, 441
IHXUDPResponse, 442
  ReadDone, 442
IHXUDPsocket, 443
  Bind, 443
  GetLocalPort, 444
  Init, 444
  LeaveMulticastGroup, 444
  Read, 445
  Write, 445
  WriteTo, 445
IHXUDPsocket, 444
  JoinMulticastGroup, 444
IHXUpgradeCollection, 447
  Add, 447
  GetAt, 448
  GetCount, 448
  Remove, 448
  RemoveAll, 448
IHXUpgradeHandler, 450
  HasComponents, 450
  RequestUpgrade, 450
IHXUserContext, 452
  IsMemberOf, 452
IHXUserImpersonation, 453
  Start, 453
  Stop, 453
IHXUserProperties, 454
  GetAuthorityName, 454
  GetPrincipalID, 454
IHXValues, 455
  GetFirstPropertyBuffer, 455
Index for Volume 2

IsLive, IHXPlayer, 264
IsLive, IHXStreamSource, 421
IsLost, IHXPacket, 246
IsLost, IHXRTPacket, 370
IsMemberOf, IHXUserContext, 452
IsRecordStartPosSet, IHXRecordTimeline, 330
IsSiteVisible, IHXSite2, 388
IUnknown, 494
AddRef, 494
QueryInterface, 494
Release, 495
IXHSiteEnumerator
GetFirstSite, 390

J
JoinMulticastGroup, IHXUDPSocket, 444

K
KeyExists, IHXKeyValueList, 202

L
LeaveInterruptState, IHXInterruptState, 195
LeaveMulticastGroup, IHXUDPSocket, 444
LoadFrom, IHXConfigFile, 84
LogAccess, IHXPPVDatabase, 299
LogAccessAttempt, IHXPPVDBManager, 303
LogInformation, IHXInfoLogger, 192
LogReg, IHXPPVDatabase, 300
LogRegistrationAttempt, IHXRegistrationLogger, 336

M
Macintosh
interrupt time execution, 193
MakeChallenge, IHXServerAuthConversation, 376
MakeDir, IHXDirHandler, 108
MakeDirDone, IHXDirHandlerResponse, 110
MakeResponse, IHXClientAuthConversation, 74
MimeTypeFound, IHXFileMimeMapperResponse, 141
ModifiedProp, IHXPropWatchResponse, 324
Move, IHXFileMove, 142
MoveSiteToTop, IHXSite2, 388

N
NeedsWindowedSites, IHXSiteUser, 398
NeedsWindowedSites, IHXSiteUserSupplier, 400
NetworkRedirect, IHXPlayerController, 276
NewConnection
IHXListenResponse, 205
_NewEnum, IHXAuthenticationDBAccess, 42
Next, IHXAsyncEnumAuthenticationDB, 15
NextDone, IHXAsyncEnumAuthenticationDBResponse, 17

O
OnBegin, IHXAutoConfigResponse, 58
OnBegin, IHXClientAdviseSink, 70
OnBegin, IHXPlayerConnectionAdviseSink, 269
OnBegin, IHXRenderer, 359
OnBeginDone, IHXPlayerConnectionResponse, 273
OnBuffer, IHXAudioHook, 29
OnBuffering, IHXClientAdviseSink, 71
OnBuffering, IHXRenderer, 360
OnCanceled, IHXContextMenuResponse, 91
OnCommand, IHXContextMenuResponse, 91
OnComplete, IHXAutoConfigResponse, 58
OnConnection, IHXPlayerConnectionAdviseSink, 269
OnConnectionDone, IHXPlayerConnectionResponse, 273
OnContacting, IHXClientAdviseSink, 71
OnDone, IHXPlayerConnectionAdviseSink, 270
OnDryNotification, IHXDryNotification, 115
OnEnd, IHXPacketHook, 248
OnEndOfPackets, IHXPacketHookHelperResponse, 250
OnEndOfPackets, IHXRenderer, 360
OnFileHeader, IHXPacketHook, 248
OnHeader, IHXRenderer, 360
OnInit, IHXAudioHook, 30
OnMuteChange, IHXVolumeAdviseSink, 478
OnPacket, IHXPacketHook, 248
OnPacket, IHXPacketHookHelperResponse, 250
OnPacket, IHXRenderer, 361
OnPause, IHXClientAdviseSink, 71
OnPause, IHXPlayerConnectionAdviseSink, 270
OnPause, IHXRenderer, 361
OnPauseDone, IHXPlayerConnectionResponse, 274
OnPlaybackPos, IHXRecordTimeline, 330
OnPosLength, IHXClientAdviseSink, 72
OnPostSeek, IHXClientAdviseSink, 72
OnPreSeek, IHXRenderer, 361
OnPreSeek, IHXClientAdviseSink, 72
OnPreSeek, IHXRenderer, 361
OnPresentationClosed, IHXClientAdviseSink, 72
OnPresentationOpened, IHXClientAdviseSink, 73
OnProgress, IHXAutoConfigResponse, 58
OnSoundLevelAnimation, IHXTrackSink, 437
OnStart, IHXPacketHook, 248
OnStatisticsChanged, IHXClientAdviseSink, 73
OnStop, IHXClientAdviseSink, 73
OnStop, IHXPlayerConnectionAdviseSink, 270
OnStopDone, IHXPlayerConnectionResponse, 274
OnStream, IHXAudioStreamInfoResponse, 41
OnStreamHeader, IHXPacketHook, 248
OnSubscribe, IHXASMStreamSink, 14
OnTag, IHXXMLTagObject, 491
OnTagDone, IHXXMLTagObjectResponse, 493
OnTimeSync, IHXAudioDeviceResponse, 28
OnTimeSync, IHXAudioPlayerResponse, 35
OnTimeSync, IHXRenderer, 362
OnURL, IHXPlayerConnectionAdviseSink, 270
OnURLDone, IHXPlayerConnectionResponse, 274
OnVolumeChange, IHXVolumeAdviseSink, 478
Open, IHXAudioDevice, 24
OpenRequest, IHXPlayer2, 267
OpenURL, IHXPlayer, 264
OptimizedBlt, IHXVideoSurface, 471
OverrideServices, IHXOverrideDefaultServices, 243

P
PacketReady, IHXBackChannel, 59
PacketReady, IHXEncoderResponse, 120
PacketReady, IHXFormatResponse, 167
PacketReady, IHXPacketTimeOffsetHandlerResponse, 255
PacketReady, IHXRawSinkObject, 326
PacketsHaveStarted, IHXLiveText, 222
Pause, IHXAudioDevice, 24
Pause, IHXPlayer, 264
Pause, IHXPlayerController, 276
Pause, IHXTtrack, 436
PauseDone, IHXTrackSink, 438
PixEffectInfo, 505
PixImageInfo, 508
PixInitInfo, 508
PlayerClosed, IHXPlayerCreationSink, 268
PlayerCreated, IHXPlayerCreationSink, 268
PositionChanged, IHXPassiveSiteWatcher, 256
PPVAccessLog, 510
PPVPermission, 512
PPVRegLog, 513
PrefetchDone, IHXPrefetchSink, 316
PrefetchTrackAdded, IHXPrefetchSink, 316
PrefetchTrackRemoved, IHXPrefetchSink, 317
Process, IHXEncoderResponse, 121
Process, IHXLiveRealPix, 211
Process, IHXLiveText, 222
PutRedirect, IHXPPVDatabase, 300

Q
QueryInterface, IUnknown, 494

R
ramgen, 161
Read, IHXFileObject, 144
Read, IHXTCPASocket, 429
Read, IHXUDPASocket, 445
ReadDir, IHXDirHandler, 109
ReadDirDone, IHXDirHandlerResponse, 111
ReadDone, IHXFileResponse, 149
ReadDone, IHXTCPResponse, 426
ReadDone, IHXUDPResponse, 442
ReadPref, IHXPreferenceEnumerator, 310
ReadPref, IHXPreferences, 311
ReconfigServer, IHXServerControl2, 379
ReconfigServerDone, IHXReconfigServerResponse, 329
Redirect, IHXPlayerController, 276
RegisterGUID, IHXPPVDatabase, 300
RelativeEnter, IHXOptimizedScheduler, 240
RelativeEnter, IHXScheduler, 374
RelativeEnter, IHXThreadSafeScheduler, 432
Release, IUnknown, 495
ReleaseSingleSiteUser, IHXMultiInstanceSiteUserSupplier, 231
Reload, IHXConfigFile, 84
ReloadPlugins, IHXPluginReloader, 295
Remove, IHXAsyncOSelection, 20
Remove, IHXAudioDeviceManager, 26
Remove, IHXFileRemove, 146
Remove, IHXOptimizedScheduler, 240
Remove, IHXScheduler, 374
Remove, IHXThreadSafeScheduler, 433
RemoveAdviseSink, IHXPlayer, 265
RemoveAdviseSink, IHXVolume, 477
RemoveAll, IHXUpgradeCollection, 448
RemoveChildPlayer, IHXPlayerNavigator, 280
RemoveDryNotification, IHXAudioStream2, 40
RemoveErrorSink, IHXErrorSinkControl, 128
RemoveFinalHook, IHXAudioDeviceManager, 27
RemoveGroup, IHXGroupManager, 179
RemoveHook, IHXEventHookMgr, 131
RemoveParentPlayer, IHXPlayerNavigator, 280
RemovePassiveSiteWatcher, IHXSite2, 388
RemovePostMixHook, IHXAudioPlayer, 34
RemovePrefetchSink, IHXPrefetch, 315
RemovePreMixHook, IHXAudioStream, 38
RemovePrincipal, IHXAuthenticationDBManager, 46
RemovePrincipalDone, IHXAuthenticationDBManagerResponse, 48
RemoveRedirect, IHXRedirectDBManager, 332
RemoveRedirectDone, IHXRedirectDBManagerResponse, 334
RemoveSink, IHXGroupManager, 179
RemoveSink, IHXPlayerSinkControl, 281
RemoveSite, IHXSiteManager, 394
RemoveStreamInfoResponse, IHXAudioPlayer, 34
RemoveStreamSink, IHXASMSocket, 12
RemoveTrack, IHXGroup, 173
RemoveUser, IHXPPVDatabase, 301
Index for Volume 2

Rename, IHXFileRename, 147
Replace, IHXAudioDeviceManager, 27
ReplaceCurr, IHXKeyValueListIter, 203
ReplaceCurr, IHXKeyValueListIterOneKey, 204
Report, IHXErrorMessages, 123
ReportQualityOfService, IHXStream, 418
RequestUpgrade, IHXUpgradeHandler, 450
Reset, IHXAsyncEnumAuthenticationDB, 15
Reset, IHXAudioDevice, 24
ResetDone, IHXAsyncEnumAuthenticationDBResponse, 17
ResetRoot, IHXPreferences2, 312
ResponseReady, IHXChallengeResponse, 69
ResponseReady, IHXClientAuthResponse, 76
RestartServer, IHXServerControl2, 379
Resume, IHXAudioDevice, 24
Resume, IHXPlayerController, 277
RevertData, IHXDataRevert, 105
RevertedDataReady, IHXDataRevertResponse, 106
RevertedFileHeaderReady, IHXDataRevertResponse, 107
RevertedStreamHeaderReady, IHXDataRevertResponse, 107
RevertFileHeader, IHXDataRevert, 105
RevertStreamHeader, IHXDataRevert, 105
RevokeAllPermissions, IHXPPVDatabase, 301
RevokeAllPermissions, IHXPPVDBManager, 304
RevokeAllPermissionsDone, IHXPPVDBManagerResponse, 308
RevokePermission, IHXPPVDatabase, 301
RevokePermissions, IHXPPVDBManager, 305
RevokePermissionsDone, IHXPPVDBManagerResponse, 308
SendImage, IHXLiveRealPix, 211
Set, IHXBuffer, 66
Set, IHXPacket, 246
Set, IHXRTPacket, 371
SetActiveBuf, IHXActivePropUser, 2
SetActiveBuf, IHXActiveRegistry, 8
SetActiveBufDone, IHXActivePropUserResponse, 4
SetActiveInt, IHXActivePropUser, 3
SetActiveInt, IHXActiveRegistry, 8
SetActiveIntDone, IHXActivePropUserResponse, 5
SetActiveStr, IHXActivePropUser, 3
SetActiveStr, IHXActiveRegistry, 8
SetActiveStrDone, IHXActivePropUserResponse, 6
SetAsActive, IHXActiveRegistry, 9
SetAsInactive, IHXActiveRegistry, 9
SetAsLost, IHXPacket, 246
SetAsLost, IHXRTPacket, 371
SetAudioPushdown, IHXAudioPushdown, 36
SetBackgroundColor, IHXLiveText, 222
SetBufByld, IHXRegistry, 346
SetBufByName, IHXRegistry, 346
SetCacheSize, IHXPlugin2Handler, 288
SetClientContext, IHXPlayer, 265
SetConfiguration, IHXObjectConfiguration, 237
SetConnectionTimeout, IHXConnectionlessControl, 86
SetContext, IHXObjectConfiguration, 238
SetCredentials, IHXAuthenticationDBManager, 47
SetCredentialsDone, IHXAuthenticationDBManagerResponse, 49
SetCurrentGroup, IHXGroupManager, 179
SetCursor, IHXSite2, 388
SetDoLooping, IHXLiveText, 223
SetEncoderDone, IHXLiveText, 223
SetEvent, IHXSiteEvent, 409
SetFilename, IHXConfigFile, 85
SetFinalHook, IHXAudioDeviceManager, 27
SetFlags, IHXLiveText2, 227
SetFocus, IHXFocusNavigation, 165
SetFocusEllipse, IHXDrawFocus, 112
SetFocusPolygon, IHXDrawFocus, 112
SetFocusRect, IHXDrawFocus, 112
SetGranularity, IHXStream, 418
SetGroupProperties, IHXGroup, 173
SetGUIDForPrincipalID, IHXGUIDDBManager, 187
SetGUIDForPrincipalIDDone, IHXGUIDDBManagerResponse, 189
SetHyperlinkInfo, IHXLiveText, 223
SetID, IHXLoadBalancedListen, 229
SetIntByld, IHXRegistry, 346

S
Save, IHXConfigFile, 85
SaveAs, IHXConfigFile, 85
Seek, IHXFileFormatObject, 139
Seek, IHXFileObject, 145
Seek, IHXPlayer, 265
Seek, IHXSyncFileFormatObject, 424
Seek, IHXTrack, 436
SeekDone, IHXFileResponse, 149
SeekDone, IHXFormatResponse, 167
SeekDone, IHXTrackSink, 438
Select, IHXClientEngineSelector, 79
SendChallenge, IHXChallenge, 68
SendControlBuffer, IHXDataConvertResponse, 100
SendControlBuffer, IHXDataRevertResponse, 107
SendEffect, IHXLiveRealPix, 211

571
SetIntByName, IHXRegistry, 347
SetKeyboardFocus, IHXKeyboardFocus, 199
SetMinimumPreroll, IHXPlayer2, 267
SetMulticastTransportConverter, IHXDataConvert, 98
SetMute, IHXVolume, 477
SetOption, IHXSetSocketOption, 382
SetOptionBuffer, IHXOptions, 241
SetOptionCString, IHXOptions, 242
SetOptionULONG32, IHXOptions, 242
SetPacketFormat, IHXPacketFormat, 247
SetParentPlayer, IHXPlayerNavigator, 280
SetPassword, IHXPPVDatabase, 301
SetPermissions, IHXPPVDBManager, 305
SetPermissionsDone, IHXPPVDBManagerResponse, 308
SetPlayerController, IHXPlayerConnectionAdviseSink, 270
SetPosition, IHXSite, 386
SetPresentationProperties, IHXGroupManager, 179
SetPropertyBuffer, IHXValues, 458
SetPropertyCString, IHXValues, 458
SetPropertyULONG32, IHXValues, 459
SetRecordStartPos, IHXRecordTimeline, 330
SetRegistryID, IHXPlayerConnectionAdviseSink, 271
SetRepeatedEvent, IHXSltaEvent, 410
SetRequest, IHXRequest, 365
SetRequestHeaders, IHXRequest, 364
SetRequiredPlugins, IHXPlugin2Handler, 289
SetReserveLimit, IHXLoadBalancedListen, 229
SetResponse, IHXTCPSocket, 429
SetResponseHeaders, IHXRequest, 364
SetRTP, IHXRTPPacket, 371
SetSingleSiteUser, IHXMultiInstanceSiteUserSupplier, 231
SetSize, IHXBuffer, 66
SetSize, IHXSite, 386
SetSoundLevel, IHXTrack, 436
SetStatus, IHXStatusMessage, 415
SetStrByld, IHXRegistry, 347
SetStrByName, IHXRegistry, 347
SetStreamInfoResponse, IHXAudioPlayer, 34
SetStringAccessAsBufferByld, IHXRegistryAltStringHandling, 348
SetStyle, IHXDrawFocus, 113
SetTAC, IHXQSLTA, 198
SetTAC, IHSXLSLA, 407
SetTargetBandwidth, IHXQSLTA, 198
SetTargetBandwidth, IHXSLTA, 408
SetTextMotion, IHXLiveText, 223
SetTimeBetweenDryStreamResends, IHXLiveText2, 227
SetTimeOffset, IHXPacketTimeOffsetHandler, 254
SetTransportType, IHXTransportControl, 440
SetType, IHXLiveText, 224
Setup, IHXClientEngineSetup, 81
SetURL, IHXRequest, 364
SetVolume, IHXAudioDevice, 24
SetVolume, IHXVolume, 477
SetWatchByld, IHXPropWatch, 321
SetWatchByName, IHXPropWatch, 321
SetWatchOnRoot, IHXPropWatch, 322
SetWindowDimensions, IHXLiveText, 224
SetZOrder, IHXSite2, 389
ShowMenu, IHXContextMenu, 90
ShowSite, IHXSite2, 389
ShutdownServer, IHXServerControl, 378
SiteAdded, IHXEventHook, 129
SiteRemoved, IHXEventHook, 130
SitesNeeded, IHXSiteSupplier, 395
SitesNotNeeded, IHXSiteSupplier, 396
SizeChanged, IHXPassiveSiteWatcher, 256
Skip, IHXAsyncEnumAuthenticationDB, 16
SkipDone, IHXAsyncEnumAuthenticationDBResponse, 18
SourceReady, IHXFileViewSourceResponse, 162
Start, IHXProcess, 318
Start, IHXUserImpersonation, 453
StartEncoder, IHXLiveRealPix, 211
StartHook, IHXPacketHookHelper, 249
StartHook, IHXPacketHookManager, 251
StartPackets, IHXBroadcastFormatObject, 61
StartPackets, IHXEncoder, 117
StartPackets, IHXRawSourceObject, 328
StartStream, IHXRenderer, 362
Stat, IHXFileStat, 150
StatDone, IHXFileStatResponse, 151
Stop, IHXPlayer, 265
Stop, IHXTrack, 436
Stop, IHXUserImpersonation, 453
StopDone, IHXTrackSink, 438
StopEncoder, IHXLiveRealPix, 212
StopHook, IHXPacketHookHelper, 249
StopHook, IHXPacketHookManager, 251
StopPackets, IHXBroadcastFormatObject, 61
StopPackets, IHXEncoder, 117
StopPackets, IHXRawSourceObject, 328
StreamDone, IHXEncoderResponse, 121
StreamDone, IHXFormatResponse, 167
StreamDone, IHXRawSinkObject, 326
StreamHeaderReady, IHXEncoderResponse, 121
StreamHeaderReady, IHXFormatResponse, 167
StreamHeaderReady, IHXRawSinkObject, 326
Subscribe, IHXASMSource, 10
Index for Volume 2

Subscribe, IHXASMStream, 13

T
  TestFullScreen, IHXSiteFullScreen, 393
  TrackAdded, IHXGroupSink, 182
  TrackRemoved, IHXGroupSink, 183
  TrackStarted, IHXGroupSink, 183
  TrackStopped, IHXGroupSink, 183

U
  UNIX
    server forks, 380
  Unsubscribe, IHXASMSource, 10
  Unsubscribe, IHXASMStream, 13
  Update, IHXStatistics, 414
  UpdateFileObjectStats, IHXFastFileStats, 133
  UpdateSiteWindow, IHXSite2, 389
  UseWordwrap, IHXLiveText, 225

V
  ValidateUser, IHXPPVDatabase, 302
  ViewRights, IHXViewDRMRights, 472
  ViewSourceURLReady, IHXViewSourceURLResponse, 475

W
  WantWrite, IHXTCPSocket, 430
  Write, IHXAudioDevice, 25
  Write, IHXAudioStream, 39
  Write, IHXFileObject, 145
  Write, IHXTCPSocket, 430
  Write, IHXUDPsocket, 445
  WriteDone, IHXFileResponse, 149
  WriteKey, IHXRegConfig, 335
  WritePref, IHXPreferences, 311
  WriteReady, IHXTCPResponse, 426
  WriteTo, IHXUDPsocket, 445

X
  XML, 480