

Omnissa Horizon Session Enhancement SDK Programming Guide

For Horizon 7, Horizon 8, and Horizon Cloud Service

Omnissa Horizon Session Enhancement SDK 4.0

You can find the most up-to-date technical documentation on the Omnissa website at: <https://docs.omnissa.com/>

Contents

Preface	8
Intended Audience	8
Overview of the Omnissa Horizon Session Enhancement SDK	9
Introduction to the API.....	9
What's New in Omnissa Horizon Session Enhancement SDK 4.0	10
Key Concepts of Omnissa Horizon Session Enhancement.....	11
Omnissa Horizon Session Enhancement Program Flow	12
RPC API.....	17
Overlay API.....	19
Screen Capture API.....	20
Virtual Channel and Side Channel Security	20
Installation.....	21
Sample Code	22
Data Types and Error Codes	23
Data Types.....	23
Error Codes.....	31
Channel Interaction Functions	35
v1.Broadcast.....	36
v1.Connect	36
v1.Disconnect.....	37
v1.GetChannelState.....	37
v1.GetConnectionState	38
v1.Poll.....	38
v1.RegisterChannelNotifySink	39
v1.RegisterObserver	39
v1.ThreadInitialize	40
v1.ThreadUninitialize.....	40
v1.UnregisterChannelNotifySink	41
v1.UnregisterObserver	41

v2.GetSessionType	42
v2.SwitchToStreamDataMode	42
v3.Poll.....	43
RPC Functions.....	45
v1.AppendNamedParam	46
v1.AppendNamedReturnVal.....	47
v1.AppendParam.....	47
v1.AppendReturnVal	48
v1.CreateChannelObject	48
v1.CreateContext.....	49
v1.DestroyChannelObject	50
v1.DestroyContext.....	50
v1.GetCommand	51
v1.GetId.....	51
v1.GetMinimalStreamDataSize.....	52
v1.GetNamedCommand.....	52
v1.GetNamedParam	53
v1.GetNamedReturnVal	54
v1.GetObjectName.....	54
v1.GetObjectState	55
v1.GetParam.....	55
v1.GetParamCount	56
v1.GetReturnCode.....	56
v1.GetReturnVal	57
v1.GetReturnValCount	58
v1.GetStreamDataHeaderTail.....	58
v1.GetStreamDataHeaderTailSize.....	59
v1.GetStreamDataInfo.....	59
v1.GetStreamDataSize.....	60
v1.Invoke	61
v1.SetCommand.....	61
v1.SetNamedCommand	62

v1.SetReturnCode	63
v1.VariantClear	63
v1.VariantCopy	64
v1.VariantFromBlob	64
v1.VariantFromChar	65
v1.VariantFromDouble	65
v1.VariantFromFloat	66
v1.VariantFromInt32	66
v1.VariantFromInt64	67
v1.VariantFromShort	67
v1.VariantFromStr	68
v1.VariantFromUInt32	68
v1.VariantFromUInt64	69
v1.VariantFromUShort	69
v1.VariantInit	70
v2.FreeStreamDataPayload	70
v2.GetStreamData	71
v2.GetStreamDataInfo	72
v2.IsSideChannelAvailable	72
v2.RequestSideChannel	73
v2.SetOps	73
v3.CreateContext	75
v3.GetObjectOptions	75
v4.GetObjectStateByName	76
VDPOverlay Functions	78
VDPOverlayGuest_Interface Functions	78
VDPOverlayClient_Interface Functions	95
VDPScreenCapture Functions	116
v1.Init	116
v1.Exit	116
v1.GetLocalTopology	117
v1.GetRemoteTopology	118

v1.GetHostWindowByRect	118
v1.GetHostWindowByPoint.....	119
v1.MapLocalToRemoteRect.....	120
v1.MapRemoteToLocalRect.....	120
v2.IsApplicationMode	121
v2.ReadBackScreen	122
v2.ReadBackRelease.....	122
v3.IsReadBackWindowSupported	123
v3.ReadBackWindowBegin.....	123
v3.ReadBackWindowEnd	125
v3.ReadBackWindow.....	125
v4.GetReadBackCapabilities.....	126
v4.ReadBackWindowBegin.....	127
v4.GetReadBackWindowInfo.....	128
v4.RegisterForReadBackRequests	129
v4.UnregisterForReadBackRequests	130
v4.ReadBackRequestUpdate	130
Channel Sinks	132
v1.OnChannelStateChanged.....	132
v1.OnConnectionStateChanged	132
v1.OnPeerObjectCreated	133
RPC Sinks	134
v1.OnAbort.....	134
v1.OnDone	134
v1.OnInvoke	135
v1.OnObjectStateChanged	135
Overlay Sinks	137
VDPOverlayGuest_Sink Functions	137
VDPOverlayClient_Sink Functions	139
Screen Capture Sinks.....	147

v1.OnTopologyChanged	147
v3.OnReadBackWindowReady	147
v3.OnReadBackWindowDestroyed.....	148
v3.OnReadBackWindowRemoteError	148
v4.OnReadBackWindowLocationChanged	149
v4.OnReadBackWindowImageChanged	149
v4.OnReadBackRequest	150

Preface

This document, *Omnissa Horizon Session Enhancement SDK Programming Guide*, provides information about developing applications using the Omnissa Horizon[®] Session Enhancement Application Programming Interface (API). Omnissa provides several software development kit (SDK) products, each of which targets different developer communities and platforms.

Intended Audience

This guide is intended for software developers who want to create applications that are used remotely over connections to Omnissa Horizon 7, Omnissa Horizon 8, or Omnissa Horizon Cloud Service deployments.

NOTE: Within this document, file paths, registry keys, and similar code-related items are revised to reflect the SDK's use with the Omnissa stack. References to items that were for Horizon versions 2406 and earlier versions contain asterisks (***) to redact the previous name.

Overview of the Omnissa Horizon Session Enhancement SDK

With the Omnissa Horizon Session Enhancement Software Development Kit (SDK), you can develop applications that communicate between a client and a remote desktop over a Horizon connection using the Blast Extreme or PCoIP display protocol.

The SDK contains resources such as documentation, include files, and code samples, to help you develop applications that use the Omnissa Horizon Session Enhancement API.

Introduction to the API

The Omnissa Horizon Session Enhancement API specifies how the client side and the desktop side of an application can communicate over a Horizon connection. All interactions with the API are asynchronous.

Any software that uses the Horizon Session Enhancement API must have two components:

- Application

This is the code that runs on a remote desktop.

- Plug-In

This is the code that is installed on a client.

The Horizon Session Enhancement API consists of two distinct APIs:

- Remote Procedure Call (RPC) API

The RPC API provides an asynchronous, callback-driven communication channel between applications that run on a remote desktop and a plug-in that runs on a client. The RPC API also handles the marshaling and unmarshaling of parameters.

- Overlay API

The Overlay API solves the problem of displaying rendered images on the client. Images appear to a user as a local window on the remote desktop.

OpenSSL Issue

The Horizon Session Enhancement API dynamically loads the OpenSSL library to implement its security features. If a software's application and plug-in components also dynamically load the OpenSSL library in the same way as the Horizon Session Enhancement API, you must adhere to the following rules to prevent crashes or exceptions.

- 1 Plug-in components must not call the `CRYPTO_set_locking_callback()`, `CRYPTO_set_id_callback()`, and `CRYPTO_set_add_lock_callback()` functions since the `remotemks` service already calls these functions.

- 2 Application components must set up the preceding callbacks before loading the Horizon Session Enhancement API library. They must also ensure that those callbacks are valid before unloading the Horizon Session Enhancement API library.
- 3 If the code is shared by both the plug-in and application components, you must call the preceding three callback functions if `CRYPTO_get_locking_callback()` returns NULL. You must call those three functions to set callbacks at the same time.

Supported Versions of Horizon Software

The Horizon Session Enhancement API supports the following types of deployments.

- Horizon pods running Horizon 7 or Horizon 8 (Horizon 2006 and later) software.

To support the latest features and interfaces of the Horizon Session Enhancement API, ensure that your Horizon pods are running on the latest release version of Horizon 7 or Horizon 8.

- First-gen Horizon Cloud Service on Microsoft Azure pods.

To support the latest features and interfaces of the Horizon Session Enhancement API, ensure that your first-gen Horizon Cloud pods are running on the latest release version of the pod manifest.

- Horizon Cloud Service next-gen Azure Edge deployments.

Note If your deployments are running on an older release version of Horizon software or of the first-gen Horizon Cloud Service on Microsoft Azure pod manifest, some features and interfaces of the Horizon Session Enhancement API are not supported.

Supported Client Operating Systems

The Horizon Session Enhancement API supports all Windows, Linux, and Mac operating systems that the Horizon Client software supports. For more information about supported operating systems, see the [Omnissa Horizon Client Documentation](#).

What's New in Omnissa Horizon Session Enhancement SDK 4.0

The following list summarizes the new features and changes found in version 4.0 of the Omnissa Horizon Session Enhancement SDK.

- This version of the SDK supports the new Omnissa-branded stack and is also backwards compatible with the older version as fallback.
- A new screen capture interface has been introduced to assist with capturing the contents of the Horizon client window

Key Concepts of Omnissa Horizon Session Enhancement

To effectively use the Omnissa Horizon Session Enhancement API, it is important to become familiar with the key concepts in Horizon Session Enhancement.

Connection

A connection refers to a Horizon session over the Blast Extreme or PCoIP protocol. You cannot alter a connection through the Horizon Session Enhancement API, but you can determine the current state of a connection. If a connection is not in the connected state, no action can be taken with the API. You can receive notification of a change in a connection's state using `VDPService_ChannelNotifySink` through the `v1.OnConnectionStateChanged` callback. You can also retrieve the current state of a connection using the `v1.GetConnectionState` method that is found in the `VDPService_ChannelInterface` API.

Channel

A channel represents the link between a remote application and a local plug-in. The state of a channel is not necessarily the same as the state of a connection.

You can receive notification of a change in the state of a channel through the `VDPService_ChannelNotifySink` function that you register with the channel. The `v1.OnChannelStateChanged` callback delivers the state change. You can query the current state of a channel using the `v1.GetChannelState` method in `VDPService_ChannelInterface`.

Side Channel

A side channel represents an additional link between a remote application and a local plug-in. A side channel belongs to a channel object and is set up via channel. A side channel can only be established after a channel object is connected. A side channel is designed to reduce application response time when there is network congestion in the main channel. For example, an application can use the main channel to transfer real-time control messages and use the side channel to transfer large amounts of user data.

Channel Context

A channel context is a wrapper for the parameters and return values of a remote call. A channel context holds all of the information for the receiver of a remote call to determine which method is requested. Interaction with the channel context is done using `VDPRPC_ChannelContextInterface`.

Overlay

An overlay is a window or image that is displayed over another so that the image or window overlay appears to be part of the underlying UI. This is typically done for video that plays locally, but needs to appear as if it is playing on the remote desktop.

Remote Procedure Call

A remote procedure call (RPC) is an invocation of a method on a non-local machine. Typically, the remote machine publishes a set of methods that it responds to, and the client invokes the methods through some channel. A call to `v1.Invoke` initiates an RPC.

Sink

A sink is a structure of function pointers and is used to communicate asynchronously with user code via callbacks. Each API call has one or more sets of sinks. The user must register the sinks to receive the necessary callbacks that give the user information about state changes or events.

Variant

To ease cross-platform communication, all parameters that are used with the VDP RPC API are wrapped in the `VDP_RPC_VARIANT` data type. This data type contains an identifier that indicates the type of data in the structure and the data itself. The use of variants is done through `VDPRPC_VariantInterface`.

OmniSSA Horizon Session Enhancement Program Flow

A typical Horizon Session Enhancement program flow involves the initialization of an application, a plug-in, threads, and a channel. It also includes sink registration, the calling of RPC and Overlay API methods, and shutting down.

Application Initialization

The user controls the startup of the remote side of the Horizon Session Enhancement system.

Upon application launch, the user code calls the `VDPService_ServerInit` method and gets the `VDP_SERVICE_QUERY_INTERFACE` structure. The user code then calls the `QueryInterface()` method to fetch all the interfaces that it needs to do its work.

Note If `QueryInterface()` returns `FALSE`, your Horizon software version does not support the function interface that you are trying to fetch.

Plug-In Initialization

On the local side, it is the Horizon Session Enhancement system that initializes the plug-in code. In the `VDPService_PluginInit` call, the user code must store the passed-in reference to the `VDP_SERVICE_QUERY_INTERFACE` structure and use it to request all the interfaces that it needs. At this point the user code is only loaded.

Once the matching application for the loaded plugin starts, `VDPService_PluginCreateInstance` is called. In this callback, the user may return a pointer that is returned in each callback, so that the user code can maintain its state.

To match a plug-in and an application, `VDPService` calls the plug-in's `VDPService_PluginGetTokenName` method and compares the string that is returned with the string that is given by the application. Before returning from the `VDPService_PluginCreateInstance` callback, the user code must call `v1.Connect` in `VDPService_ChannelInterface`.

Note Due to a limitation in the underlying protocol used, the TokenName variable must be less than 16 bytes in length.

Sink Registration

To receive callbacks from the Horizon Session Enhancement system, you must register sinks for different notifications. The first sink to register is `VDPService_ChannelNotifySink`. This sink notifies you of changes to the connection state, the channel state, and when the application has created an object. For more information about object creation, see [Channel Object](#). To register the sink, use the `v1.RegisterChannelNotifySink` method in `VDPService_ChannelInterface`. After the sink is registered, you receive a handle for that sink that you can use to unregister the sink. You must register `VDPService_ChannelNotifySink` before you call `v1.Connect` to ensure that you receive a notification when the channel is available.

After you register `VDPService_ChannelNotifySink`, you most likely will not receive a callback for a connection state change. This is because by the time the application or plug-in is started, the connection is likely to be in the connected state. To confirm that the connection is in the proper state prior to any actions, use the `GetConnectionState` method.

In addition to `VDPService_ChannelNotifySink`, the following sinks exist:

- `VDPRPC_ObjectNotifySink`
This is for individual channel objects.
- `VDPRPC_RequestCallback`
This is for callbacks for each RPC call.
- `VDPOverlayGuest_Sink`
This is for important overlay notifications for the guest.
- `VDPOverlayClient_Sink`
These are for important overlay notifications for the client.

Thread Initialization

On the application side, the main thread is the one that the user calls `VDPService_ServerInit` on. On the plug-in side, the main thread is the one that the `VDPService_PluginCreateInstance` callback is received on. For other threads, you must call `ThreadInitialize` before you call any other method in the RPC APIs or the Overlay APIs.

If a thread is no longer needed, you must uninitialize it by calling the `v1.ThreadUninitialize` method.

Channel

For communication to occur, the channel between the application and the plug-in must be active. To initialize the channel connection, call the `v1.Connect` method. It must be called on both sides of the connection for each channel. To shut down a channel, call the `v1.Disconnect` method.

After you call `v1.Disconnect`, or whenever the channel is in a disconnected state, you must free all your channel objects using the `v1.DestroyChannelObject` method. If the channel is connected again, you must recreate any required objects.

Query Interface

`QueryInterface()` returns an interface, or a structure of function pointers. Both applications and plug-ins must call `QueryInterface()` to retrieve the necessary interfaces.

The query interface data type `VDP_SERVICE_QUERY_INTERFACE` is a structure that is defined in `vdpService.h`. The application and the plug-in receive a reference to this structure differently. The structure has two members: a version attribute, and a function pointer. The version attribute notifies the user's application which version of the APIs are available. The function pointer is how the user's code will access the other APIs in the system. The function pointer has the following definition.

```
Bool (*QueryInterface) (const GUID *iid, void *iface);
```

The `QueryInterface()` function fetches the functions that the user needs to interact with the Horizon Session Enhancement API. The following table lists the GUIDs that are defined by Horizon Session Enhancement and the function lists that the GUIDs return.

Note If `QueryInterface()` returns `FALSE`, your Horizon software version does not support the function interface that you are trying to fetch.

Table 1-1. Horizon Session Enhancement GUIDs – Only showing the latest GUID for each interface

GUID	Returned Function List	Version	Header File
GUID_VDPService_ChannelInterface_V1	VDPService_ChannelInterface	v1	vdpService_interfaces.h
GUID_VDPService_ChannelInterface_V2	VDPService_ChannelInterface	v2	vdpService_interfaces.h
GUID_VDPService_ChannelInterface_V3	VDPService_ChannelInterface	v3	vdpService_interfaces.h
GUID_VDPService_ChannelInterface_V4	VDPService_ChannelInterface	v4	vdpService_interfaces.h
GUID_VDPRPC_ChannelObjectInterface_V1	VDPRPC_ChannelObjectInterface	v1	vdprpc_interfaces.h
GUID_VDPRPC_ChannelObjectInterface_V2	VDPRPC_ChannelObjectInterface	v2	vdprpc_interfaces.h
GUID_VDPRPC_ChannelObjectInterface_V3	VDPRPC_ChannelObjectInterface	v3	vdprpc_interfaces.h
GUID_VDPRPC_ChannelObjectInterface_V4	VDPRPC_ChannelObjectInterface	v4	vdprpc_interfaces.h
GUID_VDPRPC_ChannelContextInterface_V1	VDRPC_ChanneContextInterface	v1	vdprpc_interfaces.h
GUID_VDPRPC_ChannelContextInterface_V2	VDRPC_ChanneContextInterface	v2	vdprpc_interfaces.h
GUID_VDPRPC_VariantInterface_V1	VDPRPC_VariantInterface	v1	vdprpc_interface.h

GUID	Returned Function List	Version	Header File
GUID_VDPService_ServerInterface_V1	VDPService_ServerInterface	v1	vdpService_interfaces.h
GUID_VDPService_LocalJobInterface_V1	VDPService_LocalJobInterface	v1	vdpService_interfaces.h
GUID_VDPService_ObserverInterface_V1	VDPService_ObserverInterface	v1	vdpService_interfaces.h
GUID_VDPRPC_StreamDataInterface_V1	VDPRPC_StreamDataInterface	v1	vdprpc_interfaces.h
GUID_VDPRPC_StreamDataInterface_V2	VDPRPC_StreamDataInterface	v2	vdprpc_interfaces.h
GUID_VDPOverlay_GuestInterface_V1	VDPOverlay_GuestInterface	v1	vdpOverlay.h
GUID_VDPOverlay_GuestInterface_V2	VDPOverlay_GuestInterface	v2	vdpOverlay.h
GUID_VDPOverlay_GuestInterface_V3	VDPOverlay_GuestInterface	v3	vdpOverlay.h
GUID_VDPOverlay_GuestInterface_V4	VDPOverlay_GuestInterface	v4	vdpOverlay.h
GUID_VDPOverlay_ClientInterface_V1	VDPOverlay_ClientInterface	v1	vdpOverlay.h
GUID_VDPOverlay_ClientInterface_V2	VDPOverlay_ClientInterface	v2	vdpOverlay.h
GUID_VDPOverlay_ClientInterface_V3	VDPOverlay_ClientInterface	v3	vdpOverlay.h
GUID_VDPOverlay_ClientInterface_V4	VDPOverlay_ClientInterface	v4	vdpOverlay.h
GUID_VDPOverlay_ClientInterface_V5	VDPOverlay_ClientInterface	v5	vdpOverlay.h
GUID_VDPScreenCapture_Interface_V1	VDPScreenCapture_Interface	v1	vdpScreenCapture.h
GUID_VDPScreenCapture_Interface_V2	VDPScreenCapture_Interface	v2	vdpScreenCapture.h
GUID_VDPScreenCapture_Interface_V3	VDPScreenCapture_Interface	v3	vdpScreenCapture.h
GUID_VDPScreenCapture_Interface_V4	VDPScreenCapture_Interface	v4	vdpScreenCapture.h

The following sample code shows how to request an interface.

```
VDP_SERVICE_QUERY_INTERFACE qi;
VDPService_ChannelInterface ci;
qi.QueryInterface(&GUID_VDPService_ChannelInterface_V1, &ci);
```

Application

The user launches the application, which is the component that runs on the remote desktop. After the Application starts and vdpService.dll is loaded, the application calls VDPService_ServerInit(). When the application exits, it must call VDPService_ServerExit().

The following table describes the two server functions.

Table 1-2. Horizon Session Enhancement Server Functions

Function	Description
VDPService_ServerInit	The application calls this function when it starts. It must pass an identifying string (the token) to the function. The function returns a pointer to VDP_SERVICE_QUERY_INTERFACE and the channel handle for this application, which uses the channel handle to initialize user threads.
VDPService_ServerExit	The application calls this function when it closes down.
VDPService_ServerInit2	Same as VDPService_ServerInit but for a different session. Caller needs to have sufficient privilege.
VDPService_ServerExit2	Same as VDPService_ServerExit but for a different session. Caller needs to have sufficient privilege.

The following sample code shows how an application initializes.

```
/* program startup ( _tWinMain for example ) */
VDP_SERVICE_QUERY_INTERFACE qi;
void *channelHandle;
VDPRPC_VariantInterface vi;
VDPOverlay_GuestInterface ogi;
/* other interfaces omitted */
VDPService_ServerInit("example" /* token */, &qi, &channelHandle);
qi.QueryInterface(&GUID_VDPRPC_VariantInterface_V1, &vi);
qi.QueryInterface(&GUID_VDPOverlay_GuestInterface_V1, &ogi);
/* ... */
```

Plug-in

The main difference between the plug-in and the application is that the Horizon software loads the code on the client. Therefore, the user-compiled code must be in a DLL or a shared object that the system loads. The plug-in must export the following functions.

Table 1-3. Horizon Session Enhancement Exported Plug-In Functions

Function	Description
VDPService_PluginInit	Invoked when the DLL or SO is loaded. The plug-in receives its reference to VDP_SERVICE_QUERY_INTERFACE.
VDPService_PluginInitWithPath	Similar to the VDPService_PluginInit function, but with an additional parameter for the absolute path to where the plug-in was loaded from the disk.
VDPService_PluginExit	Invoked when the library is unloaded and the user session ends.
VDPService_PluginGetTokenName	Horizon Session Management uses this function to match the plug-in with the application. The token that this function returns must match the token that the matching application passes to VDPService_ServerInit for communication to occur.

Function	Description
VDPService_PluginCreateInstance	Invoked when a new channel's identifier matches the one that VDPService_PluginGetTokenName returns. More than one instance of a plug-in may exist. Horizon Session management matches instances of the plug-in to the correct channel.
VDPService_PluginDestroyInstance	Called when the channel this plug-in instance runs on closes.

RPC API

With the RPC API, applications and plug-ins can communicate using virtual channels. You must perform all VDPService initialization steps before you call the RPC API.

Channel Object

Before communication can occur, a channel object with the same name must exist on both sides of the connection. To create a channel object, call the `v1.CreateChannelObject` method. It does not matter whether the channel object is created in the application or in the plug-in first. The initial state of the channel object is disconnected.

When a channel object is created, a message is sent to the other side of the connection, where the callback function `v1.OnPeerObjectCreated` is called. To create a matching object, call the `v1.CreateChannelObject` method. After the matching object is created, the state of the object on both sides is connected and both sides receive a state change notification.

After a channel object is connected, you can request a side channel for this object. There are two types of side channels: virtual side channel and TCP side channel. A virtual side channel is an additional virtual channel. A TCP side channel is a TCP socket connection between a client and an agent. When a side channel is established and both sides receive a state change notification, the state of the channel object will change to `VDP_RPC_OBJ_SIDE_CHANNEL_CONNECTED`.

For a TCP side channel, an agent application can switch to stream data mode to save resources. In stream data mode, all VDPService internal threads will be exited and an application has to use a TCP socket to send data to and receive data from a plug-in. RPC packets can be created and parsed by stream data APIs.

Invoke

After you create a channel object, you can invoke an RPC with the `v1.Invoke` method. You must make the `v1.Invoke` call on the thread that you create the object on, unless the object is configured to allow invoke on any thread.

The `v1.Invoke` call requires a `ChannelContext` data structure, which is a wrapper for all the data for the RPC, such as the command, parameters, and so on. You create a context with the `v1.CreateContext` function. After the context is created, add information for the RPC to the context with the `VDPRPC_ChannelContextInterface` methods and pass the context to `v1.Invoke`. Even though you create the context, if the call to `Invoke` succeeds, the API is responsible for freeing the context. This is because of the asynchronous nature of the API. When the call to `v1.Invoke` returns, the context might still be in use. If the call to `Invoke` fails, you are responsible for freeing the context using `v1.DestroyContext`.

Each channel context has a unique ID that you can retrieve with the `v1.GetId` method. The ID of a context that is passed to an `v1.Invoke` call is returned as the `contextId` parameter to the `v1.OnDone` and `v1.OnAbort` handlers. You can use the `contextId` to map the callbacks back to the original `v1.Invoke` call. The ID of the context (e.g. the one returned from `context.v1.GetId`) that is passed to the `v1.OnDone` and `v1.OnAbort` handlers represents the context ID from the other side of the connection this is why the originating context ID is passed in as a separate parameter.

Variant

All data that you add to a channel context must be in a `VDP_RPC_VARIANT` data structure. The following code sample shows how to add data to a variant and append it to a context.

```
VDP_RPC_VARIANT var;  
VDPRPC_VariantInterface variface;  
VDPRPC_ChannelContextInterface ctxtiface;  
void *contextHandle;  
  
// Call VariantInit() before using the variant  
// Failure to call VariantInit() can cause memory corruption issues  
variface.v1.VariantInit(&var);  
  
// Add the parameters to the context  
variface.v1.VariantFromInt32(&var, 32);  
ctxtiface.v1.AppendParam(contextHandle, &var);  
  
// The same variant can be used for multiple parameters  
variface.v1.VariantFromString(&var, "sample string");  
ctxtiface.v1.AppendNamedParam(contextHandle, "sample param", &var);  
  
// Call VariantClear() after all parameters are added to the context  
// Failure to call VariantClear() can lead to memory leaks  
variface.v1.VariantClear(&var);
```

You must call `v1.VariantInit` before using a variant to avoid causing memory corruption and you must call `v1.VariantClear` method when you are done with the variant to ensure that all resources are freed.

It is recommended that you use the `RPCVariant` C++ class included with the sample code which will automatically call `v1.VariantInit` and `v1.VariantClear` for you.

```
RPCPluginInstance* plugin;  
VDPRPC_VariantInterface variface;  
VDPRPC_ChannelContextInterface ctxtiface;  
void *contextHandle;  
  
RPCVariant var(plugin);  
  
// Add the parameters to the context  
variface.v1.VariantFromInt32(&var, 32);  
ctxtiface.v1.AppendParam(contextHandle, &var);
```

```
// The same variant can be used for multiple parameters
variface.v1.VariantFromString(&var, "sample string");
ctxtiface.v1.AppendNamedParam(contextHandle, "sample param", &var);
```

OnInvoke

On a successful `v1.Invoke` call, the peer object receives an `v1.OnInvoke` callback. In this callback you receive a channel context. The context contains all of the information for the call. To respond, add the appropriate return code and return values to the channel context, which is returned to the caller when the `v1.OnInvoke` call returns.

Application Shutdown

The application must call `VDPService_ServerExit`.

Plug-In Shutdown

The following functions are called during the plug-in shutdown process.

- `VDPService_PluginDestroyInstance` is called when the channel associated with the remote desktop application is closed. Each call to `VDPService_PluginCreateInstance` has a corresponding call to `VDPService_PluginDestroyInstance`.
- `VDPService_PluginExit` is called when the Horizon session ends, immediately before the plugin DLL is unloaded. The plug-in must free all resources and shut down.

Overlay API

With the Overlay API, you can overlay an image on top of another window or image. You typically do this to make video that is playing on the local client appear as if it is playing in a window on the remote desktop.

Guest Setup

To use the Overlay API, the first step is to initialize the guest interface by calling the `v1.Init` method. After a successful initialization, register the window that you want to overlay by calling the `v1.RegisterWindow` or `v3.RegisterWindow` method. The size and position of the registered window are tracked and sent to the client automatically. If the client does not reject the registered window, you receive the `v1.OnOverlayReady` callback. When you receive this callback, you call the `v1.EnableOverlay` function to display the overlay on the client.

When you are finished with the window, unregister it by calling `v1.UnregisterWindow`.

Client Setup

On the client, the first step is to initialize the interface by calling `v1.Init`, which returns a context ID. You use the ID to identify the plug-in instance. When the guest registers a window, the client is notified through the `v1.OnWindowRegistered` sink callback, which gives you a window ID. You need both the context ID and the window ID to update the overlay.

After you receive the `v1.OnOverlayReady` callback, you can start displaying your image by calling the `v1.Update` or `v2.Update` method. The API does not keep a copy of the image unless the `copyImage` flag is set to true. If you do not own the image resource or you need to free it, you must set the `copyImage` flag.

When you are finished with the overlay, call the `v1.Exit` method.

Screen Capture API

The `VDPScreenCapture` API is called from your Session Enhancement SDK plugin that runs on the Horizon Client and allows you to query information about the topology of the remote desktop as well as capture images of the remote desktop.

Screen Capture

Use the Screen Capture functions capture an image of the entire remote desktop or a specified area of the remote desktop.

Read Back Window

The Read Back Window functions allow you to track a window associated with a specific process on the remote desktop. You can get notifications when it resizes, moves, or updates. You can also capture the image of the window. These functions are only supported by the Blast protocol and are dependent on the version of both the Horizon Client and Horizon Agent.

Virtual Channel and Side Channel Security

This topic describes the security features of virtual channels and side channels which run over Horizon session connections.

Virtual Channel Security

Virtual channels run over the session connection that is established by the remote protocol and rely on security offered by the protocol. The communication over these supported protocols is highly secure and based on industry-recommended security practices. The endpoints negotiate the actual session encryption algorithm that is used by the selected protocol.

In addition, you can increase the security of virtual channels by configuring a list of allowed channels. This configuration allows only the channels in the list to be opened by legitimate requests and prevents all other channels from being opened. To create the allow list, add the channels as registry entries to the `.reg` file included with the SDK. For more information, see [OmniSSA Knowledge Base \(KB\) article 84156](#).

For detailed information about the types of security offered by the supported protocols, see [Understanding Client Connections](#).

To configure the cipher suites and protocols used by the client, follow the client-specific procedure described in [Configuring Security Protocols and Cipher Suites for Specific Client Types](#).

For information about the security features of Horizon Cloud Service next-gen, see the [Horizon Cloud Service – next-gen Security Overview](#) technical article.

Side Channel Security

Side channels rely on the Advanced Encryption Standard (AES) 128-cipher algorithm in Cipher Block Chaining (CBC) mode. The algorithm uses an explicit initialization Vector (IV) as a confidentiality mechanism within the context of the IPsec. The random number is generated on the remote desktop and exchanged through the main virtual channel which is secured by the protocol's security layer. This exchange does not require any negotiation for an SSL/TLS handshake. The application using the side channel must opt in to use the encryption.

The following API methods provide the implementation details for the encryption:

- `v1.CreateChannelObject()`: Use config flags to negotiate the encryption support between sender and receiver.
- `v3.GetObjectOptions()`: Use this function to verify whether both the sender and receiver support encryption.
- `v3.CreateContext()`: Use this function to create the encryption context before send and invoke events.

Installation

To use the Horizon Session Enhancement API, you must use `vdpService.dll`, which is installed by the Horizon agent software.

Remote Desktop

The file `vdpService.dll` must exist on the remote desktop. When you install the Horizon agent software, this file is automatically installed on the remote desktop. For the location of the installation directory, see the applicable registry:

- Agent versions 2412 and later - `HKLM\Software\Omnissa\Horizon\RemoteExperienceAgent\InstallPath`
- Agent versions 2406 and earlier - `HKLM\Software***** Inc.***** VDM\RemoteExperienceAgent\InstallPath`

The 64-bit version of `vdpService.dll` is installed under x64 in the same directory.

To load `vdpService.dll`, use the code in `vdpService_import.cpp` which is included with the SDK.

Client

The file `vdpService.dll` (.so, .dylib) already exists on the client system and is loaded by Horizon Client. Your plugin must not load `vdpService.dll`, because loading extra copies can cause problems.

Windows Client

Register the `vdpService` plugin to the applicable registry location:

- Client versions 2412 and later - `HKLM\Software\Omnissa\Horizon\VDPService\Plugins`
- Client versions 2406 and earlier - `HKLM\Software***** Inc.***** VDPService\Plugins`
- Use the function `VDPService_RegisterPlugin` which is included in the SDK to register your plugin. There is also a matching function to unregister the plugin `VDPService_UnRegisterPlugin` (see `helpers.cpp/.h`).

Linux Client

Copy the vdpService plugin to the applicable location:

- Client versions 2412 and later - /lib/.omnissa/vdpService or ~/.omnissa/vdpService
- Client versions 2406 and earlier - /lib/.*****/vdpService or ~/.*****/vdpService
- The Makefile that comes with the samples shows how to copy the .so library to the correct folder

Make sure that the plug-ins have the execute permission.

Mac Client

Copy the vdpService plugin to the applicable location:

- Client versions 2412 and later - /Library/.omnissa/vdpService or ~/.omnissa/vdpService
- Client versions 2406 and earlier - /Library/.*****/vdpService or ~/.*****/vdpService
- The Makefile that comes with the samples shows how to copy the .dylib library to the correct folder

Make sure that the plug-ins have the execute permission. The plugin must be signed with your Apple developer's ID certificate unless SIP is disabled.

Sample Code

The Horizon Session Enhancement SDK includes a directory called samples that contains examples of how to use the API.

Data Types and Error Codes

The Horizon Session Enhancement API has three groups of data types. The API also specifies error codes for various error conditions.

Data Types

The Horizon Session Enhancement API uses the data types VDP Service, VDP RPC, and Overlay.

VDP Service Data Types

Table 2-1. VDPService Data Types

Data Type	Description
VDPService_ConnectionState	This enum indicates the current state of the remote connection.
VDPService_ChannelState	This enum indicates the current state of a particular channel.
VDPService_SessionType	This enum indicates the type of the current session (Blast Extreme or PCoIP).

VDP RPC Data Types

The VDP RPC data types are for use with the VDP RPC API.

Table 2-2. VDP RPC Data Types

Data Type	Description
VDP_RPC_VARENUM	This enum indicates the type of data that is stored in a VDP_RPC_VARIANT.
VDP_RPC_BLOB	Stores data that does not fit in any predefined VDP_RPC_VARENUM. Because VDP Service sends the data as is, it cannot protect against changes in byte endianness or structure alignment and padding. Use care to avoid errors.
VDP_RPC_VARIANT	Wraps the data for the RPC calls. Any data that is sent with the Invoke call must be contained in a VDP_RPC_VARIANT.
VDPRPC_ObjectState	Represents the state of an object. Only objects in the VDP_RPC_OBJ_CONNECTED state can be used in the Invoke call.
VDPRPC_ObjectConfigurationFlags	Used to configure channel objects with ChannelObjectInterface.v1.CreateChannelObject.
VDPRPC_ChannelContextOps	Used to configure the channel contexts with ChannelContextInterface.v2.SetOps.
VDPRPC_SideChannelType	Virtual side channel or TCP side channel.

VDPOverlay Data Types

The VDPOverlay data types are for use with the Overlay API. They are found in vdpOverlay.h.

This first table provides the VDP Overlay Guest data types. The second table provides the VDP Overlay Client data types.

Table 2-3. VDPOverlayGuest Data Types

Data Type	Description
VDPOverlay_WindowId	An identifier that represents a remote or guest-side overlay. In earlier versions of the API, the windowId and the HWND were the same but in the current version they can be different.
VDPOverlay_HWND	A representation of the native OS window.
VDPOverlay_UserArgs	Parameter that is passed through to the callback on the remote side.
VDPOverlay_LayoutMode	This enum represents all of the different layouts that the VDP Overlay API supports.
VDPOverlay_Error	Returned by many of the Overlay functions. Indicates the results that may occur.
VDP_OVERLAY_INFO_STR_MAX_LEN	The maximum length, including the NULL terminator, of an information string rendered on top of an overlay. The value of this constant is set to 1024 bytes.

Table 2-4. VDPOverlayClient Data Types

Data Type	Description						
VDPOverlayClient_ContextId	Returned from VDPOverlayClient.v1.Init(). This ID is used in every call to the Client API.						
VDPOverlay_OverlayId	An identifier that represents a local or client-side overlay that doesn't map to a window in the remote desktop. An OverlayId can be used in any function that takes a WindowId but a WindowId can not be used as an OverlayId.						
VDPOverlayClient_OverlayInfo	<p>This structure is used in the call to VDPOverlayClient.v1/v2.GetInfo().</p> <p>In V1 the first member of VDPOverlayClient_OverlayInfo was cbSize which was set by the caller to determine the version of the struct. But doing that was not backward compatible. For example, a program written to V2 would return an error if it called GetInfo() because the size wouldn't be set correctly.</p> <p>Starting with V2 the first member of VDPOverlayClient_OverlayInfo is a version and is set by GetInfo() to the version of the function that filled the structure. For backward compatibility when calling v1.GetInfo() the caller must set version = VDP_OVERLAY_INFO_V1_SIZE before calling v1.GetInfo().</p>						
VDPOverlayClient_YUVImageData	<p>This structure is used in the call to VDPOverlayClient.v2.Update() when the image format is a YUV format.</p> <table><tr><th>Member</th><th>Description</th></tr><tr><td>pImage[3]</td><td>An array with a pointer to the pixels for each plane of the image.<ul style="list-style-type: none">BGRX - image[0] = BGRX plane, image[1] = NULL, image[2] = NULLYUV - image[0] = Y plane, image[1] = U plane, image[2] = V plane</td></tr><tr><td>pitch[3]</td><td>An array with the # of bytes, per row, for each plane of the image.</td></tr></table>	Member	Description	pImage[3]	An array with a pointer to the pixels for each plane of the image. <ul style="list-style-type: none">BGRX - image[0] = BGRX plane, image[1] = NULL, image[2] = NULLYUV - image[0] = Y plane, image[1] = U plane, image[2] = V plane	pitch[3]	An array with the # of bytes, per row, for each plane of the image.
Member	Description						
pImage[3]	An array with a pointer to the pixels for each plane of the image. <ul style="list-style-type: none">BGRX - image[0] = BGRX plane, image[1] = NULL, image[2] = NULLYUV - image[0] = Y plane, image[1] = U plane, image[2] = V plane						
pitch[3]	An array with the # of bytes, per row, for each plane of the image.						

Data Type	Description																
VDPOverlayClient_InfoStringProperties	<p>This structure is used in the calls to VDPOverlayClient.v4.GetInfoStringProperties() and VDPOverlayClient.v4.SetInfoStringProperties().</p> <p>V1 through V3 of VDPOverlayClient_Interface do not support this structure. The members of this structure are defined as follows.</p> <table> <tr> <th>Member</th><th>Description</th></tr> <tr> <td>v4.enabled</td><td>A Boolean that activates/deactivates the information string.</td></tr> <tr> <td>v4.fgColor</td><td>An uint32 specifying the foreground/text color used to render the information string. 0 specifies the default foreground color.</td></tr> <tr> <td>v4.bgColor</td><td>An uint32 specifying the background color used to render the information string. 0 specifies the default background color.</td></tr> <tr> <td>v4.xBox</td><td> An int32 specifying the horizontal distance between the background and the edge of the overlay. Positive numbers position the background on the left. Negative numbers position the background on the right. 0 uses the default margin. </td></tr> <tr> <td>v4.yBox</td><td> An int32 specifying the vertical distance between the background and the edge of the overlay. Positive numbers position the background on the top. Negative numbers position the background on the bottom. 0 uses the default margin. </td></tr> <tr> <td>v4.wBox</td><td> An int32 defining the width of the background. Positive numbers denote the maximum width of the background. The text will be scaled to fit in this width using the LETTERBOX_SHRINK_ONLY layout mode. Negative numbers denote an absolute width for the background. The text will be scaled to fit in this width using the LETTERBOX layout mode. 0 sizes the background to the width of the text. </td></tr> <tr> <td>v4.hBox</td><td> An int32 defining the height of the background. Positive numbers denote the maximum height of the background. The text will be scaled to fit in this height using the LETTERBOX_SHRINK_ONLY layout mode. Negative numbers denote an absolute height for the background. The text will be scaled to fit in this height using the LETTERBOX layout mode. 0 sizes the background to the height of the text. </td></tr> </table>	Member	Description	v4.enabled	A Boolean that activates/deactivates the information string.	v4.fgColor	An uint32 specifying the foreground/text color used to render the information string. 0 specifies the default foreground color.	v4.bgColor	An uint32 specifying the background color used to render the information string. 0 specifies the default background color.	v4.xBox	An int32 specifying the horizontal distance between the background and the edge of the overlay. Positive numbers position the background on the left. Negative numbers position the background on the right. 0 uses the default margin.	v4.yBox	An int32 specifying the vertical distance between the background and the edge of the overlay. Positive numbers position the background on the top. Negative numbers position the background on the bottom. 0 uses the default margin.	v4.wBox	An int32 defining the width of the background. Positive numbers denote the maximum width of the background. The text will be scaled to fit in this width using the LETTERBOX_SHRINK_ONLY layout mode. Negative numbers denote an absolute width for the background. The text will be scaled to fit in this width using the LETTERBOX layout mode. 0 sizes the background to the width of the text.	v4.hBox	An int32 defining the height of the background. Positive numbers denote the maximum height of the background. The text will be scaled to fit in this height using the LETTERBOX_SHRINK_ONLY layout mode. Negative numbers denote an absolute height for the background. The text will be scaled to fit in this height using the LETTERBOX layout mode. 0 sizes the background to the height of the text.
Member	Description																
v4.enabled	A Boolean that activates/deactivates the information string.																
v4.fgColor	An uint32 specifying the foreground/text color used to render the information string. 0 specifies the default foreground color.																
v4.bgColor	An uint32 specifying the background color used to render the information string. 0 specifies the default background color.																
v4.xBox	An int32 specifying the horizontal distance between the background and the edge of the overlay. Positive numbers position the background on the left. Negative numbers position the background on the right. 0 uses the default margin.																
v4.yBox	An int32 specifying the vertical distance between the background and the edge of the overlay. Positive numbers position the background on the top. Negative numbers position the background on the bottom. 0 uses the default margin.																
v4.wBox	An int32 defining the width of the background. Positive numbers denote the maximum width of the background. The text will be scaled to fit in this width using the LETTERBOX_SHRINK_ONLY layout mode. Negative numbers denote an absolute width for the background. The text will be scaled to fit in this width using the LETTERBOX layout mode. 0 sizes the background to the width of the text.																
v4.hBox	An int32 defining the height of the background. Positive numbers denote the maximum height of the background. The text will be scaled to fit in this height using the LETTERBOX_SHRINK_ONLY layout mode. Negative numbers denote an absolute height for the background. The text will be scaled to fit in this height using the LETTERBOX layout mode. 0 sizes the background to the height of the text.																
VDPOverlay_LayoutMode	This enum represents all of the different layouts that the VDP Overlay API supports.																
VDPOverlay_Error	Returned by many of the Overlay functions. Indicates the results that may occur.																
VDPOverlay_ImageFormat	<p>This enum defines the pixel format of an image passed to VDPOverlayClient_Interface.v2.Update(). Note that VDPOverlayClient_Interface.v1.Update() always assumes VDP_OVERLAY_BGRX formatted images.</p>																

VDPScreenCapture Data Types

The VDPScreenCapture data types are for use with the Screen Capture API. They are found in vdpScreenCapture.h.

Table 2-5. VDPScreenCapture Data Types

Data Type	Description								
VDPScreenCapture_ContextId	Returned from VDPScreenCapture.v1Init(). This ID is used in every call to the Screen Capture API. VDP_SCREEN_CAPTURE_ID_NONE is a value that denotes an invalid VDPScreenCapture_ContextId.								
VDPScreenCapture_Rect	This is used to hold a rectangle.								
VDPScreenCapture_HWND	Type used by the ReadBackWindow interface to hold a remote window handle.								
VDPScreenCapture_ReadBackWindowHandle	Returned from VDPScreenCapture.v3/v4.ReadBackWindowBegin(). This ID is used when calling ReadBackWindow functions. VDP_SCREEN_CAPTURE_READBACK_WINDOW_HANDLE_NONE is a value that denotes an invalid VDPScreenCapture_ReadBackWindowHandle.								
VDPScreenCapture_ImageHandle	Returned from VDPScreenCapture.v2.ReadBackScreen() or v3.ReadBackWindow() to refer to the image returned. VDP_SCREEN_CAPTURE_IMAGE_HANDLE_NONE is a value that denotes an invalid VDPScreenCapture_ImageHandle.								
VDPScreenCapture_ReadBackRequestId	Returned from VDPScreenCapture.v4.RegisterForReadBackRequests(). VDP_SCREEN_CAPTURE_READBACK_REQUEST_ID_NONE is a value that denotes an invalid VDPScreenCapture_ReadBackRequestId.								
VDPScreenCapture_ImageFormat	Used to denote the format of an image returned from VDPScreenCapture interface.								
	<table> <tr> <th>Member</th><th>Description</th></tr> <tr> <td>VDP_SCREEN_CAPTURE_BGRX</td><td>32-bit RGB image. The high byte is ignored</td></tr> <tr> <td>VDP_SCREEN_CAPTURE_YUVI420</td><td>YUV I420 image</td></tr> </table>	Member	Description	VDP_SCREEN_CAPTURE_BGRX	32-bit RGB image. The high byte is ignored	VDP_SCREEN_CAPTURE_YUVI420	YUV I420 image		
Member	Description								
VDP_SCREEN_CAPTURE_BGRX	32-bit RGB image. The high byte is ignored								
VDP_SCREEN_CAPTURE_YUVI420	YUV I420 image								
VDPScreenCapture_OverlayOptions	Determines how overlays are handled during a screen capture								
	<table> <tr> <th>Member</th><th>Description</th></tr> <tr> <td>VDP_SCREEN_CAPTURE_OVERLAY_INCLUDE_ALL</td><td>The image includes all overlays</td></tr> <tr> <td>VDP_SCREEN_CAPTURE_OVERLAY_EXCLUDE_ALL</td><td>The image excludes all overlays</td></tr> <tr> <td>VDP_SCREEN_CAPTURE_OVERLAY_EXCLUDE_FLAGGED</td><td>The image will exclude overlays marked with VDP_OVERLAY_UPDATE_FLAG_EXCLUDE_F</td></tr> </table>	Member	Description	VDP_SCREEN_CAPTURE_OVERLAY_INCLUDE_ALL	The image includes all overlays	VDP_SCREEN_CAPTURE_OVERLAY_EXCLUDE_ALL	The image excludes all overlays	VDP_SCREEN_CAPTURE_OVERLAY_EXCLUDE_FLAGGED	The image will exclude overlays marked with VDP_OVERLAY_UPDATE_FLAG_EXCLUDE_F
Member	Description								
VDP_SCREEN_CAPTURE_OVERLAY_INCLUDE_ALL	The image includes all overlays								
VDP_SCREEN_CAPTURE_OVERLAY_EXCLUDE_ALL	The image excludes all overlays								
VDP_SCREEN_CAPTURE_OVERLAY_EXCLUDE_FLAGGED	The image will exclude overlays marked with VDP_OVERLAY_UPDATE_FLAG_EXCLUDE_F								

	ROM_READ_BACK in the call to VDPOverlayClient_Interface.v2.Update()
--	------------------------------------------------------------------------

VDP_SCREEN_CAPTURE_TOPOLOGY_CHANGED flags

Flags that can be passed to topology changed callback registered via VDPScreenCapture_Sink.v1.OnTopologyChanged(). One or more of these flags will be set when the function is called.

Flag	Description
VDP_SCREEN_CAPTURE_LOCAL_TOPOLOGY_CHANGED	Denotes that the local topology has changed.
VDP_SCREEN_CAPTURE_REMOTE_TOPOLOGY_CHANGED	Denotes that the remote topology has changed.
VDP_SCREEN_CAPTURE_HOST_WINDOWID_CHANGED	Denotes that the OS window ID of the Horizon Client window has changed.

VDP_SCREEN_CAPTURE_READBACK_CAPS flags

Flags returned from VDPScreenCapture.v4.GetReadBackCapabilities().

Flag	Description
VDP_SCREEN_CAPTURE_READBACK_CAPS_READBACK_SCREEN	Set if v2.ReadBackScreen() is available.
VDP_SCREEN_CAPTURE_READBACK_CAPS_READBACK_WINDOW	Set if v3.ReadBackWindow() is available.
VDP_SCREEN_CAPTURE_READBACK_CAPS_VISIBLE_AREA_ONLY	Set if v3.ReadBackWindow() w/ VISIBLE_AREA_ONLY is available.
VDP_SCREEN_CAPTURE_READBACK_CAPS_AUXILIARY_WINDOWS	Set if v4.ReadBackWindowBegin() w/ auxiliary windows is available.

VDP_SCREEN_CAPTURE_READBACK_BEGIN flags

Flags that can be passed to VDPScreenCapture_Interface.v3/v4.ReadBackWindowBegin().

Flag	Description
------	-------------

VDP_SCREEN_CAPTURE_READBACK_BEGIN_VISIBLE_AREA_ONLY	Clips the non-visible area of the window. The area of the window that is obscured by other windows is also obscured in the image returned.
VDP_SCREEN_CAPTURE_READBACK_BEGIN_LOCATION_CHANGED	Set if you want to be notified when the location of the readback window changes. Only a single event is queued at a time so the location tracking is closer to real time but you get fewer location points.
VDP_SCREEN_CAPTURE_READBACK_BEGIN_LOCATION_CHANGED_EX	Set if you want to be notified when the location of the readback window changes. Every location update is queued as a separate event. This can generate a lot of events but you get a more accurate movement path.
VDP_SCREEN_CAPTURE_READBACK_BEGIN_IMAGE_CHANGED	Set if you want to be notified when the image of the readback window changes.

VDP_SCREEN_CAPTURE_READBACK flags

Flags that can be used in VDPScreenCapture_ReadBackParameters.v3.flags.

Flag	Description
VDP_SCREEN_CAPTURE_READBACK_MAINTAIN_ASPECT_RATIO	Maintain the aspect ratio when scaling.
VDP_SCREEN_CAPTURE_READBACK_SHRINK_ONLY	VDPScreenCapture_ReadBackParameters.v2.scaledWidth/Height now represents a maximum size; i.e. scaling will only happen if the size of the image is larger than given scaled size.
VDP_SCREEN_CAPTURE_READBACK_RETURN_TEST_IMAGE	Returns a solid color test image.
VDP_SCREEN_CAPTURE_READBACK_WRITE_IMAGE_FILE	Writes the returned image to a file. This allows you to verify what the API is returning to your application. This is helpful when tracking down problems in the rendering pipeline. The image file is written as follows: <ul style="list-style-type: none"> ReadBackScreen -> %TEMP%\ReadBackScreen.png ReadBackWindow -> %TEMP%\ReadBackWindow-<ID>.png

VDPScreenCapture_ReadBackParameters

This structure is used in the call to VDPScreenCapture_Interface.v2.ReadBackScreen() and VDPScreenCapture_Interface.v3.ReadBackWindow().

Flag	Description
uint32 version	You must set the version number so that the API knows how big the structure is: <ul style="list-style-type: none"> VDP_SCREEN_CAPTURE_READ_BACK_PARAMETERS_V2 VDP_SCREEN_CAPTURE_READ_BACK_PARAMETERS_V3
VMRect v2.srcRect	When used with v2.ReadBackScreen(), the area within the remote desktop, in remote topology coordinates, to read. The area may not cross screen boundaries. Set srcRect to all 0s to read the entire first screen. When used with v3.ReadBackWindow(), the area within the remote window, in remote window coordinates, to read. e.g. X,Y == 0,0 is the top left corner of the remote window. Set srcRect to all 0s to read the entire window.
VDPScreenCapture_ImageFormat v2.format	The format of the returned image.

VDPScreenCapture_OverlayOptions v2.overlayOptions	Determine which overlays are included in the read back. See the flags defined in the VDPScreenCapture_OverlayOptions enum for more information Note: certain overlays are always excluded from the read back. See VDPOverlayClient_Interface.v3.SetRemoteWindow() for information.
bool v2.includeCursor	Determines if the cursor is included in the read back.
int32 v2.scaledWidth int32 v2.scaledHeight	The scaled width/height of the returned image. If either value is 0 no scaling takes places.
int32 v2.alignment	The alignment requirements, in bytes, of the returned image. The pointer to the start of each scan line will be aligned to be a multiple of this value. Pass 0 for a default / don't care value. The graphics library used by the Horizon Client must support the value.
uint32 v3.flags	See VDP_SCREEN_CAPTURE_READBACK flags definitions for more information.

VDPScreenCapture_ImageInfo

This structure is used in the call to VDPScreenCapture_Interface.v2.ReadBackScreen().

Flag	Description
uint32 version	You must set the version number so that the API knows how big the structure is: <ul style="list-style-type: none"> VDP_SCREEN_CAPTURE_IMAGE_INFO_V2 VDP_SCREEN_CAPTURE_IMAGE_INFO_V4
VDPScreenCapture_ImageHandle v2.hImage	A unique handle which is passed to VDPScreenCapture.v2.ReadBackRelease() to release resources allocated by the image. Use the pointers in the v2.image array to access the pixels.
void* v2.image[3]	An array with a pointer to the pixels for each plane of the image. <ul style="list-style-type: none"> BGRX - image[0] = BGRX plane, image[1] = NULL, image[2] = NULL YUV - image[0] = Y plane, image[1] = U plane, image[2] = V plane
int32 v2.pitch[3]	An array with the # of bytes, per row, for each plane of the image.
int32 v2.width int32 v2.height	The width/height of the image.
VDPScreenCapture_ImageFormat v2.format	The format of the image
VMRect v4.location	The location on the remote desktop, in remote topology coordinates, used to read back the image.

VDPScreenCapture_ReadBackWindowInfo

This structure is used in the call to VDPScreenCapture_Interface.v4.GetReadBackWindowInfo().

Flag	Description
------	-------------

uint32 version	You must set the version number so that the API knows how big the structure is. The version number will be updated to reflect what information the API returned: <ul style="list-style-type: none"> VDP_SCREEN_CAPTURE_READBACK_WINDOW_INFO_V4
VDPScreenCapture_ReadBackWindowHandle v4.hReadBackWindow	The readback window handle.
VDPScreenCapture_RemoteError v4.state	The current state of the readback window.
VDPScreenCapture_HWND v4.hWnd uint32 v4.processId uint32 v4.flags	The parameters passed to v3/v4.ReadBackWindowBegin().
const VDPScreenCapture_HWND* v4.hWndAux const uint32* v4.processIdAux int32 v4.auxWindowCount	The auxiliary window handles and process IDs passed to v4.ReadBackWindowBegin().
VMRect v4.location	The location on the remote desktop, in remote topology coordinates, used to read back the image.

VDPScreenCapture_ReadBackRequestParams

This structure is used in the call to VDPScreenCapture_Interface.v4.ReadBackRequestUpdate().

Flag	Description
uint32 version	You must set the version number so that the API knows how big the structure is: <ul style="list-style-type: none"> VDP_SCREEN_CAPTURE_READBACK_REQUEST_INFO_V4
void* v4.image[3]	An array with a pointer to the pixels for each plane of the image. <ul style="list-style-type: none"> BGRX - image[0] = BGRX plane, image[1] = NULL, image[2] = NULL YUV - image[0] = Y plane, image[1] = U plane, image[2] = V plane
int32 v4.pitch[3]	An array with the # of bytes, per row, for each plane of the image.
int32 v4.width int32 v4.height	The width/height of the image.
VDPOverlay_ImageFormat v4.format	The format of the image. VDPScreenCapture uses an overlay to display the image so the format must be one supported by the VDPOverlay API. See VDPOverlay_ImageFormat in vdpOverlay.h for more information.
VDPOverlay_LayoutMode v4.layoutMode	This is the layout mode of the overlay that defines how the image is scaled to fit into the destination rectangle. See VDPOverlay_LayoutMode in vdpOverlay.h for more information.
uint32 v4.layer	This is the layer of the overlay. When there are multiple overlays the one with the higher layer is on top. See VDPOverlay_SetLayer() in vdpOverlay.h for more information.
int32 v4.clipRgnNRects VDPScreenCapture_Rect* v4.clipRgnRects	An array of rectangles, in image coordinates, that define the clip region for the image. e.g. X,Y == 0,0 is the top left corner of the image. Pass 0/NULL if there isn't a clip region.
VMRect v4.dstRect	The area within the remote desktop, in remote topology coordinates, to place the image. e.g. X,Y == 0,0 is the top left corner of the remote desktop.

uint32 v4.imageFlags

Flags used when updating the image. See VDP_OVERLAY_UPDATE_FLAG in vdpOverlay.h for information.

Error Codes

The Horizon Session Enhancement API specifies codes to indicate errors.

RPC OnAbort Reason Error Codes

If the call to the `VDPRPC_ChannelObjectInterface.v1.OnInvoke()` method fails due to a Horizon Session Enhancement error, the supplied `OnAbort` method is called and the last parameter to this method contains one of the following error codes.

Table 2-5. RPC OnAbort Reason Error Codes

Code	Description
VDP_RPC_E_APARTMENT_UNINITIALIZED	This error occurs if the <code>OnInvoke</code> call is made on a thread that is not initialized to be used with the Horizon Session Management API.
VDP_RPC_E_APARTMENT_THREAD	This error occurs if the <code>OnInvoke</code> call involves an object that was not created on the calling thread and the object is not configured to allow <code>OnInvoke</code> calls on different threads.
VDP_RPC_E_OBJECT_NOT_CONNECTED	This error occurs if the object handle that is used for the <code>OnInvoke</code> call points to an object that is not connected. This error indicates that the peer object on the remote side is not yet created.
VDP_RPC_E_PARAMETER	One of the required parameters that is passed to the <code>OnInvoke</code> call is invalid.
VDP_RPC_E_MEMORY	The system fails to allocate the required memory to send the request.

VDPOverlay_Error Codes

If an error occurs, many of the methods that are defined in `vdpOverlay.h` return one of the following errors.

Table 2-6. VDPOverlay_Error Codes

Code	Description
VDP_OVERLAY_ERROR_SUCCESS	No error. The call is successful.
VDP_OVERLAY_ERROR_NOT_INITIALIZED	The call fails because the VDP Overlay components are not properly loaded in the Horizon environment.
VDP_OVERLAY_ERROR_ALREADY_INITIALIZED	This error is only returned from the <code>VDPOverlayGuest_Interface.v1.Init()</code> call. The guest Overlay system is already initialized.
VDP_OVERLAY_ERROR_INVALID_PARAMETER	One of the required parameters that is passed to the call is invalid.

VDP_OVERLAY_ERROR_ALLOCATION_ERROR	The system fails to allocate the required memory or system resource to handle the call.
VDP_OVERLAY_ERROR_NO_MORE_OVERLAYS	This error results from a failed attempt to register a window and may be received in the VDPOverlayGuest_Sink.v1.OnOverlayCreateError() or VDPOverlayClient.v2.CreateOverlay() callback. This error may be due to a client-side error. It can also occur if the call tries to register a window that is already registered with a different plug-in.
VDP_OVERLAY_ERROR_OVERLAY_REJECTED	This error results from a failed attempt to register a window and is returned in the reason field of the VDPOverlayGuest_Sink.v1.OnOverlayRejected() callback. This error occurs if the client does not accept the overlay registration request.
VDP_OVERLAY_ERROR_OVERLAY_NOT_READY	This error occurs when either VDPOverlayGuest_Interface.v1.EnableOverlay or VDPOverlayGuest_Interface.v1.DisableOverlay fails. It indicates that the registered window is not ready, that is, the VDPOverlayGuest_Sink.v1.OnOverlayReady() callback is not yet received.
VDP_OVERLAY_ERROR_WINDOW_NOT_REGISTERED	The window ID that is specified in the call is not yet registered. Many Overlay methods may return this error.
VDP_OVERLAY_ERROR_WINDOW_ALREADY_REGISTERED	The window is already registered. This error can be returned from the VDPOverlayGuest_Interface.v1.RegisterWindow() method.
VDP_OVERLAY_ERROR_NOT_LOCAL_OVERLAY	The overlayId of a guest-side overlay was passed to a function that can only be called on a local overlay.
VDP_OVERLAY_ERROR_HOST_OVERLAY_ERROR	There is an error with a low level library. This error code should be treated as similar to INVALID_PARAMETER.
VDP_OVERLAY_ERROR_NOT_SUPPORTED_BY_CLIENT	The version of the client-side VDP Overlay API does support the feature.

VDPScreenCapture_Error Codes

If an error occurs, many of the methods that are defined in vdpScreenCapture.h return one of the following errors.

Table 2-7. VDPScreenCapture_Error Codes

Code	Description
VDP_SCREEN_CAPTURE_ERROR_SUCCESS	No error. The call is successful.
VDP_SCREEN_CAPTURE_ERROR_HOST_NOT_READY	The call failed because the Screen Capture API is not done initializing.
VDP_SCREEN_CAPTURE_ERROR_HOST_VERSION_ERROR	This error is only if the version of vdpService.dll is not matched with the version of the Horizon client. This can only happen if the version of vdpService.dll being used is not the version that shipped with Horizon.
VDP_SCREEN_CAPTURE_ERROR_HOST_ERROR	There is an error with a low level library. This error code should be treated as similar to INVALID_PARAMETER.

VDP_SCREEN_CAPTURE_ERROR_INVALID_PARAMETER	One of the parameters passed into a function is not valid.
VDP_SCREEN_CAPTURE_ERROR_ALLOCATION_ERROR	The system fails to allocate the required memory or system resource to handle the call.
VDP_SCREEN_CAPTURE_ERROR_NOT_ALLOWED_IN_APP_MODE	Many of the function in the Screen Capture API can not be used when the Horizon client is remoting a single application. They can only be used when remoting the entire desktop.
VDP_SCREEN_CAPTURE_ERROR_READBACK_WINDOW_NOT_READY	The ReadBackWindow interface must initialize with the Horizon Agent on the remote desktop before it is ready.
VDP_SCREEN_CAPTURE_ERROR_READBACK_WINDOW_DESTROYED	The ReadBackWindow interface is tracking a window that no longer exists on the remote desktop.
VDP_SCREEN_CAPTURE_ERROR_UNSUPPORTED	The function that you are trying to use is not supported.
VDP_SCREEN_CAPTURE_ERROR_UNSUPPORTED_PROTOCOL	The function is not supported by the protocol. The ScreenCapture and ReadBackWindow interfaces are only supported by the Blast protocol.
VDP_SCREEN_CAPTURE_ERROR_UNSUPPORTED_AGENT	The function is not supported by the Horizon Agent on the remote desktop. This is usually because the version of the Horizon Agent does not support the ReadBackWindow interface.
VDP_SCREEN_CAPTURE_ERROR_UNSUPPORTED_OPTION	When calling v3.ReadBackWindow(), one of the options in the flags parameter is not supported by the Horizon Client.

VDPScreenCapture_RemoteError Codes

List of all possible error codes for VDPScreenCapture_Sink.v3.OnReadBackWindowRemoteError().

Table 2-8. VDPScreenCapture_RemoteError Codes

Code	Description
VDP_SCREEN_CAPTURE_REMOTE_ERROR_WINDOW_READY	No error. The remote window tracking is ready to return images.
VDP_SCREEN_CAPTURE_REMOTE_ERROR_WINDOW_NOT_READY	The remote window tracking failed because the Horizon Agent isn't ready to return images.
VDP_SCREEN_CAPTURE_REMOTE_ERROR_WINDOW_DESTROYED	The remote window tracking failed because the remote window was destroyed.
VDP_SCREEN_CAPTURE_REMOTE_ERROR_TRACKING_FULL	The remote window tracking failed because the Horizon Agent is tracking too many windows.
VDP_SCREEN_CAPTURE_REMOTE_ERROR_INVALID_WINDOW	The remote window tracking failed because the requested window does not exist.
VDP_SCREEN_CAPTURE_REMOTE_ERROR_INVALID_PID	The remote window tracking failed because the requested window does not belong to process specified.

VDP_SCREEN_CAPTURE_REMOTE_ERROR_UNSUPPORTED

The remote window tracking failed because the Horizon Agent does not support a requested capability.

VDP_SCREEN_CAPTURE_REMOTE_ERROR_CAPTURE_FAILED

The remote window tracking failed because the window on the remote desktop could not be captured.

Channel Interaction Functions

The Horizon Session Enhancement SDK contains the header file `vdpService_interfaces.h`. This file declares two structures of function pointers, `VDPService_ChannelInterface` and `VDPService_ObserverInterface`.

You can use the `VDPService_ChannelInterface` APIs to interact with the remote connection or channel. With `VDPService_ObserverInterface`, two components within the same process can communicate with each other.

`VDPService_ChannelInterface` consists of the following functions:

- `v1.Connect`
- `v1.Disconnect`
- `v1.GetChannelState`
- `v1.GetConnectionState`
- `v1.Poll`
- `v1.RegisterChannelNotifySink`
- `v1.ThreadInitialize`
- `v1.ThreadUninitialize`
- `v1.UnregisterChannelNotifySink`
- `v2.GetSessionType`
- `v2.SwitchToStreamdataMode`
- `v3.Poll`

`VDPService_ObserverInterface` consists of the following functions:

- `v1.Broadcast`
- `v1.RegisterObserver`
- `v1.UnregisterObserver`

This chapter includes the following topics:

- [v1.Broadcast](#)
- [v1.Connect](#)
- [v1.Disconnect](#)
- [v1.GetChannelState](#)
- [v1.GetConnectionState](#)
- [v1.Poll](#)
- [v1.RegisterChannelNotifySink](#)
- [v1.RegisterObserver](#)
- [v1.ThreadInitialize](#)
- [v1.ThreadUninitialize](#)
- [v1.UnregisterChannelNotifySink](#)
- [v1.UnregisterObserver](#)
- [v2.GetSessionType](#)
- [v2.SwitchToStreamDataMode](#)
- [v3.Poll](#)

v1.Broadcast

Broadcasts a given name's message to all observers. Basically, it will call all registered callbacks.

This function is a member of `VDPService_ObserverInterface`.

Method Signature

```
BOOL (*v1.Broadcast)(const char *name, const void *cookie, const void *data);
```

Parameters

Parameter	Description
name	The name of the message.
cookie	User-defined data. It can be as simple as request ID.
data	Message data.

Return Values

Value	Description
TRUE	Success
FALSE	Failure

v1.Connect

Starts the channel connection. You must call `v1.Connect` on both the application and the plug-in side, though the order does not matter. Call this method prior to exiting the `VDPService_PluginCreateInstance` callback.

This function is a member of `VDPService_ChannelInterface`.

Method Signature

```
Bool (*v1.Connect)(void);
```

Parameters

None

Return Values

Value	Description
TRUE	Call succeeded.

FALSE	Call failed.
-------	--------------

v1.Disconnect

Closes the underlying channel connection. You can call this method on either the plug-in or the application side.

This function is a member of `VDPService_ChannelInterface`.

Method Signature

```
Bool (*v1.Disconnect)(void);
```

Parameters

None

Return Values

Value	Description
TRUE	Call succeeded.
FALSE	Call failed.

v1.GetChannelState

Queries the current state of the channel connection between application and plug-in instances.

The channel to query is determined by the ID of the calling thread.

This function is a member of `VDPService_ChannelInterface`.

Method Signature

```
VDPService_ChannelState (*v1.GetChannelState)(void);
```

Parameters

None

Return Values

Value	Description
<code>VDP_SERVICE_CHAN_UNINITIALIZED</code>	The channel for this thread could not be found.
<code>VDP_SERVICE_CHAN_DISCONNECTED</code>	The channel is inactive.
<code>VDP_SERVICE_CHAN_PENDING</code>	The channel is open on the calling end, but not yet connected.

VDP_SERVICE_CHAN_CONNECTED

The channel is active.

v1.GetConnectionState

Used to query the state of the underlying user session. Note that depending on when a sink was registered, you might not receive a callback noting that the connection state has changed. Use this method to determine the state of the connection at any time.

This function is a member of VDPService_ChannelInterface.

Method Signature

```
VDPService_ConnectionState (*v1.GetConnectionState)(void);
```

Parameters

None

Return Values

Value	Description
VDP_SERVICE_CONN_UNINITIALIZED	The user session cannot be found.
VDP_SERVICE_CONN_DISCONNECTED	The user session is currently inactive.
VDP_SERVICE_CONN_PENDING	The user session is not connected, but active on the calling end.
VDP_SERVICE_CONN_CONNECTED	The user session is active.

v1.Poll

Allows the Horizon Session Enhancement system to process any waiting events. This call is required on any thread that the v1.ThreadInitialize call was made to so that the Horizon Session Enhancement system can function. If there are no waiting events, this call will just return.

Note All waiting events will be processed, so control may not be returned to you for some time. Most events will cause calls to registered sinks. Callbacks might be fired.

On Windows, if the thread uses its own message loop, using the method is not required.

This function is a member of VDPService_ChannelInterface.

Method Signature

```
void (*v1.Poll)(void);
```

Parameters

None

Return Values

None

v1.RegisterChannelNotifySink

Registers the given `VDPService_ChannelNotifySink` with the channel associated with the calling thread. You may register any number of sinks, and each will receive a callback when an event occurs.

The `sinkHandle` parameter will be set to the handle assigned to the given sink. This is used to unregister the sink with the channel.

This function is a member of `VDPService_ChannelInterface`.

Method Signature

```
Bool (*v1.RegisterChannelNotifySink)(const VDPService_ChannelNotifySink *sink, void *userData, uint32 *sinkHandle);
```

Parameters

Parameter	Description
<code>sink</code>	The sink to register with the channel.
<code>userData</code>	Data that will be passed into any callbacks to this sink. Can be NULL.
<code>sinkHandle</code>	Set to the handle assigned to this sink.

Return Values

Value	Description
TRUE	The sink was successfully registered.
FALSE	Sink registration failed.

v1.RegisterObserver

Registers an observer with the given name and callbacks.

This function is a member of `VDPService_ObserverInterface`.

Method Signature

```
VDPService_ObserverId (*v1.RegisterObserver)(const char *name, void *context, VdpServiceObserverCallback cb);
```

Parameters

Parameter	Description
name	The name of message caller is interested.
context	Context pointer caller want to passed in callback.
cb	Callback function when given name message is available.

Return Values

Value	Description
uint32	The ID of the registered observer or 0 (failed).

v1.ThreadInitialize

Initializes the thread for use with the Horizon Session Enhancement APIs. This method must be called on any thread that is not the main thread. Do not call this method on the thread that received the VDPService_PluginCreateInstance callback or that the VDPService_ServerInit call was made from.

This function is a member of VDPService_ChannelInterface.

Method Signature

```
Bool (*v1.ThreadInitialize)(void *channelHandle, uint32 unusedFlag);
```

Parameters

Parameter	Description
channelHandle	Represents the channel instance that this plug-in instance is running on. The channel handle is returned from the VDPService_ServerInit call or passed from the VDPService_PluginCreateInstance method.
unusedFlag	Currently unused.

Return Values

Value	Description
TRUE	The thread was successfully initialized.
FALSE	Thread initialization failed.

v1.ThreadUninitialize

Uninitializes the calling thread, freeing all resources associated with Horizon Session

Enhancement. No API calls must be made from this thread after this call. Only call this method on threads that had v1.ThreadInitialize invoked.

This function is a member of VDPService_ChannelInterface.

Method Signature

```
Bool (*v1.ThreadUninitialize)(void);
```

Parameters

None

Return Values

Value	Description
TRUE	The thread was successfully uninitialized.
FALSE	Thread uninitialization failed.

v1.UnregisterChannelNotifySink

Removes the sink associated with the given handle from the list of sinks that the channel associated with the calling thread will notify of Horizon Session Enhancement events.

This function is a member of VDPService_ChannelInterface.

Method Signature

```
(*v1.UnregisterChannelNotifySink)(uint32 sinkHandle);
```

Parameters

Parameter	Description
sinkHandle	The handle returned from v1.RegisterChannelNotifySink of the sink to be unregistered.

Return Values

Value	Description
TRUE	The sink that matches the given handle was successfully unregistered.
FALSE	The sink is still registered with the handle.

v1.UnregisterObserver

Unregisters an observer with the given name and callbacks.

This function is a member of VDPService_ObserverInterface.

Method Signature

```
BOOL (*v1.UnregisterObserver)(VDPService_ObserverId id);
```

Parameters

Parameter	Description
id	The observer id returned from v1.RegisterObserver.

Return Values

Value	Description
TRUE	Unregister succeeded.
FALSE	Unregister failed.

v2.GetSessionType

Gets the current virtual channel type.

This function is a member of VDPService_ChannelInterface.

Method Signature

```
VDPService_SessionType (*v2.GetSessionType)(void);
```

Parameters

None

Return Values

Value	Description
VDP_SERVICE_NONE_SESSION	Session type not determined yet.
VDP_SERVICE_PCOIP_SESSION	vdpservice is running in a PCoIP session.
VDP_SERVICE_BLAST_SESSION	vdpservice is running in a Blast Extreme session.

v2.SwitchToStreamDataMode

Switches vdpservice to TCP socket mode. This is an agent-only feature. In this mode, user can use output socket handle to send and receive data via a TCP socket handler. All internal vdpservice threads are terminated in order to

save resources. Only VDPRPC_StreamDataInterface and VDPService_ServerExit APIs can be called for the data processing and final clean-up.

This function is a member of VDPService_ChannelInterface.

Method Signature

```
BOOL (*v2.SwitchToStreamDataMode)(const char *tcpObjName, void *channelHandle, int *fd);
```

Parameters

Parameter	Description
tcpObjName	The name of the object which requested the TCP side channel.
channelHandle	Represents the channel interface that this plug-in is running on. The channelHandle is returned from the Vdpservice_ServerInit call or passed from the VDPService_PluginCreateInstance method.
fd	Output TCP socket handle.

Return Values

Value	Description
TRUE	Switching to stream data mode succeeded.
FALSE	Switching to stream data mode failed.

v3.Poll

Allows the Horizon Session Enhancement system to process any waiting events. This call is required on any thread that the v1.ThreadInitialize call was made to so that the Horizon Session Enhancement system can function. If there are no waiting events, this call will be blocked until the next event or timeout is reached.

Note All waiting events will be processed, so control may not be returned to you for some time. Most events will cause calls to registered sinks. Callbacks might be fired.

On Windows, if the thread uses its own message loop, using the method is not required.

This function is a member of VDPService_ChannelInterface.

Method Signature

```
void (*v3.Poll)(int timeout);
```

Parameters

Parameter	Description
timeout	The time limit after which the Poll method will return.

Return Values

None

RPC Functions

The `vdprpc_interfaces.h` header file included in the Horizon Session Enhancement SDK contains a set of structures of function pointers to send RPC messages.

This chapter includes the following topics:

- [v1.AppendNamedParam](#)
- [v1.AppendNamedReturnVal](#)
- [v1.AppendParam](#)
- [v1.AppendReturnVal](#)
- [v1.CreateChannelObject](#)
- [v1.CreateContext](#)
- [v1.DestroyChannelObject](#)
- [v1.DestroyContext](#)
- [v1.GetCommand](#)
- [v1.GetId](#)
- [v1.GetMinimalStreamDataSize](#)
- [v1.GetNamedCommand](#)
- [v1.GetNamedParam](#)
- [v1.GetNamedReturnVal](#)
- [v1.GetObjectName](#)
- [v1.GetObjectState](#)
- [v1.GetParam](#)
- [v1.GetParamCount](#)
- [v1.GetReturnCode](#)
- [v1.GetReturnVal](#)
- [v1.GetReturnValCount](#)
- [v1.GetStreamDataHeaderTail](#)
- [v1.GetStreamDataHeaderTailSize](#)
- [v1.GetStreamDataInfo](#)
- [v1.GetStreamDataSize](#)
- [v1.Invoke](#)
- [v1.SetCommand](#)
- [v1.SetNamedCommand](#)
- [v1.SetReturnCode](#)
- [v1.VariantClear](#)
- [v1.VariantCopy](#)
- [v1.VariantFromBlob](#)
- [v1.VariantFromChar](#)
- [v1.VariantFromDouble](#)
- [v1.VariantFromFloat](#)
- [v1.VariantFromInt32](#)
- [v1.VariantFromInt64](#)

- [v1.VariantFromShort](#)
- [v1.VariantFromStr](#)
- [v1.VariantFromUInt32](#)
- [v1.VariantFromUInt64](#)
- [v1.VariantFromUShort](#)
- [v1.VariantInit](#)
- [v2.FreeStreamDataPayload](#)
- [v2.GetStreamData](#)
- [v2.GetStreamDataInfo](#)
- [v2.IsSideChannelAvailable](#)
- [v2.RequestSideChannel](#)
- [v2.SetOps](#)
- [v3.CreateContext](#)
- [v3.GetObjectOptions](#)
- [v4.GetObjectStateByName](#)

v1.AppendNamedParam

Append the given Variant as a parameter to the given context and assign it a name. Note that the parameter is added to the end of the list with all parameters, even those without assigned names.

This function is a member of `VDPRPC_ChannelContextInterface`.

Method Signature

```
Bool (*v1.AppendNamedParam)(void *contextHandle, const char *name, const VDP_RPC_VARIANT *v);
```

Parameters

Parameter	Description
contextHandle	The context to append the parameter to.
name	Name to assign to the parameter.
v	The data for the new parameter.

Return Values

Value	Description
TRUE	Parameter successfully added.
FALSE	Unable to append the parameter to the given context.

v1.AppendNamedRetVal

Similar to v1.AppendRetVal but also assigns a name to the return value. The return value is added to the end of the list of all return values, even those without assigned names.

This function is a member of VDPRPC_ChannelContextInterface.

Method Signature

```
Bool (*v1.AppendNamedRetVal)(void *contextHandle, const char *name, const VDP_RPC_VARIANT *v);
```

Parameters

Parameter	Description
contextHandle	Context to add the return value to.
name	Name for the given return value.
v	Data for the return value.

Return Values

Value	Description
TRUE	Name and return value successfully added.
FALSE	Failed to add return value.

v1.AppendParam

Adds the given Variant to the context as a parameter for the method. Appends the parameter to the end of the list.

This function is a member of VDPRPC_ChannelContextInterface.

Method Signature

```
Bool (*v1.AppendParam)(void *contextHandle, const VDP_RPC_VARIANT *v);
```

Parameters

Parameter	Description
contextHandle	The handle for the context to add the parameter to.
v	Variant to store in the context. A copy of the data is made.

Return Values

Value	Description
TRUE	Data was successfully stored.
FALSE	Failed to append the Variant to the context.

v1.AppendReturnVal

Add the given Variant as a return value. The return values can be thought of as out parameters in a procedure call. The user can return any data desired here. The Variant is added to the end of the return value list.

This function is a member of VDPRPC_ChannelContextInterface.

Method Signature

```
Bool (*v1.AppendReturnVal)(void *contextHandle, const VDP_RPC_VARIANT *v);
```

Parameters

Parameter	Description
contextHandle	The handle for the context to append to.
v	Data to append.

Return Values

Value	Description
TRUE	Return value was successfully added.
FALSE	Failed to add the given Variant as a return value.

v1.CreateChannelObject

Creates a channel object with the given name. This call, with the same object name, must be made on both the plug-in and the application for communication to occur.

Objects begin in the VDP_RPC_OBJ_PENDING state. After the peer object is created, which might be prior to the call, the state goes to VDP_RPC_OBJ_CONNECTED. The sink registered with the object receives notifications of events involving the new object. A handle for the created object is returned in the objectHandle parameter

Note Objects must be used on the thread on which they are created, unless configured with the VDP_RPC_OBJ_CONFIG_INVOKE_ALLOW_ANY_THREAD flag. If this option is used, the user is responsible for thread safety.

This function is a member of VDPRPC_ChannelObjectInterface.

Method Signature

```
Bool (*v1.CreateChannelObject)(const char *name, const VDPRPC_ObjectNotifySink *sink, void *userData,
    VDPRPC_ObjectConfigurationFlags configFlags, void **objectHandle);
```

Parameters

Parameter	Description
name	Name for the created object.
sink	Sink to be registered with the new object.
userData	Data to be sent to all sink callbacks. Can be NULL.
configFlags	Set of configuration options for the new object.
objectHandle	Handle for the created object is stored here.

Return Values

Value	Description
TRUE	The object was successfully created.
FALSE	Creation of the object failed.

v1.CreateContext

Allocates and returns a new channel context to be used for an RPC.

This function is a member of VDPRPC_ChannelObjectInterface.

Method Signature

```
Bool (*v1.CreateContext)(void *objectHandle, void **ppcontextHandle);
```

Parameters

Parameter	Description
objectHandle	A handle for a valid channel object.
ppcontextHandle	A handle for the new channel context is returned here.

Return Values

Value	Description
TRUE	The new context was successfully created and returned.
FALSE	Context creation failed.

v1.DestroyChannelObject

Frees all resources associated with the given channel object.

This function is a member of V DPRPC_ChannelObjectInterface.

Method Signature

```
Bool (*v1.DestroyChannelObject)(void *objectHandle);
```

Parameters

Parameter	Description
objectHandle	The handle, returned from CreateChannelObject, for the object to destroy.

Return Values

Value	Description
TRUE	The object was successfully destroyed.
FALSE	Destruction of the object failed.

v1.DestroyContext

Frees all resources associated with a given context. Call this method only on contexts that you have created using CreateContext(). Only contexts that will not be used should be destroyed by the user.

This function is a member of V DPRPC_ChannelObjectInterface.

Method Signature

```
Bool (*v1.DestroyContext)(void *contextHandle);
```

Parameters

Parameter	Description
contextHandle	The handle for the context to destroy.

Return Values

Value	Description
TRUE	The new context was successfully destroyed.
FALSE	Destruction of the context failed.

v1.GetCommand

Queries the command code that was assigned to the given context. Use this method to determine the remote method that was being called. Use the SetCommand method to set the command code. If 0 is returned, use GetNamedCommand to fetch the command code.

This function is a member of VDPRPC_ChannelContextInterface.

Method Signature

```
uint32 (*v1.GetCommand)(void *contextHandle);
```

Parameters

Parameter	Description
contextHandle	The handle for the context to query.

Return Values

Value	Description
uint32	The uint32 command code set for this context. 0 indicates the command was not set as a uint32.

v1.GetId

Returns the unique ID for the given context. This ID can be used to map callbacks to the Invoke call that they refer to.

This function is a member of VDPRPC_ChannelContextInterface.

Method Signature

```
uint32 (*v1.GetId)(void *contextHandle);
```

Parameters

Parameter	Description
contextHandle	The handle for the context to be queried.

Return Values

Value	Description
uint32	The ID for the given context.

v1.GetMinimalStreamDataSize

Gets the minimal stream data size before checking the RPC packet length.

This function is a member of V DPRPC_StreamDataInterface.

Method Signature

```
int (*v1.GetMinimalStreamDataSize)(int fd);
```

Parameters

Parameter	Description
fd	A valid socket handle returned by v2.SwitchToStreamDataMode.

Return Values

Value	Description
uint32	The minimal size.

v1.GetNamedCommand

Gets the command code assigned to the given context as a string. If the command was not stored as a string, this method returns FALSE, and you must use the GetCommand method instead to get the command code.

This function is a member of V DPRPC_ChannelContextInterface.

Method Signature

```
Bool (*v1.GetNamedCommand)(void *contextHandle, char *buffer, int bufferSize);
```

Parameters

Parameter	Description
contextHandle	The handle for the context to query.
buffer	Out parameter that the name is to be stored in.
bufferSize	Size of the buffer to store the name.

Return Values

Value	Description
TRUE	Named command successfully returned.
FALSE	The command was not stored as a string.

v1.GetNamedParam

Fetch the parameter at the given index and return the name, if any, that was assigned to the parameter. If no name was given, the name parameter remains untouched.

This function is a member of VDPRPC_ChannelContextInterface.

Method Signature

```
Bool (*v1.GetNamedParam)(void *contextHandle, int index, char *name, int nameSize, VDP_RPC_VARIANT *copy);
```

Parameters

Parameter	Description
contextHandle	The context to fetch the parameter from.
index	The index of the parameter to return.
name	The buffer to store the assigned name in. Can be NULL if you are not interested in the name.
nameSize	Size of the passed-in buffer.
copy	Variant into which the parameter data is to be copied.

Return Values

Value	Description
TRUE	Parameter at the given index returned and name (if any) found.
FALSE	Unable to fetch the parameter and name at the given index.

v1.GetNamedReturnVal

Fetches the return value at the given index. Also returns the name assigned to the return value.

This returned value might be empty, for example, if the length of the name is 0.

This function is a member of `VDPRPC_ChannelContextInterface`.

Method Signature

```
Bool (*v1.GetNamedReturnVal)(void *contextHandle, int index, char *name, int nameSize, const VDP_RPC_VARIANT *v);
```

Parameters

Parameter	Description
contextHandle	Context to query.
index	Index of the return value to fetch.
name	Buffer to store the name into. Can be NULL.
nameSize	Size of the name buffer.
v	Variant to copy the return value data into.

Return Values

Value	Description
TRUE	Successfully fetched the return value and name at the given index.
FALSE	Failed to find the return value or the name.

v1.GetObjectName

Queries the given object for the name it was assigned at creation.

This function is a member of `VDPRPC_ChannelObjectInterface`.

Method Signature

```
Bool (*v1.GetObjectName)(void *objectHandle, char *buf, uint32 bufSize);
```

Parameters

Parameter	Description
objectHandle	The handle, returned from CreateChannelObject, for the object to query.
buf	The name of the object is stored in this parameter.
bufSize	Size of the passed-in buf.

Return Values

Value	Description
TRUE	The name was successfully returned.
FALSE	An error occurred and the name was not returned.

v1.GetObjectState

Queries the current state of the given object.

This function is a member of VDPRPC_ChannelObjectInterface.

Method Signature

```
VDPRPC_ObjectState (*v1.GetObjectState)(void *objectHandle);
```

Parameters

Parameter	Description
objectHandle	The handle, returned from CreateChannelObject, for the object to query.

Return Values

Value	Description
VDP_RPC_OBJ_UNINITIALIZED	Object with the given handle could not be found.
VDP_RPC_OBJ_DISCONNECTED	Matching peer object was destroyed.
VDP_RPC_OBJ_PENDING	Object created locally, waiting for other end to create a peer object.
VDP_RPC_OBJ_CONNECTED	Given object is connected to its peer on the other side of the channel.

v1.GetParam

Fetches the parameter at the given index. The parameter list index begins at zero.

This function is a member of VDPRPC_ChannelContextInterface.

Method Signature

```
Bool (*v1.GetParam)(void *contextHandle, int i, VDP_RPC_VARIANT *copy);
```

Parameters

Parameter	Description
contextHandle	The context to query.
i	Index of the parameter to fetch.
copy	Variant into which the parameter is to be copied.

Return Values

Value	Description
TRUE	Parameter at the given index was successfully returned.
FALSE	Unable to find parameter at the given index.

v1.GetParamCount

Returns the number of parameters appended to the given context.

This function is a member of VDPRPC_ChannelContextInterface.

Method Signature

```
int (*v1.GetParamCount)(void *contextHandle);
```

Parameters

Parameter	Description
contextHandle	The handle for the context to query.

Return Values

Value	Description
int	Number of parameters stored in the given context.

v1.GetReturnCode

Queries the return code of a Remote Procedure Call (RPC). The return code is meant to indicate the success or failure of the remote method call, or as an error code.

This function is a member of VDPRPC_ChannelContextInterface.

Method Signature

```
uint32 (*v1.GetReturnCode)(void *contextHandle);
```

Parameters

Parameter	Description
contextHandle	The handle of the context to query.

Return Values

Value	Description
uint32	Return code set for the given context.

v1.GetReturnVal

Fetches the return value at the given index. Index of the return values begin at zero.

This function is a member of VDPRPC_ChannelContextInterface.

Method Signature

```
Bool (*v1.GetReturnVal)(void *contextHandle, int i, const VDP_RPC_VARIANT *v);
```

Parameters

Parameter	Description
contextHandle	The context to query.
i	Index of the return value to fetch.
v	Variant into which the return value data is to be copied.

Return Values

Value	Description
TRUE	Return value successfully fetched.
FALSE	Failed to locate return value at the given index.

v1.GetReturnValCount

Returns the number of Variants stored in the given context as return values.

This function is a member of VDPRPC_ChannelContextInterface.

Method Signature

```
int (*v1.GetReturnValCount)(void *contextHandle);
```

Parameters

Parameter	Description
contextHandle	The context to query.

Return Values

Value	Description
int	Number of return values stored in the given context.

v1.GetStreamDataHeaderTail

Fetches the header and tail data for stream data mode to send via the TCP socket. This function is mainly for optimization by eliminating a memcpy.

This function is a member of VDPRPC_StreamDataInterface.

Method Signature

```
Bool (*v1.GetStreamDataHeaderTail)(int fd, int *reqId, int reqCmd, VDP_RPC_BLOB *blob, char *header, int headerBufLen, char *tail, int tailBufLen);
```

Parameters

Parameter	Description
fd	A valid socket handle return by v2.SwitchToStreamDataMode.
reqId	RPC request ID is returned here for the caller to track each request.
reqCmd	Request command.
blob	Blob data which will be sent using the TCP socket.
header	Buffer to hold header data.

headerBufSize	Header buffer size. Must be greater than or equal to the size returned by v1.GetStreamDataHeaderTailSize.
---------------	-----------------------------------------------------------------------------------------------------------

tail	Buffer to hold header data.
------	-----------------------------

tailBufLen	Tail buffer size. Must be greater than or equal to the size returned by v1.GetStreamDataHeaderTailSize.
------------	---------------------------------------------------------------------------------------------------------

Return Values

Value	Description
TRUE	Successfully obtained the header and the tail.
FALSE	Failure.

v1.GetStreamDataHeaderTailSize

Obtains the size of the header and the tail for stream data mode (TCP socket) if neither compression nor encryption is needed. Because stream data mode is an agent-only feature, data needs to be encapsulated in RPC format to the client. This API is used to calculate the size of the header and the tail.

This function is a member of V DPRPC_ StreamDataInterface.

Method Signature

```
Bool (*v1.GetStreamDataHeaderTailSize)(int fd, int dataSize, int *headerLen, int *tailLen);
```

Parameters

Parameter	Description
fd	A valid socket handle returned by SwitchToStreamDataMode.
dataSize	The size of the data that the client intends to send.
headerLen	The size of the header is returned here.
TailLen	The size of the tail is returned here.

Return Values

Value	Description
TRUE	Successfully obtained the sizes.
FALSE	Failure. Must be invalid socket handle.

v1.GetStreamDataInfo

Parses stream data information from received binary data.

This function is a member of VDPRPC_StreamDataInterface.

Method Signature

```
int (*v1.GetStreamDataInfo)(int fd, const char *recvData, int *reqId, int *reqType, int *reqCmd, VDP_RPC_BLOB *blob);
```

Parameters

Parameter	Description
fd	A valid socket handle returned by v2.SwitchToStreamDataMode.
recvData	Data that needs to be parsed.
reqId	RPC request ID is returned here.
reqType	RPC request type is returned here.
reqCmd	RPC request command is returned here.
blob	Blob data that is sent by client is returned here.

Return Values

Value	Description
TRUE	Successfully parsed recvData as an RPC packet.
FALSE	Failure.

v1.GetStreamDataSize

Gets the RPC packet length. The parameter recvData must have at least the minimal-size amount of data.

This function is a member of VDPRPC_StreamDataInterface.

Method Signature

```
int (*v1.GetStreamDataSize)(int fd, const char *recvData);
```

Parameters

Parameter	Description
fd	A valid socket handle returned by v2.SwitchToStreamDataMode.
recvData	Data that needs to be parsed.

Return Values

Value	Description
uint32	The size of the whole RPC packet.

v1.Invoke

Initiates an RPC between the given object and its peer on the other end of the channel.

This function is a member of `VDPRPC_ChannelObjectInterface`.

Method Signature

```
Bool (*v1.Invoke)(void *objectHandle, void *contextHandle, const VDPRPC_RequestCallback *callback, void *userData);
```

Parameters

Parameter	Description
objectHandle	Handle for the object to send the RPC through.
contextHandle	A handle for the context containing the data for this RPC callback.
callback	User-supplied callbacks to be used after the Invoke call.
userData	User-supplied data that will be passed to the callback methods. Can be NULL.

Return Values

Value	Description
TRUE	Invoke call succeeded and RPC was sent.
FALSE	No RPC was sent due to an error.

v1.SetCommand

Sets the command code for the given context. The command code represents the remote method that the context is meant to represent.

Note You can also store the command as a string using `SetNamedCommand`. However, you can only use one method. If you call `SetNamedCommand` after calling `SetCommand`, the `uint32` command code is overwritten. Do not use 0 as the command code because the Horizon Session Enhancement system uses 0 to indicate an error.

This function is a member of `VDPRPC_ChannelContextInterface`.

Method Signature

```
Bool (*v1.SetCommand)(void *contextHandle, uint32 command);
```

Parameters

Parameter	Description
contextHandle	The handle for the context to set.
command	The command code for the context.

Return Values

Value	Description
TRUE	Context command code was successfully set.
FALSE	Unable to set the command code.

v1.SetNamedCommand

Sets the command code for the given context with a name. You can either set the command as a uint32 (using SetCommand) or as a string, using this method. Use only one method. If you try to use both, the second command used will overwrite the previous command.

This function is a member of V DPRPC_ChannelContextInterface.

Method Signature

```
Bool (*v1.SetNamedCommand)(void *contextHandle, const char *command);
```

Parameters

Parameter	Description
contextHandle	The handle for the context to set.
command	The command string to use.

Return Values

Value	Description
TRUE	Command string was successfully set.
FALSE	Unable to set the command string.

v1.SetReturnCode

Sets the return code for the given context. This should be done in the OnInvoke callback. This value represents the success or failure of the remote call.

This function is a member of VDPRPC_ChannelContextInterface.

Method Signature

```
Bool (*v1.SetReturnCode)(void *contextHandle, uint32 code);
```

Parameters

Parameter	Description
contextHandle	The handle for the context to set.
code	Value for the return code.

Return Values

Value	Description
TRUE	Return code of the context set
FALSE	Unable to set the return code.

v1.VariantClear

Clears and frees any resources held by the given Variant. Call this method whenever you are finished with a Variant.

This function is a member of VDPRPC_VariantInterface.

Method Signature

```
Bool (*v1.VariantClear)(VDP_RPC_VARIANT *v);
```

Parameters

Parameter	Description
v	The variant to clear.

Return Values

Value	Description
TRUE	The Variant is returned to initialized state.

FALSE	The Variant is unchanged.
-------	---------------------------

v1.VariantCopy

Copies the data held from the Variant src to the Variant target. Any data held by target is overwritten. Any data previously held in target is freed before being overwritten with the data in src.

This function is a member of VDPRPC_VariantInterface.

Method Signature

```
Bool (*v1.VariantCopy)(VDP_RPC_VARIANT *target, const VDP_RPC_VARIANT *src);
```

Parameters

Parameter	Description
target	The variant to copy the data to.
src	The variant to copy the data from.

Return Values

Value	Description
TRUE	Copy succeeded.
FALSE	Copy failed. The target is unchanged.

v1.VariantFromBlob

Stores a copy of the given VDP_RPC_BLOB in the given Variant. Use this method only for data that does not fit any of the other types. Data is sent as-is, so changes in architecture (such as sending from the Linux client to the Windows guest) and differences in structure padding and alignment can cause problems with your data.

This function is a member of VDPRPC_VariantInterface.

Method Signature

```
Bool (*v1.VariantFromBlob)(VDP_RPC_VARIANT *v, VDP_RPC_BLOB *blob);
```

Parameters

Parameter	Description
v	The variant to set.
blob	The VDP_RPC_BLOB to copy.

Return Values

Value	Description
TRUE	The VDP_RPC_BLOB was successfully copied into the Variant.
FALSE	Setting the Variant failed.

v1.VariantFromChar

Stores the given char in the given Variant and sets the internal type properly.

This function is a member of VDPRPC_VariantInterface.

Method Signature

```
Bool (*v1.VariantFromChar)(VDP_RPC_VARIANT *v, char c);
```

Parameters

Parameter	Description
v	The variant to set.
c	The char to store.

Return Values

Value	Description
TRUE	The char was successfully stored in the Variant.
FALSE	Setting the Variant failed.

v1.VariantFromDouble

Stores the given double in the given Variant and sets the internal type properly.

This function is a member of VDPRPC_VariantInterface.

Method Signature

```
Bool (*v1.VariantFromDouble)(VDP_RPC_VARIANT *v, double d);
```

Parameters

Parameter	Description
v	The variant to set.

d	The double to store.
---	----------------------

Return Values

Value	Description
TRUE	The double was successfully stored in the Variant.
FALSE	Setting the Variant failed.

v1.VariantFromFloat

Stores the given float in the given Variant and sets the internal type properly.

Method Signature

```
Bool ((*v1.VariantFromFloat)(VDP_RPC_VARIANT *v, float f);
```

Parameters

Parameter	Description
v	The variant to set.
f	The float to store.

Return Values

Value	Description
TRUE	The float was successfully stored in the Variant.
FALSE	Setting the Variant failed.

v1.VariantFromInt32

Stores the given int32 in the given Variant and sets the internal type properly.

This function is a member of VDPRPC_VariantInterface.

Method Signature

```
Bool (*v1.VariantFromInt32)(VDP_RPC_VARIANT *v, int32 i);
```

Parameters

Parameter	Description
-----------	-------------

v	The variant to set.
i	The int32 to store.

Return Values

Value	Description
TRUE	The int32 was successfully stored in the Variant.
FALSE	Setting the Variant failed.

v1.VariantFromInt64

Stores the given int64 in the given Variant and sets the internal type properly.

Method Signature

```
Bool (*v1.VariantFromInt64)(VDP_RPC_VARIANT *v, int64 i);
```

Parameters

Parameter	Description
v	The variant to set.
i	The int64 to store.

Return Values

Value	Description
TRUE	The int64 was successfully stored in the Variant.
FALSE	Setting the Variant failed.

v1.VariantFromShort

Stores the given short in the given Variant and sets the internal type properly.

This function is a member of VDPRPC_VariantInterface.

Method Signature

```
Bool (*v1.VariantFromShort)(VDP_RPC_VARIANT *v, short s);
```

Parameters

Parameter	Description
-----------	-------------

v	The variant to set.
s	The short to store.

Return Values

Value	Description
TRUE	The short was successfully stored in the Variant.
FALSE	Setting the Variant failed.

v1.VariantFromStr

Stores a copy of the given const char * in the given Variant and sets the internal type properly.

Method Signature

```
Bool (*v1.VariantFromStr)(VDP_RPC_VARIANT *v, const char *str);
```

Parameters

Parameter	Description
v	The variant to set.
str	The const char * to copy.

Return Values

Value	Description
TRUE	The const char * was successfully copied into the Variant.
FALSE	Setting the Variant failed.

v1.VariantFromUInt32

Stores the given uint32 in the given Variant and sets the internal type properly.

This function is a member of VDPRPC_VariantInterface.

Method Signature

```
Bool (*v1.VariantFromUInt32)(VDP_RPC_VARIANT *v, uint32 ui);
```

Parameters

Parameter	Description
v	The variant to set.
ui	The uint32 to store.

Return Values

Value	Description
TRUE	The uint32 was successfully stored in the Variant.
FALSE	Setting the Variant failed.

v1.VariantFromUInt64

Stores the given uint64 in the given variant and sets the internal type properly.

Method Signature

```
Bool (*v1.VariantFromUInt64)(VDP_RPC_VARIANT *v, uint64 ui);
```

Parameters

Parameter	Description
v	The variant to set.
ui	The uint64 to store.

Return Values

Value	Description
TRUE	The uint64 was successfully stored in the variant.
FALSE	Setting the variant failed.

v1.VariantFromUShort

Stores the given unsigned short in the given Variant and sets the internal type properly.

This function is a member of VDPRPC_VariantInterface.

Method Signature

```
Bool (*v1.VariantFromUShort)(VDP_RPC_VARIANT *v, unsigned short us);
```

Parameters

Parameter	Description
v	The variant to set.
us	The unsigned short to store.

Return Values

Value	Description
TRUE	The unsigned short was successfully stored in the Variant.
FALSE	Setting the Variant failed.

v1.VariantInit

Initializes the given VDP_RPC_VARIANT. To prevent memory corruption issues, you must initialize a variant before using it.

This function is a member of VDPRPC_VariantInterface.

Method Signature

```
Bool (*v1.VariantInit)(VDP_RPC_VARIANT *v);
```

Parameters

Parameter	Description
v	The variant to be initialized.

Return Values

Value	Description
TRUE	The variant was successfully initialized.
FALSE	Initialization failed.

v2.FreeStreamDataPayload

Frees payload memory for the blob data that is returned by v2.GetStreamData or v2.GetStreamDataInfo.

This function is a member of VDPRPC_StreamDataInterface.

Method Signature

```
int (*v2.FreeStreamDataPayload)(VDP_RPC_BLOB *payload);
```

Parameters

Parameter	Description
payload	Blob data that needs to be freed.

Return Values

Value	Description
TRUE	Payload successfully freed.
FALSE	Failed to free payload data.

v2.GetStreamData

Obtains the stream data to send via the TCP socket. This API is used when data needs either compression or encryption. It also works if neither of them is needed, but the data involves one additional memory allocation and memcpy. Be sure to call v2.FreeStreamDataPayload to avoid a memory leak.

This function is a member of VDPRPC_StreamDataInterface.

Method Signature

```
int (*v2.GetStreamData)(int fd, uint32 ctxOptions, int *reqId, int reqCmd, VDP_RPC_BLOB *blob, VDP_RPC_BLOB *payload);
```

Parameters

Parameter	Description
fd	A valid socket handle returned by v2.SwitchToStreamDataMode.
ctxOptions	Compression and encryption options.
reqId	RPC request ID is returned here for caller to track each request.
reqCmd	Request command.
blob	Blob data that needs to be sent.
payload	Actual RPC packet data is returned in payload.

Return Values

Value	Description
TRUE	Payload creation succeeded.
FALSE	Payload creation failed.

v2.GetStreamDataInfo

Same as v1.GetStreamDataInfo except for one more parameter, bNeedCleanup, to indicate whether the blob data needs to be cleaned up. The size of recvData has to be greater than or equal to the size returned by v1.GetStreamDataSize.

This function is a member of VDPRPC_StreamDataInterface.

Method Signature

```
int (*v2.GetStreamDataInfo)(int fd, const char *recvData, int *reqId, int *reqType, int reqCmd, Bool *bNeedCleanup, VDP_RPC_BLOB *blob);
```

Parameters

Parameter	Description
fd	A valid socket handle returned by v2.SwitchToStreamDataMode.
recvData	Data that needs to be parsed.
reqId	RPC request ID is returned here.
reqType	RPC request type is returned here.
reqCmd	RPC request command is returned here.
bNeedCleanup	Boolean value is returned here to indicate if the blob data need to be freed by v2.FreeStreamDataPayload.
blob	Blob data that is sent from the client is returned here.

Return Values

Value	Description
TRUE	RecvData is parsed as RPC packet successfully.
FALSE	Failure.

v2.IsSideChannelAvailable

Determines whether a side channel of the given type is available for use by any channel object. Currently, only one object can use an available channel.

This function is a member of VDPRPC_ChannelObjectInterface.

Method Signature

```
Bool (*v2.IsSideChannelAvailable)(VDPRPC_SideChannelType type);
```


Parameters

Parameter	Description
type	Side channel type. Either virtual side channel (VDP_RPC_SIDE_CHANNEL_TYPE_PCOIP) or TCP side channel (VDP_RPC_SIDE_CHANNEL_TYPE_TCP)

Return Values

Value	Description
TRUE	Side channel of the given type is available.
FALSE	Side channel of the given type is not available.

v2.RequestSideChannel

Requests a particular type of side channel for a given object.

This function is a member of VDPRPC_ChannelObjectInterface.

Method Signature

```
Bool (*v2.RequestSideChannel)(void *objectHandle, VDPRPC_SideChannelType type, const char *token);
```

Parameters

Parameter	Description
objectHandle	Handle for the object.
type	The type of side channel being requested.
token	The name of the side channel to use. If NULL, the application token is used.

Return Values

Value	Description
TRUE	Request succeeded.
FALSE	Request failed.

v2.SetOps

Sets channel context options. The most common use is to set the RPC call in post mode, which does not expect any response for this channel context.

This function is a member of VDPRPC_ChannelContextInterface.

Method Signature

```
Bool (*v2.SetOps)(void *contextHandle, VDPRPC_ChannelContextOps option, const VDP_RPC_VARIANT *v);
```

Parameters

Parameter	Description
contextHandle	Handle for a valid channel context.
option	<p>Specifies how the RPC call should behave. This parameter can take the following VDPRPC_ChannelContextOps values:</p> <p>VDP_RPC_CHANNEL_CONTEXT_OPT_POST</p> <p>You must set this option before calling Invoke() on the sender side.</p> <p>If v is set to 0, the RPC is set to the default request mode. In request mode, a response is sent back from the peer and delivered to the application through the OnDone callback.</p> <p>If v is set to 1, the RPC is set to post mode. In post mode, the peer doesn't return a response and no OnDone callback is received for this RPC.</p> <p>VDP_RPC_CHANNEL_CONTEXT_OPT_BEGIN_ASYNC_RESULT</p> <p>You must set this option before returning from OnInvoke on the receiver side. This option has no effect on RPCs in post mode.</p> <p>If v is set to 0, the RPC operates in default synchronous mode. In synchronous mode, the response from the RPC is sent back to the sender immediately after OnInvoke returns.</p> <p>If v is set to 1, the RPC operates in asynchronous mode. In asynchronous mode, the response from the RPC is not sent until v2.SetOps is called with the VDP_RPC_CHANNEL_CONTEXT_OPT_END_ASYNC_RESULT option.</p> <p>RPCs are processed on a single thread through the application's OnInvoke callback. If one RPC call takes a long time to execute, other RPC calls will be blocked until it finishes executing. Asynchronous mode is a way to process an RPC in a different thread so that other RPCs can be processed in parallel.</p> <p>VDP_RPC_CHANNEL_CONTEXT_OPT_END_ASYNC_RESULT</p> <p>When you set this option, the result of an asynchronous RPC is immediately sent back to the peer.</p> <p>Do not set this option on synchronous RPCs.</p>
v	A uint32 that specifies whether a certain option is activated or deactivated. A value of 1 activates and a value of 0 deactivates the option.

Return Values

Value	Description
TRUE	Success
FALSE	Failure

v3.CreateContext

Same as v1.CreateContext but supports compression and encryption options.

This function is a member of VDPRPC_ChannelObjectInterface.

Method Signature

```
Bool (*v3.CreateContext)(void *objectHandle, uint32 options, void **ppcontextHandle);
```

Parameters

Parameter	Description
objectHandle	Handle for a valid channel object.
options	<p>Specifies whether compression and encryption will apply for this context.</p> <p>You can combine the uint32 values using the bitwise OR operator " ". However, you can select only one encryption option and one compression option. To get the values supported by the current channel, use v3.GetObjectOptions.</p> <p>The following available encryption options are supported on the TCP side channel only. Other side channel types transfer data over the main channel, which is already encrypted and secured by the remote protocol.</p> <ul style="list-style-type: none">• VDP_RPC_CRYPT_AES• VDP_RPC_CRYPT_SALSA <p>The following available compression options are supported on the main channel and all side channels.</p> <ul style="list-style-type: none">• VDP_RPC_COMP_SNAPPY• VDP_RPC_COMP_ZLIB• VDP_RPC_COMP_MSFT
ppcontextHandle	A handle of a new channel context is returned here.

Return Values

Value	Description
TRUE	A new context was successfully created and returned.
FALSE	Context creation failed.

v3.GetObjectOptions

Obtains the following object options after an object is created: (1) encryption and compression options which both sides agree on; and (2) side channel types which peer does not support.

This function is a member of VDPRPC_ChannelObjectInterface.

Method Signature

```
Bool (*v3.GetObjectOptions)(void *objectHandle, uint32 *options);
```

Parameters

Parameter	Description
objectHandle	Handle for the object.
options	<p>Returns a set of bits that determines which encryption and compression options are supported on this channel when calling <code>v3.CreateContext</code>. Bits not included in the returned set are reserved for internal use and can be disregarded.</p> <p>The available encryption options are as follows.</p> <ul style="list-style-type: none">• <code>VDP_RPC_CRYPT_AES</code>• <code>VDP_RPC_CRYPT_SALSA</code> <p>The available compression options are as follows.</p> <ul style="list-style-type: none">• <code>VDP_RPC_COMP_SNAPPY</code>• <code>VDP_RPC_COMP_ZLIB</code>• <code>VDP_RPC_COMP_MSFT</code>

Return Values

Value	Description
TRUE	Request succeeded.
FALSE	Request failed.

v4.GetObjectStateByName

Retrieves the current state of the given object based on the name of the object.

This function is a member of `VDPRPC_ChannelObjectInterface`.

Method Signature

```
VDPRPC_ObjectState (*v4.GetObjectStateByName)(const char *name);
```

Parameters

Parameter	Description
name	The name of the given object.

Return Values

Value	Description
VDP_RPC_OBJ_UNINITIALIZED	Object with the given handle could not be found.
VDP_RPC_OBJ_DISCONNECTED	Matching peer object was destroyed.
VDP_RPC_OBJ_PENDING	Object created locally, waiting for other end to create a peer object.
VDP_RPC_OBJ_CONNECTED	Given object is connected to its peer on the other side of the channel.

VDPOverlay Functions

The vdpOverlay.h header file defines the set of functions to use in order to support overlay functionality in an application. This chapter covers VDPOverlayGuest_Interface functions and VDPOverlayClient_Interface functions.

VDPOverlayGuest_Interface Functions

With VDPOverlayGuest_Interface functions, you can work with windows; enable and disable the client-side overlay; work with the layout mode for the overlay; send a message to the client-side plug-in; and release all allocated resources.

v1.DisableOverlay

Deactivates the client-side overlay. Deactivating the overlay is a lightweight way to hide the client-side overlay. Unlike v1.UnregisterWindow(), resources used to maintain the overlay are not released.

This function is a member of VDPOverlayGuest_Interface.

Method Signature

```
VDPOverlay_Error (*v1.DisableOverlay)(VDPOverlay_WindowId windowId, VDPOverlay_UserArgs userArgs);
```

Parameters

Parameter	Description
windowId	The operating system window identifier. It must be previously registered with VDPOverlayGuest_RegisterWindow().
userArgs	Data that is to be passed to the client-side plug-in when VDPOverlayClient_OverlayEnabled event is sent.

Return Values

Value	Description
VDP_OVERLAY_ERROR_SUCCESS	No error. The function was successful.
VDP_OVERLAY_ERROR_NOT_INITIALIZED	Overlay API was not initialized.
VDP_OVERLAY_ERROR_INVALID_PARAMETER	The windowId or userArgs parameter was invalid, or there was an error with msg.

v1.EnableOverlay

Enables the client-side overlay. Once the window is registered and ready, this function must be called to display the client-side overlay.

This function is a member of VDPOverlayGuest_Interface.

Method Signature

```
VDPOverlay_Error (*v1.EnableOverlay)(VDPOverlay_WindowId windowId, VDPOverlay_UserArgs userArgs);
```

Parameters

Parameter	Description
windowId	The operating system window identifier. It must be previously registered with VDPOverlayGuest_RegisterWindow().
userArgs	Data that is to be passed to the client-side plug-in when VDPOverlayClient_OverlayEnabled event is sent.

Return Values

Value	Description
VDP_OVERLAY_ERROR_SUCCESS	No error. The function was successful.
VDP_OVERLAY_ERROR_NOT_INITIALIZED	Overlay API was not initialized.
VDP_OVERLAY_ERROR_INVALID_PARAMETER	The windowId or userArgs parameter was invalid, or there was an error with msg.

v1.Exit for the Guest-Side Library

Frees all allocated resources held by the Horizon Session Enhancement Overlay APIs and unregisters all windows.

This function is a member of VDPOverlayGuest_Interface.

Method Signature

```
VDPOverlay_Error (*v1.Exit)(void);
```

Parameters

None

Return Values

Value	Description
VDP_OVERLAY_ERROR_SUCCESS	Overlay successfully shut down.
VDP_OVERLAY_ERROR_NOT_INITIALIZED	Overlay was never initialized.

v1.GetLayoutMode

Gets the current layout mode for the overlay. The layout mode is used to determine how an image is drawn (for example, scaled, cropped, and so on) when the size of the image does not match the size of the overlay.

This function is a member of VDPOverlayGuest_Interface.

Method Signature

```
VDPOverlay_Error (*v1.GetLayoutMode)(VDPOverlay_WindowId windowId, VDPOverlay_LayoutMode *pLayoutMode);
```

Parameters

Parameter	Description
windowId	The window ID of the overlay. It must have been previously registered with <code>VDPOverlayGuest_RegisterWindow()</code> .
pLayoutMode	Current layout mode is stored here.

Return Values

Value	Description
<code>VDP_OVERLAY_ERROR_SUCCESS</code>	Current layout mode was successfully retrieved.
<code>VDP_OVERLAY_ERROR_NOT_INITIALIZED</code>	Overlay API was not initialized.
<code>VDP_OVERLAY_ERROR_INVALID_PARAMETER</code>	<code>pLayoutMode</code> is NULL.
<code>VDP_OVERLAY_ERROR_WINDOW_NOT_REGISTERED</code>	The given <code>windowId</code> has not been registered with the Overlay API.

v1.Init for the Guest-Side Library

Initializes the guest-side overlay library. This must be the first overlay API function called.

This function is a member of `VDPOverlayGuest_Interface`.

Method Signature

```
VDPOverlay_Error (*v1.Init)(const VDPOverlayGuest_Sink* sink, void* userData);
```

Parameters

Parameter	Description
sink	Function pointers called to notify users of overlay events.
userData	Parameter that is passed to sink function callbacks.

Return Values

Value	Description
<code>VDP_OVERLAY_ERROR_SUCCESS</code>	Initialization succeeded.
<code>VDP_OVERLAY_ERROR_ALREADY_INITIALIZED</code>	Init has already been called.
<code>VDP_OVERLAY_ERROR_INVALID_PARAMETER</code>	NULL sink parameter, or invalid sink version.
<code>VDP_OVERLAY_ERROR_ALLOCATION_ERROR</code>	Internal system error.

v1.IsOverlayEnabled

Queries whether the overlay associated with the given `windowId` is currently enabled.

This function is a member of VDPOverlayGuest_Interface.

Method Signature

```
Bool (*v1.IsOverlayEnabled)(VDPOverlay_WindowId windowId);
```

Parameters

Parameter	Description
windowId	The window ID of the overlay. It must have been previously registered with VDPOverlayGuest_RegisterWindow().

Return Values

Value	Description
TRUE	The overlay is enabled.
FALSE	The overlay is disabled.

v1.IsWindowRegistered

Determines if a window is currently registered with the guest-side Overlay API.

This function is a member of VDPOverlayGuest_Interface.

Method Signature

```
Bool (*v1.IsWindowRegistered)(VDPOverlay_WindowId windowId);
```

Parameters

Parameter	Description
windowId	The window ID of the overlay.

Return Values

Value	Description
TRUE	Window is currently registered.
FALSE	The given window ID is not registered.

v1.RegisterWindow

Registers a window to be overlayed. The position, size, and so on of the window are sent to the client so that a client-side plug-in can draw an area of the desktop UI that covers the window, giving the illusion that the drawing is happening on the guest side.

This function is a member of VDPOverlayGuest_Interface.

Method Signature

```
VDPOverlay_Error (*v1.RegisterWindow)(VDPOverlay_WindowId windowId, VDPOverlay_UserArgs userArgs);
```

Parameters

Parameter	Description
windowId	The window ID of the overlay cast to a VDPOverlay_WindowId. A window can only be registered once.
userArgs	Data that is to be passed to the client-side plug-in when the OnWindowRegistered() event handler is called.

Return Values

Value	Description
VDP_OVERLAY_ERROR_SUCCESS	Window was successfully registered.
VDP_OVERLAY_ERROR_NOT_INITIALIZED	Overlay not initialized.
VDP_OVERLAY_ERROR_INVALID_PARAMETER	Invalid window ID.
VDP_OVERLAY_ERROR_ALLOCATION_ERROR	Internal system error.
VDP_OVERLAY_ERROR_WINDOW_ALREADY_REGISTERED	The given window ID has already been registered with the Overlay system.

v1.SendMsg for the Guest-Side Library

Sends a message to the client-side plug-in. The client's OnUserMsg event handler is called with the message.

This function is a member of VDPOverlayGuest_Interface.

Method Signature

```
VDPOverlay_Error (*v1.SendMsg)(VDPOverlay_WindowId windowId, void *msg, uint32 msgLen);
```

Parameters

Parameter	Description
windowId	The window ID of the overlay. It must have been previously registered with VDPOverlayGuest_RegisterWindow(). VDP_OVERLAY_WINDOW_ID_NONE can also be passed if the message is not directed to a particular window.
msg	Buffer that contains the message.
msgLen	Size of the msg buffer.

Return Values

Value	Description
VDP_OVERLAY_ERROR_SUCCESS	The message was sent to the client.
VDP_OVERLAY_ERROR_NOT_INITIALIZED	Overlay API has not been initialized.
VDP_OVERLAY_ERROR_INVALID_PARAMETER	Error occurred sending the supplied message.
VDP_OVERLAY_ERROR_WINDOW_NOT_REGISTERED	The given windowId has not been registered with the Overlay API.

v1.SetLayoutMode

Sets the current layout mode for the overlay. The layout mode is used to determine how an image is drawn (for example, scaled, cropped, and so on) when the size of the image doesn't match the size of the overlay.

This function is a member of `VDPOverlayGuest_Interface`.

Method Signature

```
VDPOverlay_Error (*v1.SetLayoutMode)(VDPOverlay_WindowId windowId, VDPOverlay_LayoutMode layoutMode);
```

Parameters

Parameter	Description
windowId	The window ID of the overlay. It must have been previously registered with <code>VDPOverlayGuest_RegisterWindow()</code> .

Parameter	Description
layoutMode	<p>Determines how the image is drawn. This can be one of the following VDPOverlay_LayoutMode values:</p> <p>VDP_OVERLAY_LAYOUT_CENTER</p> <p>The image will be drawn centered in the overlay and clipped to the size of the overlay. No scaling will take place.</p> <p>VDP_OVERLAY_LAYOUT_TILE</p> <p>The image will be tiled to fill the overlay. The image is not scaled but will be clipped on the right/bottom edges of the overlay.</p> <p>VDP_OVERLAY_LAYOUT_SCALE / VDP_OVERLAY_LAYOUT_SCALE_SHRINK_ONLY</p> <p>The image will be drawn to fill the entire overlay. No attempt at maintaining the aspect ratio of the image is made.</p> <p>VDP_OVERLAY_LAYOUT_CROP / VDP_OVERLAY_LAYOUT_CROP_SHRINK_ONLY</p> <p>The image will be scaled to fill the entire overlay while maintaining the aspect ratio. Parts of the image will be clipped if necessary.</p> <p>VDP_OVERLAY_LAYOUT_LETTERBOX / VDP_OVERLAY_LAYOUT_LETTERBOX_SHRINK_ONLY</p> <p>The image will be scaled such that either the width or height of the image will match the width/height of the overlay. The other dimension will be scaled to maintain the aspect ratio. No part of the image will be clipped but the image may not fill the entire overlay.</p> <p>VDP_OVERLAY_LAYOUT_MULTIPLE_CENTER / VDP_OVERLAY_LAYOUT_MULTIPLE_CORNER</p> <p>Multiple mode splits the overlay into 9 equal-sized boxes (like a tic-tac-toe board). The image is then scaled to fit into the center and corner boxes. This mode can be combined with any of the basic layout modes to determine how the image is scaled to fit in the box.</p> <p>If, after applying the layout mode, the image doesn't fill the entire box, VDP_OVERLAY_LAYOUT_MULTIPLE_CENTER places the image in the center of each box and VDP_OVERLAY_LAYOUT_MULTIPLE_CORNER justifies the image within each box to the nearest corner of the overlay. When combined with basic layout modes that always fill the overlay (e.g. VDP_OVERLAY_LAYOUT_SCALE and VDP_OVERLAY_LAYOUT_TILE), the multiple modes VDP_OVERLAY_LAYOUT_MULTIPLE_CENTER and VDP_OVERLAY_LAYOUT_MULTIPLE_CORNER behave the same.</p> <p>VDP_OVERLAY_LAYOUT_TO_MULTIPLE (multipleMode, basicMode)</p> <p>Where multipleMode must be either VDP_OVERLAY_LAYOUT_MULTIPLE_CENTER or VDP_OVERLAY_LAYOUT_MULTIPLE_CORNER, and basicMode must be one of the basic layout modes listed earlier in this table.</p> <p>VDP_OVERLAY_LAYOUT_TO_MULTIPLE returns a layout mode from the given multiple and basic layout modes.</p>

Return Values

Value	Description
VDP_OVERLAY_ERROR_SUCCESS	No error. The function was successful.
VDP_OVERLAY_ERROR_NOT_INITIALIZED	Overlay API was not initialized.

Value	Description
VDP_OVERLAY_ERROR_INVALID_PARAMETER	The windowId or layoutMode parameter was invalid, or there was an error with msg.
VDP_OVERLAY_ERROR_NOT_LOCAL_OVERLAY	The overlayId of a guest-side overlay was passed to a function that can only be called on a local overlay.

v1.UnregisterWindow

Unregisters a previously registered window. This method not only deactivates the client-side overlay, but also releases any resources allocated to maintain the overlay.

This function is a member of VDPOverlayGuest_Interface.

Method Signature

```
VDPOverlay_Error (*v1.UnregisterWindow)(VDPOverlay_WindowId windowId, VDPOverlay_UserArgs userArgs);
```

Parameters

Parameter	Description
windowId	The window ID of the overlay. The windowId must have been previously registered with VDPOverlayGuest_RegisterWindow().
userArgs	Data that is to be passed to the client-side plug-in when the VDPOverlayClient_WindowUnregistered event is sent.

Return Values

Value	Description
VDP_OVERLAY_ERROR_SUCCESS	Window was successfully unregistered.
VDP_OVERLAY_ERROR_NOT_INITIALIZED	Overlay has not been initialized.
VDP_OVERLAY_ERROR_WINDOW_NOT_REGISTERED	The given window ID was never registered with the Overlay system.

v2.GetColorkey

Retrieves the color key currently assigned to the windowId.

The color key is VDP_OVERLAY_HOST_COLORKEY_NONE until the windowId is assigned a color key by the overlay services.

This function is a member of VDPOverlayGuest_Interface.

Method Signature

```
VDPOverlay_Error (*v2.GetColorkey)(VDPOverlay_WindowId windowId, uint32* colorkey);
```

Parameters

Parameter	Description
windowId	The window ID of the overlay, which must have been previously registered with VDPOverlayGuest_RegisteredWindow().
colorkey	A pointer to a unit32 that contains the color key.

Return Values

Value	Description
VDP_OVERLAY_ERROR_SUCCESS	Current information retrieved.
VDP_OVERLAY_ERROR_NOT_INITIALIZED	Overlay API was not initialized.
VDP_OVERLAY_ERROR_INVALID_PARAMETER	The windowId or colorkey parameter was invalid.
VDP_OVERLAY_ERROR_WINDOW_NOT_REGISTERED	The given windowId has not been registered with the Overlay API.

v3.GetAreaRect

Gets the current constraining area of the overlay that was set by v3.SetAreaRect().

This function is a member of VDPOverlayGuest_Interface.

Method Signature

```
VDPOverlay_Error (*v3.GetAreaRect)(VDPOverlay_WindowId windowId, VDPOverlay_Rect* pRect);
```

Parameters

Parameter	Description
windowId	The window ID of the overlay. The windowId must have been previously registered.
pRect	A pointer to a VDPOverlay_Rect which returns the area of the window that is displaying the overlay. An area of all zeros indicates that the overlay doesn't have a constraining area set on it.

Return Values

Value	Description
VDP_OVERLAY_ERROR_SUCCESS	Constraining area of the overlay was successfully retrieved.
VDP_OVERLAY_ERROR_NOT_INITIALIZED	Overlay API was not initialized.
VDP_OVERLAY_ERROR_INVALID_PARAMETER	One of the parameter entries was invalid
VDP_OVERLAY_ERROR_WINDOW_NOT_REGISTERED	The given windowId has not been registered with the Overlay API.

v3.GetLayer

Gets the layer of an overlay as set by v3.SetLayer().

This function is a member of VDPOverlayGuest_Interface.

Method Signature

```
VDPOverlay_Error (*v3.GetLayer)(VDPOverlay_WindowId windowId, uint32* pLayer);
```

Parameters

Parameter	Description
windowId	The window ID of the overlay. The windowId must have been previously registered.
pLayer	Returns the layer of the overlay.

Return Values

Value	Description
VDP_OVERLAY_ERROR_SUCCESS	Layer of the overlay was successfully retrieved.
VDP_OVERLAY_ERROR_NOT_INITIALIZED	Overlay API was not initialized.
VDP_OVERLAY_ERROR_INVALID_PARAMETER	pLayer is NULL.
VDP_OVERLAY_ERROR_WINDOW_NOT_REGISTERED	The given windowId has not been registered with the Overlay API.

v3.RegisterWindow

Registers a window to be overlayed. The position, size, and so on, of the window are sent to the client so that a client-side plug-in can draw an area of the desktop UI that covers the window, giving the illusion that the drawing is happening on the guest side.

This function performs the same operations as `v1.RegisterWindow()` but supports additional options.

- A window can be registered multiple times, allowing you to use different areas of the window to display the overlay image. Use `SetAreaRect()` to define the area within the window for displaying the overlay image.
- The first parameter is a `VDPOverlay_HWND` instead of a `VDPOverlay_WindowId`. The size of a `VDPOverlay_WindowId` is 32-bits but on 64-bit Windows an `HWND` is 64 bits. The parameter `VDPOverlay_HWND`, which is defined as an `HWND`, removes the need to cast the `HWND` to a `VDPOverlay_WindowId`. This solution guarantees that bits are not lost when casting.

This function is a member of `VDPOverlayGuest_Interface`.

Method Signature

```
VDPOverlay_Error (*v3.RegisterWindow)(VDPOverlay_HWND hWnd, VDPOverlay_UserArgs userArgs, VDPOverlay_WindowId* pWindowId);
```

Parameters

Parameter	Description
hWnd	The operating system window identifier. A window can be registered multiple times.
userArgs	Data that is to be passed to the client-side plug-in when the <code>OnWindowRegistered()</code> event handler is called.

Parameter	Description
pWindowId	Returns a VDPOverlay_WindowId for use in other VDPOverlayGuest_Interface API calls.

Return Values

Value	Description
VDP_OVERLAY_ERROR_SUCCESS	Window was successfully registered.
VDP_OVERLAY_ERROR_NOT_INITIALIZED	Overlay not initialized.
VDP_OVERLAY_ERROR_INVALID_PARAMETER	Invalid window ID.
VDP_OVERLAY_ERROR_ALLOCATION_ERROR	Internal system error.
VDP_OVERLAY_ERROR_WINDOW_ALREADY_REGISTERED	The given window ID has already been registered with the Overlay system.

v3.SetAreaRect

Sets the area of the window for displaying the overlay.

This function is a member of VDPOverlayGuest_Interface.

Method Signature

```
VDPOverlay_Error (*v3.SetAreaRect)(VDPOverlay_WindowId windowId, VDPOverlay_Rect* pRect);
```

Parameters

Parameter	Description
windowId	The operating system window identifier. The windowId must have been previously registered.
pRect	A pointer to a VDPOverlay_Rect which defines the area of the window for displaying the overlay. Passing NULL removes the constraining area and displays the image in the entire area of the window.

Return Values

Value	Description
VDP_OVERLAY_ERROR_SUCCESS	No error. The function was successful.
VDP_OVERLAY_ERROR_NOT_INITIALIZED	Overlay API was not initialized.
VDP_OVERLAY_ERROR_INVALID_PARAMETER	One of the parameter entries was invalid.
VDP_OVERLAY_ERROR_WINDOW_NOT_REGISTERED	The given windowId has not been registered with the Overlay API.

v3.SetLayer

Sets the layer on an overlay. If two overlays registered to the same window have overlapping area rectangles, you can specify which overlay is on top by setting its layer. Layers have no effect on overlays registered to different operating system windows.

This function is a member of VDPOverlayGuest_Interface.

Method Signature

```
VDPOverlay_Error (*v3.SetLayer)(VDPOverlay_WindowId windowId, uint32 layer);
```

Parameters

Parameter	Description
windowId	The operating system window identifier. The windowId must have been previously registered.
layer	The layer of the overlay. Overlays with a higher layer value will be on top of overlays with a lower layer value. If two overlays have the same layer value the overlay created last will be on top.

Return Values

Value	Description
VDP_OVERLAY_ERROR_SUCCESS	No error. The function was successful.
VDP_OVERLAY_ERROR_NOT_INITIALIZED	Overlay API was not initialized.
VDP_OVERLAY_ERROR_INVALID_PARAMETER	The windowId parameter was invalid.

v4.GetAreaRect

Gets the current constraining area of the overlay that was set by v4.SetAreaRect().

This function is a member of VDPOverlayGuest_Interface.

Method Signature

```
VDPOverlay_Error (*v4.GetAreaRect)(VDPOverlay_WindowId windowId, Bool* pEnabled, Bool* pClipToWindow, VDPOverlay_Rect* pRect);
```

Parameters

Parameter	Description
windowId	The window ID of the overlay. The windowId must have been previously registered.
pEnabled	A pointer to a Bool which returns the enabled flag passed to v4.SetAreaRect(). Pass NULL to not return the value.
pClipToWindow	A pointer to a Bool which returns the clipToWindow flag passed to v4.SetAreaRect(). Pass NULL to not return the value.
pRect	A pointer to a VDPOverlay_Rect which returns the rectangle passed to v4.SetAreaRect(). Pass NULL to not return the value.

Return Values

Value	Description
VDP_OVERLAY_ERROR_SUCCESS	Constraining area of the overlay was successfully retrieved.
VDP_OVERLAY_ERROR_NOT_INITIALIZED	Overlay API was not initialized.
VDP_OVERLAY_ERROR_INVALID_PARAMETER	One of the parameter entries was invalid
VDP_OVERLAY_ERROR_WINDOW_NOT_REGISTERED	The given windowId has not been registered with the Overlay API.

v4.GetBackgroundColor

Gets the current color used to paint the background of the overlay.

This function is a member of VDPOverlayGuest_Interface.

Method Signature

```
VDPOverlay_Error (*v4.GetBackgroundColor)(VDPOverlay_WindowId windowId, uint32* pBackgroundColor);
```

Parameters

Parameter	Description
windowId	The window ID of the overlay. The windowId must have been previously registered.
pBackgroundColor	A pointer to a uint32 which returns the color passed to v4.SetBackgroundColor().

Return Values

Value	Description
VDP_OVERLAY_ERROR_SUCCESS	The current background color was successfully retrieved.
VDP_OVERLAY_ERROR_NOT_INITIALIZED	Overlay API was not initialized.
VDP_OVERLAY_ERROR_INVALID_PARAMETER	The windowId or pBackgroundColor parameter was invalid.
VDP_OVERLAY_ERROR_WINDOW_NOT_REGISTERED	The given windowId has not been registered with the Overlay API.

v4.GetHWnd

With v1.RegisterWindow() the HWND and windowId are the same but v3.RegisterWindow() returns a unique windowId. This function provides a way to retrieve the original HWND used to register the window.

This function is a member of VDPOverlayGuest_Interface.

Method Signature

```
VDPOverlay_Error (*v4.GetHWnd)(VDPOverlay_WindowId windowId, VDPOverlay_HWND* pHWnd);
```

Parameters

Parameter	Description
windowId	The window ID of the overlay. The windowId must have been previously registered.
pHWnd	A pointer to a VDPOverlay_HWND which returns the window handle used to register the window.

Return Values

Value	Description
VDP_OVERLAY_ERROR_SUCCESS	The original HWND used to register the window was successfully retrieved.
VDP_OVERLAY_ERROR_NOT_INITIALIZED	Overlay API was not initialized.
VDP_OVERLAY_ERROR_INVALID_PARAMETER	pHWnd is NULL.
VDP_OVERLAY_ERROR_WINDOW_NOT_REGISTERED	The given windowId has not been registered with the Overlay API.

v4.GetInfoString

Gets the current information string for the overlay as set by v4.SetInfoString().

This function is a member of VDPOverlayGuest_Interface.

Method Signature

```
VDPOverlay_Error (*v4.GetInfoString)(VDPOverlay_WindowId windowId, char* infoStr, int32 infoStrSize);
```

Parameters

Parameter	Description
windowId	The window ID of the overlay. The windowId must have been previously registered.
infoStr	A pointer to a buffer that is filled with the current information string.
infoStrSize	The size of the infoStr buffer.

Return Values

Value	Description
VDP_OVERLAY_ERROR_SUCCESS	Current information retrieved.
VDP_OVERLAY_ERROR_NOT_INITIALIZED	Overlay API was not initialized.
VDP_OVERLAY_ERROR_INVALID_PARAMETER	One of the parameter entries was invalid.

VDP_OVERLAY_ERROR_WINDOW_NOT_REGISTERED

The given windowId has not been registered with the Overlay API.

v4.SetAreaRect

Sets the area of the window for displaying the overlay. Same as v3.SetAreaRect(), but with additional parameters for turning on and off the rectangle area and for clipping the rectangle area to the window.

This function is a member of VDPOverlayGuest_Interface.

Method Signature

```
VDPOverlay_Error (*v4.SetAreaRect)(VDPOverlay_WindowId windowId, Bool enabled, Bool clipToWindow, VDPOverlay_Rect* pRect);
```

Parameters

Parameter	Description
windowId	The window ID of the overlay. The windowId must have been previously registered.
enabled	Determines when the area rectangle is enabled. If TRUE, the overlay is constrained to the given rectangle. If FALSE, the overlay will be displayed in the entire area of the window.
clipToWindow	Determines if the given area rectangle is clipped to the window. Note The image displayed in the overlay is always clipped to the window. When this flag is TRUE, the given area rectangle is clipped to the window before the layout mode is applied, such that the image is scaled down to fit inside the clipped area rectangle. When this flag is FALSE, the layout mode is applied first and then the image is clipped to the window. In this case, the image doesn't shrink when the bounds of the area rectangle extend past the bounds of the window, but less of the image is shown.
pRect	A pointer to a VDPOverlay_Rect which defines the area of the window for displaying the overlay. Passing NULL removes the constraining area and displays the image in the entire area of the window.

Return Values

Value	Description
VDP_OVERLAY_ERROR_SUCCESS	No error. The function was successful.
VDP_OVERLAY_ERROR_NOT_INITIALIZED	Overlay API was not initialized.
VDP_OVERLAY_ERROR_INVALID_PARAMETER	One of the parameter entries was invalid.
VDP_OVERLAY_ERROR_WINDOW_NOT_REGISTERED	The given windowId has not been registered with the Overlay API.

v4.SetBackgroundColor

Sets the background color to use when painting the area of the window that the overlay covers. This color is visible in the area of the overlay that the image does not cover, for example, the borders of the image when the layout mode is VDP_OVERLAY_LAYOUT_LETTERBOX. The background color is also visible if the image has an alpha channel.

This function is a member of VDPOverlayGuest_Interface.

Method Signature

```
VDPOverlay_Error (*v4.SetBackgroundColor)(VDPOverlay_WindowId windowId, uint32 bgColor);
```

Parameters

Parameter	Description
windowId	The operating system window identifier. The windowId must have been previously registered.
bgColor	The color to use when painting the overlay, in XXRRGGBB format. The alpha value in the color is ignored and set to 0xFF. Pass 0 to turn off painting the background, which allows the application to show through.

Return Values

Value	Description
VDP_OVERLAY_ERROR_SUCCESS	No error. The function was successful.
VDP_OVERLAY_ERROR_NOT_INITIALIZED	Overlay API was not initialized.
VDP_OVERLAY_ERROR_INVALID_PARAMETER	The windowId or bgColor parameter was invalid.
VDP_OVERLAY_ERROR_WINDOW_NOT_REGISTERED	There is an error with a low-level library. This error code should be treated as similar to INVALID_PARAMETER.

v4.SetInfoString

Sets a string that is rendered on top of the overlay. The string can contain arbitrary information which can assist with closed captioning or debugging information.

This function is a member of VDPOverlayGuest_Interface.

Method Signature

```
VDPOverlay_Error (*v4.SetInfoString)(VDPOverlay_WindowId windowId, const char* infoStr);
```

Parameters

Parameter	Description
-----------	-------------

windowId The window ID of the overlay. The windowId must have been previously registered.

infoStr The information string. The maximum string length is VDP_OVERLAY_INFO_STR_MAX_LEN bytes, including the NULL terminator. The information string can contain the following macros:

Macro	Definition
\$(FPS)	FPS or image format (default)
\$(IMAGE_SIZE)	Source image size (WxH)
\$(IMAGE_ASPECT)	Source image aspect ratio (X.XX)
\$(IMAGE_FORMAT)	Source image format
\$(OVERLAY_ID)	Overlay ID
\$(OVERLAY_POS)	Overlay position (X,Y)
\$(OVERLAY_SIZE)	Overlay size (WxH)
\$(OVERLAY_ASPECT)	Overlay image aspect ratio (X.XX)
\$(OVERLAY_LAYER)	Overlay layer
\$(OVERLAY_LAYOUT)	Overlay layout mode
\$(OVERLAY_SURFACE)	Overlay surface type
\$(OVERLAY_FPS)	Overlay frame rate
\$(OVERLAY_FRAME_NUM)	Overlay frame number
\$(HORIZON_FPS) <i>replaced \$(VIEW_FPS) in release 2503</i>	Horizon client frame rate
\$(HORIZON_FRAME_NUM) <i>replaced \$(VIEW_FRAME_NUM) in release 2503</i>	Horizon client frame number
\$(HORIZON_WINDOW_SIZE) <i>replaced \$(VIEW_WINDOW_SIZE) in release 2503</i>	Horizon client window size
\$(HORIZON_PROTOCOL) <i>replaced \$(VIEW_PROTOCOL) in release 2503</i>	Horizon client protocol (Blast/PCoIP)
\$(REMOTE_DESKTOP_SIZE)	Horizon remote desktop size (WxH)
\$(TIME)	Current time
\$(DATE)	Current date

The following escape characters are recognized, assuming that the string is read from a file or the registry. When hardcoding the information string in C/C++ code, you must also escape the backslash character itself.

Escape Character	Definition
\n	New line; the LF character ('\n' in C/C++) is also a new line
\\$	Dollar sign
\\	Backslash

Return Values

Value	Description
VDP_OVERLAY_ERROR_SUCCESS	No error. The function was successful.
VDP_OVERLAY_ERROR_NOT_INITIALIZED	Overlay API was not initialized.
VDP_OVERLAY_ERROR_INVALID_PARAMETER	One of the parameter entries was invalid.
VDP_OVERLAY_ERROR_WINDOW_NOT_REGISTERED	The given windowId has not been registered with the Overlay API.

VDPOverlayClient_Interface Functions

With VDPOverlayClient_Interface functions, you can work with an overlay, send a message to the guest-side plug-in, and release all allocated resources.

v1.Exit

Performs cleanup operations and releases all allocated resources.

This function is a member of VDPOverlayClient_Interface.

Method Signature

```
VDPOverlay_Error (*v1.Exit)(VDPOverlayClient_ContextId contextId);
```

Parameters

Parameter	Description
contextId	The ID returned from VDPOverlayClient.v1Init().

Return Values

Value	Description
VDP_OVERLAY_ERROR_SUCCESS	Overlay API was properly shut down.
VDP_OVERLAY_ERROR_NOT_INITIALIZED	Overlay API was not initialized.
VDP_OVERLAY_ERROR_INVALID_PARAMETER	contextId was invalid.

v1.GetInfo

Retrieves the current information about the overlay.

This function is a member of VDPOverlayClient_Interface.

Method Signature

```
VDPOverlay_Error (*v1.GetInfo)(VDPOverlayClient_ContextId contextId, VDPOverlay_WindowId windowId, VDPOverlayClient_OverlayInfo* pOverlayInfo);
```

Parameters

Parameter	Description
contextId	The ID returned from VDPOverlayClient.v1Init().
windowId	Window ID that was cached from a previous OnWindowRegistered() event.
pOverlayInfo	A pointer to a VDPOverlayClient_OverlayInfo structure which will be filled in with information about the overlay.

Return Values

Value	Description
VDP_OVERLAY_ERROR_SUCCESS	Current information retrieved.
VDP_OVERLAY_ERROR_NOT_INITIALIZED	Overlay API was not initialized.
VDP_OVERLAY_ERROR_INVALID_PARAMETER	The contextId or windowId parameter was invalid, or pOverlayInfo was NULL

v1.Init for the Client-Side Library

This function initializes the client-side overlay library. This method must be the first method called in the client-side Overlay API.

This function is a member of VDPOverlayClient_Interface.

Method Signature

```
VDPOverlay_Error (*v1.Init)(const VDPOverlayClient_Sink* sink, void* userData, VDPOverlayClient_ContextId* pContextId);
```

Parameters

Parameter	Description
sink	Function pointers called when events are generated by the Overlay API.
userData	Parameter that is passed to sink callbacks. Can be NULL.
pContextId	Returns an ID that identifies the instance of the API for this plug-in instance.

Return Values

Value	Description
VDP_OVERLAY_ERROR_SUCCESS	Overlay client API was initialized.
VDP_OVERLAY_ERROR_NOT_INITIALIZED	Internal Horizon Session Enhancement initialization error.
VDP_OVERLAY_ERROR_INVALID_PARAMETER	The sink or pContextId parameter is NULL or sink reported an invalid version.
VDP_OVERLAY_ERROR_ALLOCATION_ERROR	Internal system error.

v1.SendMsg

Sends a message to the guest-side plug-in. The guest's OnUserMsg() event handler is called with the message.

This function is a member of VDPOverlayClient_Interface.

Method Signature

```
VDPOverlay_Error (*v1.SendMsg)(VDPOverlayClient_ContextId contextId, VDPOverlay_WindowId windowId, void* msg, uint32 msgLen);
```

Parameters

Parameter	Description
contextId	The ID returned from VDPOverlayClient.v1.Init().
windowId	Window ID that was cached from a previous OnWindowRegistered() event. You may also pass VDP_OVERLAY_WINDOW_ID_NONE if the message is not directed to a particular window.
msg	A pointer to a buffer that contains the message.
msgLen	Size of the msg buffer in bytes. The maximum message length is VDP_OVERLAY_USER_MSG_MAX_LEN bytes.

Return Values

Value	Description
VDP_OVERLAY_ERROR_SUCCESS	The message was sent.
VDP_OVERLAY_ERROR_NOT_INITIALIZED	Overlay API was not initialized.
VDP_OVERLAY_ERROR_INVALID_PARAMETER	The contextId or windowId parameter was invalid, or there was an error with msg.

v1.Update

Updates the overlay with a new image. The updated image is displayed when the next frame is drawn.

This function is a member of VDPOverlayClient_Interface.

Method Signature

```
VDPOverlay_Error (*v1.Update)(VDPOverlayClient_ContextId contextId, VDPOverlay_WindowId windowId, void* pImage, int32 width, int32 height, int32 pitch, Bool copyImage);
```

Parameters

Parameter	Description
contextId	The ID returned from VDPOverlayClient.v1.Init().
windowId	Window ID that was received from a previous OnWindowRegistered() event.
pImage	A pointer to the RGBX pixels to copy to the overlay.

Parameter	Description
width	Width, in pixels, of the image pointed to by plmage. If the width of the image does not match the width of the overlay, the given image is drawn according the layout mode of the overlay.
height	Height, in pixels, of the image pointed to by plmage. If the height of the image does not match the height of overlay, the given image is drawn according the layout mode of the overlay.
pitch	Number of bytes that a single row of the image occupies. In the normal case, for BGRX images, this value is width multiplied by 4.
copyImage	If TRUE, a copy of the image data is made. If FALSE, no copy is made and the image data must remain valid until another call to VDPOverlayClient_Update() is made.

Return Values

Value	Description
VDP_OVERLAY_ERROR_SUCCESS	Image was updated.
VDP_OVERLAY_ERROR_NOT_INITIALIZED	Overlay API was not initialized.
VDP_OVERLAY_ERROR_INVALID_PARAMETER	The contextId, windowId, or plmage parameter was invalid.

v2.CreateOverlay

Creates a local overlay.

The overlay is not tied to a window on the guest (such an overlay is referred to as a "guest created overlay"). Locally created overlays give the client complete control over the overlay but also require the client to do more of the work.

This function is a member of VDPOverlayClient_Interface.

Method Signature

```
VDPOverlay_Error (*v2.CreateOverlay)(VDPOverlayClient_ContextId contextId, VDPOverlay_OverlayId* pOverlayId);
```

Parameters

Parameter	Description
contextId	The ID returned from VDPOverlayClient.v1.Init().
pOverlayId	Returns a VDPOverlay_OverlayId that can be used to set properties on the overlay. This ID may also be passed to functions that take a VDPOverlay_WindowId, for example, Update(), GetInfo(), and so on.

Return Values

Value	Description
VDP_OVERLAY_ERROR_SUCCESS	No error. The function was successful.
VDP_OVERLAY_ERROR_NOT_INITIALIZED	Overlay API was not initialized.

Value	Description
VDP_OVERLAY_ERROR_INVALID_PARAMETER	The contextId parameter was invalid, or the pOverlayId is NULL.
VDP_OVERLAY_ERROR_ALLOCATION_ERROR	The system fails to allocate the required memory or system resource to handle the call.
VDP_OVERLAY_ERROR_NO_MORE_OVERLAYS	This error is returned when too many overlays have been created.

v2.DestroyOverlay

Destroys a local overlay.

All the resources associated with the overlay are released. This function cannot be called on guest-created overlays.

This function is a member of VDPOverlayClient_Interface.

Method Signature

```
VDPOverlay_Error (*v2.DestroyOverlay)(VDPOverlayClient_ContextId contextId, VDPOverlay_OverlayId overlayId);
```

Parameters

Parameter	Description
contextId	The ID returned from VDPOverlayClient.v1.Init().
overlayId	An overlay ID that was returned from a previous call to CreateOverlay().

Return Values

Value	Description
VDP_OVERLAY_ERROR_SUCCESS	No error. The function was successful.
VDP_OVERLAY_ERROR_NOT_INITIALIZED	Overlay API was not initialized.
VDP_OVERLAY_ERROR_INVALID_PARAMETER	The contextId or overlayId parameter was invalid.
VDP_OVERLAY_ERROR_NOT_LOCAL_OVERLAY	The overlayId of a guest-side overlay was passed to a function that can only be called on a local overlay.

v2.DisableOverlay

Deactivates an overlay.

Deactivating an overlay is a lightweight way to hide an overlay. Unlike DestroyOverlay(), resources used to maintain the overlay are not released.

This function is a member of VDPOverlayClient_Interface.

Method Signature

```
VDPOverlay_Error (*v2.DisableOverlay)(VDPOverlayClient_ContextId contextId, VDPOverlay_WindowId windowId);
```

Parameters

Parameter	Description
contextId	The ID returned from VDPOverlayClient.v1.Init().
windowId	A window ID that was cached from a previous OnWindowRegistered() event.

Return Values

Value	Description
VDP_OVERLAY_ERROR_SUCCESS	No error. The function was successful.
VDP_OVERLAY_ERROR_NOT_INITIALIZED	Overlay API was not initialized.
VDP_OVERLAY_ERROR_INVALID_PARAMETER	The contextId or windowId parameter was invalid.
VDP_OVERLAY_ERROR_HOST_OVERLAY_ERROR	There is an error with a low-level library. This error code should be treated as similar to INVALID_PARAMETER.

v2.EnableOverlay

Activates an overlay that was previously deactivated.

An overlay can be deactivated if either the guest or client calls DisableOverlay() for the given window ID.

This function is a member of VDPOverlayClient_Interface.

Method Signature

```
VDPOverlay_Error (*v2.EnableOverlay)(VDPOverlayClient_ContextId contextId, VDPOverlay_WindowId windowId);
```

Parameters

Parameter	Description
contextId	The ID returned from VDPOverlayClient.v1.Init().
windowId	A window ID that was cached from a previous OnWindowRegistered() event.

Return Values

Value	Description
VDP_OVERLAY_ERROR_SUCCESS	No error. The function was successful.
VDP_OVERLAY_ERROR_NOT_INITIALIZED	Overlay API was not initialized.
VDP_OVERLAY_ERROR_INVALID_PARAMETER	The contextId or windowId parameter was invalid.
VDP_OVERLAY_ERROR_HOST_OVERLAY_ERROR	There is an error with a low-level library. This error code should be treated as similar to INVALID_PARAMETER.

v2.GetInfo

Retrieves current information about the overlay.

This function is a member of VDPOverlayClient_Interface.

Method Signature

```
VDPOverlay_Error (*v2.GetInfo)(VDPOverlayClient_ContextId contextId, VDPOverlay_WindowId windowId, VDPOverlayClient_OverlayInfo* pOverlayInfo);
```

Parameters

Parameter	Description
contextId	The ID returned from VDPOverlayClient.v1.Init().
windowId	Window ID that was cached from a previous OnWindowRegistered() event.
pOverlayInfo	A pointer to a VDPOverlayClient_OverlayInfo structure which will be filled in with information about the overlay.

Return Values

Value	Description
VDP_OVERLAY_ERROR_SUCCESS	Current information retrieved.
VDP_OVERLAY_ERROR_NOT_INITIALIZED	Overlay API was not initialized.
VDP_OVERLAY_ERROR_INVALID_PARAMETER	The contextId or windowID parameter was invalid, or pOverlayInfo was NULL.

v2.InitLocal

Initializes the client-side overlay library for use with local overlays only.

The overhead of creating an RPC connection to track guest side windows is not performed. You do not need to call this function if you have already called v1.Init().

This function is a member of VDPOverlayClient_Interface.

Method Signature

```
VDPOverlay_Error (*v1.InitLocal)(const VDPOverlayClient_Sink* sink, void* userData, VDPOverlayClient_ContextId* pContextId);
```

Parameters

Parameter	Description
sink	Contains the function pointers that are called when events are generated by the Overlay library.
userData	The parameter that is passed to event handler whenever an event is delivered.
pContextId	Returns an ID that is used to identify the instance of the API. This ID must be passed to all other API functions. This ID is also passed when calling the sink handlers.

Return Values

Value	Description
VDP_OVERLAY_ERROR_SUCCESS	No error. The function was successful.
VDP_OVERLAY_ERROR_NOT_INITIALIZED	Overlay API was not initialized.
VDP_OVERLAY_ERROR_INVALID_PARAMETER	The sink or userData parameter was invalid, or pContextId is NULL.
VDP_OVERLAY_ERROR_ALLOCATION_ERROR	The system fails to allocate the required memory or system resource to handle the call.

v2.SetClipRegion

Sets the clipping region on the overlay.

This function cannot be called on guest-created overlays.

This function is a member of VDPOverlayClient_Interface.

Method Signature

```
VDPOverlay_Error (*v2.SetClipRegion)(VDPOverlayClient_ContextId contextId, VDPOverlay_OverlayId overlayId, VMRect* pClipRects, int32 nClipRects);
```

Parameters

Parameter	Description
contextId	The ID returned from VDPOverlayClient.v1.Init().
overlayId	An overlay ID that was returned from a previous call to CreateOverlay().
pClipRects	An array of VMRect's that describe the visible area of the overlay. The clipping information is relative to the screen. For example, 0,0 is the top-left corner of the screen. This means that the clipping information describes a specific area of the screen that does not change when the overlay is moved. A copy of the VMRect array is made so that the caller does not have to maintain the memory.
nClipRects	The number of VMRects in the pClipRects array. Passing the value of 0 for nClipRects removes the clip region.

Return Values

Value	Description
VDP_OVERLAY_ERROR_SUCCESS	No error. The function was successful.
VDP_OVERLAY_ERROR_NOT_INITIALIZED	Overlay API was not initialized.
VDP_OVERLAY_ERROR_INVALID_PARAMETER	The contextId or overlayId parameter was invalid, or pClipRects is NULL.

VDP_OVERLAY_ERROR_NOT_LOCAL_OVERLAY	The overlayId of a guest-side overlay was passed to a function that can only be called on a local overlay.
VDP_OVERLAY_ERROR_HOST_OVERLAY_ERROR	There is an error with a low-level library. This error code should be treated as similar to INVALID_PARAMETER.

v2.SetColorkey

Sets the color key on a local overlay.

Note The use of color keys is discouraged because they do not work well with the Blast protocol. To limit the area of the guest UI for rendering the overlay, use v2.SetClipRegion instead.

This function cannot be called on guest-created overlays.

This function is a member of VDPOverlayClient_Interface.

Method Signature

```
VDPOverlay_Error (*v2.SetColorkey)(VDPOverlayClient_ContextId contextId, VDPOverlay_OverlayId overlayId, uint32 colorkey);
```

Parameters

Parameter	Description
contextId	The ID returned from VDPOverlayClient.v1.Init().
overlayId	An overlay ID that was returned from a previous call to CreateOverlay().
colorkey	An RGB value that will limit the area of the guest UI where the overlay is drawn. When a color key is set on an overlay only the pixels on the guest's UI that match the color key value will be updated. It is the caller's responsibility to draw the color key to an area on the guest's desktop that corresponds to the position of the overlay as set by SetPosition(). Passing VDP_OVERLAY_COLORKEY_NONE will remove the color key from the overlay.

Return Values

Value	Description
VDP_OVERLAY_ERROR_SUCCESS	No error. The function was successful.
VDP_OVERLAY_ERROR_NOT_INITIALIZED	Overlay API was not initialized.
VDP_OVERLAY_ERROR_INVALID_PARAMETER	The contextId, overlayId or colorkey parameter was invalid.
VDP_OVERLAY_ERROR_NOT_LOCAL_OVERLAY	The overlayId of a guest-side overlay was passed to a function that can only be called on a local overlay.
VDP_OVERLAY_ERROR_HOST_OVERLAY_ERROR	There is an error with a low-level library. This error code should be treated as similar to INVALID_PARAMETER.

v2.SetLayer

Sets the layer on a local overlay.

This function cannot be called on guest-created overlays.

This function is a member of `VDPOverlayClient_Interface`.

Method Signature

```
VDPOverlay_Error (*v2.SetLayer)(VDPOverlayClient_ContextId contextId, VDPOverlay_OverlayId overlayId, uint32 layer);
```

Parameters

Parameter	Description
contextId	The ID returned from <code>VDPOverlayClient.v1.Init()</code> .
overlayId	An overlay ID that was returned from a previous call to <code>CreateOverlay()</code> .
layer	The layer of the overlay. Overlays with a higher layer value will be on top of overlays with a lower layer value. If two overlays have the same layer value the overlay created last will be on top.

Return Values

Value	Description
<code>VDP_OVERLAY_ERROR_SUCCESS</code>	No error. The function was successful.
<code>VDP_OVERLAY_ERROR_NOT_INITIALIZED</code>	Overlay API was not initialized.
<code>VDP_OVERLAY_ERROR_INVALID_PARAMETER</code>	The contextId or overlayId parameter was invalid.
<code>VDP_OVERLAY_ERROR_NOT_LOCAL_OVERLAY</code>	The overlayId of a guest-side overlay was passed to a function that can only be called on a local overlay.
<code>VDP_OVERLAY_ERROR_HOST_OVERLAY_ERROR</code>	There is an error with a low-level library. This error code should be treated as similar to <code>INVALID_PARAMETER</code> .

v2.SetLayoutMode

Sets the current layout mode for the overlay.

The layout mode is used to determine how an image is drawn, for example, scaled, cropped, and so on, when the size of the image doesn't match the size of the overlay.

This function is a member of `VDPOverlayClient_Interface`.

Method Signature

```
VDPOverlay_Error (*v2.SetLayoutMode)(VDPOverlayClient_ContextId contextId, VDPOverlay_WindowId windowId, VDPOverlay_LayoutMode layoutMode);
```


Parameters

Parameter	Description
contextId	The ID returned from VDPOverlayClient.v1.Init().
windowId	A window ID that was cached from a previous OnWindowRegistered() event.
layoutMode	<p>Determines how the image is drawn. This can be one of the following VDPOverlay_LayoutMode values:</p> <p>VDP_OVERLAY_LAYOUT_CENTER</p> <p>The image will be drawn centered in the overlay and clipped to the size of the overlay. No scaling will take place.</p> <p>VDP_OVERLAY_LAYOUT_TILE</p> <p>The image will be tiled to fill the overlay. The image is not scaled but will be clipped on the right/bottom edges of the overlay.</p> <p>VDP_OVERLAY_LAYOUT_SCALE / VDP_OVERLAY_LAYOUT_SCALE_SHRINK_ONLY</p> <p>The image will be drawn to fill the entire overlay. No attempt at maintaining the aspect ratio of the image is made.</p> <p>VDP_OVERLAY_LAYOUT_CROP / VDP_OVERLAY_LAYOUT_CROP_SHRINK_ONLY</p> <p>The image will be scaled to fill the entire overlay while maintaining the aspect ratio. Parts of the image will be clipped if necessary.</p> <p>VDP_OVERLAY_LAYOUT_LETTERBOX / VDP_OVERLAY_LAYOUT_LETTERBOX_SHRINK_ONLY</p> <p>The image will be scaled such that either the width or height of image will match the width/height of the overlay. The other dimension will be scaled to maintain the aspect ratio. No part of the image will be clipped but the image may not fill the entire overlay.</p> <p>VDP_OVERLAY_LAYOUT_MULTIPLE_CENTER / VDP_OVERLAY_LAYOUT_MULTIPLE_CORNER</p> <p>Multiple mode splits the overlay into 9 equal-sized boxes (like a tic-tac-toe board). The image is then scaled to fit into the center and corner boxes. This mode can be combined with any of the basic layout modes to determine how the image is scaled to fit in the box.</p> <p>If, after applying the layout mode, the image doesn't fill the entire box,</p> <p>VDP_OVERLAY_LAYOUT_MULTIPLE_CENTER places the image in the center of each box and</p> <p>VDP_OVERLAY_LAYOUT_MULTIPLE_CORNER justifies the image within each box to the nearest corner of the overlay. When combined with basic layout modes that always fill the overlay (e.g. VDP_OVERLAY_LAYOUT_SCALE and VDP_OVERLAY_LAYOUT_TILE), the multiple modes VDP_OVERLAY_LAYOUT_MULTIPLE_CENTER and VDP_OVERLAY_LAYOUT_MULTIPLE_CORNER behave the same.</p> <p>VDP_OVERLAY_LAYOUT_TO_MULTIPLE (multipleMode, basicMode)</p> <p>Where multipleMode must be either VDP_OVERLAY_LAYOUT_MULTIPLE_CENTER or VDP_OVERLAY_LAYOUT_MULTIPLE_CORNER, and basicMode must be one of the basic layout modes listed earlier in this table.</p> <p>VDP_OVERLAY_LAYOUT_TO_MULTIPLE returns a layout mode from the given multiple and basic layout modes.</p>

Return Values

Value	Description
VDP_OVERLAY_ERROR_SUCCESS	No error. The function was successful.
VDP_OVERLAY_ERROR_NOT_INITIALIZED	Overlay API was not initialized.
VDP_OVERLAY_ERROR_INVALID_PARAMETER	The contextId or windowID parameter was invalid.
VDP_OVERLAY_ERROR_NOT_LOCAL_OVERLAY	The overlayId of a guest-side overlay was passed to a function that can only be called on a local overlay.
VDP_OVERLAY_ERROR_HOST_OVERLAY_ERROR	There is an error with a low-level library. This error code should be treated as similar to INVALID_PARAMETER.

v2.SetPosition

Sets the position of a local overlay.

This function cannot be called on guest-created overlays.

This function is a member of VDPOverlayClient_Interface.

Method Signature

```
VDPOverlay_Error (*v2.SetPosition)(VDPOverlayClient_ContextId contextId, VDPOverlay_OverlayId overlayId, int32 x, int32 y);
```

Parameters

Parameter	Description
contextId	The ID returned from VDPOverlayClient.v1.Init().
overlayId	An overlay ID that was returned from a previous call to CreateOverlay().
x,y	The position of the overlay. The position is specified as the upper-left corner of the overlay in guest UI coordinates.

Return Values

Value	Description
VDP_OVERLAY_ERROR_SUCCESS	No error. The function was successful.
VDP_OVERLAY_ERROR_NOT_INITIALIZED	Overlay API was not initialized.
VDP_OVERLAY_ERROR_INVALID_PARAMETER	The contextId or overlayID parameter was invalid.
VDP_OVERLAY_ERROR_NOT_LOCAL_OVERLAY	The overlayId of a guest-side overlay was passed to a function that can only be called on a local overlay.

Value	Description
VDP_OVERLAY_ERROR_HOST_OVERLAY_ERROR	There is an error with a low-level library. This error code should be treated as similar to INVALID_PARAMETER.

v2.SetSize

Sets the size of a local overlay.

This function cannot be called on guest-created overlays.

This function is a member of VDPOverlayClient_Interface.

Method Signature

```
VDPOverlay_Error (*v2.SetSize)(VDPOverlayClient_ContextId contextId, VDPOverlay_OverlayId overlayId, int32 width, int32 height);
```

Parameters

Parameter	Description
contextId	The ID returned from VDPOverlayClient.v1.Init().
overlayId	An overlay ID that was returned from a previous call to CreateOverlay().
width, height	The size of the overlay in pixels. If the size of the image specified in Update() does not match the size of the overlay, the image is drawn as specified by the layout mode.

Return Values

Value	Description
VDP_OVERLAY_ERROR_SUCCESS	No error. The function was successful.
VDP_OVERLAY_ERROR_NOT_INITIALIZED	Overlay API was not initialized.
VDP_OVERLAY_ERROR_INVALID_PARAMETER	The contextId or overlayID parameter was invalid.
VDP_OVERLAY_ERROR_NOT_LOCAL_OVERLAY	The overlayId of a guest-side overlay was passed to a function that can only be called on a local overlay.
VDP_OVERLAY_ERROR_HOST_OVERLAY_ERROR	There is an error with a low-level library. This error code should be treated as similar to INVALID_PARAMETER.

v2.Update

Updates the overlay with a new image. The updated image is displayed when the next frame is drawn.

This function is a member of VDPOverlayClient_Interface.

Method Signature

```
VDPOverlay_Error (*v2.Update)(VDPOverlayClient_ContextId contextId, VDPOverlay_WindowId windowId, void* pImage, int32 width, int32 height, int32 pitch, VDPOverlay_ImageFormat format, uint32 flags);
```

Parameters

Parameter	Description
contextId	The ID returned from VDPOverlayClient.v1.Init().
windowId	Window ID that was received from a previous OnWindowRegistered() event.
pImage	a pointer to the pixels to copy to the overlay. <ul style="list-style-type: none">If the VDP_OVERLAY_UPDATE_FLAG_SHARED_SURFACE flag is set pImage is the handle of a DirectX shared surface.If the format is a YUV format then pImage must point to a VDPOverlayClient_YUVImageData structure.
width	Width, in pixels, of the image pointed to by pImage. If the width of the image does not match the width of the overlay, the given image is drawn according the layout mode of the overlay.
height	Height, in pixels, of the image pointed to by pImage. If the height of the image does not match the height of overlay, the given image is drawn according the layout mode of the overlay.
pitch	Number of bytes that a single row of the image occupies. In the normal case, for BGRX images, this value is width multiplied by 4.
format	The pixel format of the image. This is one of the values in VDPOverlay_ImageFormat.
flags	<ul style="list-style-type: none">VDP_OVERLAY_UPDATE_FLAG_NONE - Place holder denoting that no flags are being passed.VDP_OVERLAY_UPDATE_FLAG_COPY_IMAGE - If set, a copy of the image data is made. If FALSE, no copy is made and the image data must remain valid until another call to VDPOverlayClient_Interface.v1/v2.Update() is made.VDP_OVERLAY_UPDATE_FLAG_SHARED_SURFACE - Allows a DirectX surface handle to be passed in place of a pointer to the image. Shared surfaces are not supported on all video cards or with all image formats, so the application must be prepared to fall back to not using this flag and passing a pointer to the image in system memory.VDP_OVERLAY_UPDATE_FLAG_EXCLUDE_FROM_READ_BACK - The overlay is not included as part of image returned from VDPScreenCapture_Interface.v1.ReadBackScreen().VDP_OVERLAY_UPDATE_FLAG_WRITE_IMAGE_FILE - If set the image is written to %TEMP%\Overlay<id>-<frame#>.png. This can be used when debugging to verify the image given to VDPOverlayClient_Interface.v1/v2.Update().

Return Values

Value	Description
VDP_OVERLAY_ERROR_SUCCESS	Image was updated.
VDP_OVERLAY_ERROR_NOT_INITIALIZED	Overlay API was not initialized.
VDP_OVERLAY_ERROR_INVALID_PARAMETER	The contextId, windowId, or pImage parameter was invalid.

v3.GetTopology

Retrieves the topology of the Horizon desktop.

This function is a member of VDPOverlayClient_Interface.

Method Signature

```
VDPOverlay_Error (*v3.GetTopology)(VDPOverlayClient_ContextId contextId, VDPOverlay_Rect* desktopBounds, int32* pszDesktopTopology, VDPOverlay_Rect* desktopTopology);
```

Parameters

Parameter	Description
contextId	The ID returned from VDPOverlayClient.v1.Init().
desktopBounds	Returns a rectangle which contains the bounding box for the entire Horizon desktop. Pass NULL to avoid returning this information.
pszDesktopTopology	A pointer to an int32. On input the value is the size of the desktopTopology array which will return the Horizon desktop topology. On output the value is the number of rectangles required to hold the entire desktop topology. This value may be larger than the size passed in if the desktopTopology array is too small to hold the entire desktop topology. Passing NULL is treated the same as *pszDesktopTopology == 0
desktopTopology	A pointer to an array which returns the rectangles that make up the Horizon desktop topology. Can be NULL only if pszDesktopTopology is NULL or *pszDesktopTopology == 0.

Return Values

Value	Description
VDP_OVERLAY_ERROR_SUCCESS	Current information retrieved
VDP_OVERLAY_ERROR_NOT_INITIALIZED	Overlay API was not initialized.
VDP_OVERLAY_ERROR_INVALID_PARAMETER	One of the parameter entries was invalid.
VDP_OVERLAY_ERROR_HOST_OVERLAY_ERROR	There is an error with a low-level library. This error code should be treated as similar to INVALID_PARAMETER.

v4.GetInfoString

Gets the current information string for the overlay as set by v4.SetInfoString().

This function is a member of VDPOverlayClient_Interface.

Method Signature

```
VDPOverlay_Error (*v4.GetInfoString)(VDPOverlayClient_ContextId contextId, VDPOverlay_WindowId windowId, char* infoStr, int32 infoStrSize);
```

Parameters

Parameter	Description
contextId	The ID returned from VDPOverlayClient.v1.Init().
windowId	The window ID that was cached from a prior OnWindowRegistered() event.

Parameter	Description
infoStr	A pointer to a buffer that is filled with the current information string.
infoStrSize	The size of the infoStr buffer.

Return Values

Value	Description
VDP_OVERLAY_ERROR_SUCCESS	Current information retrieved.
VDP_OVERLAY_ERROR_NOT_INITIALIZED	Overlay API was not initialized.
VDP_OVERLAY_ERROR_INVALID_PARAMETER	One of the parameter entries was invalid.
VDP_OVERLAY_ERROR_HOST_OVERLAY_ERROR	There is an error with a low-level library. This error code should be treated as similar to INVALID_PARAMETER.

v4.GetInfoStringProperties

Gets the current information string for the overlay as set by v4.SetInfoString().

This function is a member of VDPOverlayClient_Interface.

Method Signature

```
VDPOverlay_Error (*v4.GetInfoStringProperties)(VDPOverlayClient_ContextId contextId, VDPOverlay_WindowId windowId, VDPOverlayClient_InfoStringProperties *pProperties);
```

Parameters

Parameter	Description
contextId	The ID returned from VDPOverlayClient.v1.Init().
windowId	The window ID that was cached from a prior OnWindowRegistered() event.
pProperties	A pointer to a VDPOverlayClient_InfoStringProperties structure.

Return Values

Value	Description
VDP_OVERLAY_ERROR_SUCCESS	Current information retrieved.
VDP_OVERLAY_ERROR_NOT_INITIALIZED	Overlay API was not initialized.
VDP_OVERLAY_ERROR_INVALID_PARAMETER	One of the parameter entries was invalid.
VDP_OVERLAY_ERROR_HOST_OVERLAY_ERROR	There is an error with a low-level library. This error code should be treated as similar to INVALID_PARAMETER.

v4.SetInfoString

Sets a string that is rendered on top of the overlay. The string can contain arbitrary information which can assist with closed captioning or debugging information.

This function is a member of VDPOverlayClient_Interface.

Method Signature

```
VDPOverlay_Error (*v4.SetInfoString)(VDPOverlayClient_ContextId contextId, VDPOverlay_WindowId windowId, const char* infoStr);
```

Parameters

Parameter	Description
contextId	The ID returned from VDPOverlayClient.v1.Init().
windowId	The window ID that was cached from a prior OnWindowRegistered() event.

infoStr The information string. The maximum string length is VDP_OVERLAY_INFO_STR_MAX_LEN bytes, including the NULL terminator.
The information string can contain the following macros:

Macro	Definition
\$(FPS)	FPS or image format (default)
\$(IMAGE_SIZE)	Source image size (WxH)
\$(IMAGE_ASPECT)	Source image aspect ratio (X.XX)
\$(IMAGE_FORMAT)	Source image format
\$(OVERLAY_ID)	Overlay ID
\$(OVERLAY_POS)	Overlay position (X,Y)
\$(OVERLAY_SIZE)	Overlay size (WxH)
\$(OVERLAY_ASPECT)	Overlay image aspect ratio (X.XX)
\$(OVERLAY_LAYER)	Overlay layer
\$(OVERLAY_LAYOUT)	Overlay layout mode
\$(OVERLAY_SURFACE)	Overlay surface type
\$(OVERLAY_FPS)	Overlay frame rate
\$(OVERLAY_FRAME_NUM)	Overlay frame number
\$(HORIZON_FPS) <i>replaced \$(VIEW_FPS) in release 2503</i>	Horizon client frame rate

Parameter	Description
\$(HORIZON_FRAME_NUM) <i>replaced \$(VIEW_FRAME_NUM) in release 2503</i>	Horizon client frame number
\$(HORIZON_WINDOW_SIZE) <i>replaced \$(VIEW_WINDOW_SIZE) in release 2503</i>	Horizon client window size
\$(HORIZON_PROTOCOL) <i>replaced \$(VIEW_PROTOCOL) in release 2503</i>	Horizon client protocol (Blast/PCoIP)
\$(REMOTE_DESKTOP_SIZE)	Horizon remote desktop size (WxH)
\$(TIME)	Current time
\$(DATE)	Current date

The following escape characters are recognized, assuming that the string is read from a file or the registry. When hardcoding the information string in C/C++ code, you must also escape the backslash character itself.

Escape Character	Definition
\n	New line; the LF character ('\n' in C/C++) is also a new line
\\$	Dollar sign
\\	Backslash

Return Values

Value	Description
VDP_OVERLAY_ERROR_SUCCESS	No error. The function was successful.
VDP_OVERLAY_ERROR_NOT_INITIALIZED	Overlay API was not initialized.
VDP_OVERLAY_ERROR_INVALID_PARAMETER	One of the parameter entries was invalid.
VDP_OVERLAY_ERROR_HOST_OVERLAY_ERROR	There is an error with a low-level library. This error code should be treated as similar to INVALID_PARAMETER.

v4.SetInfoStringProperties

Sets properties for how the information string is rendered.

This function is a member of VDPOverlayClient_Interface.

Method Signature

```
VDPOverlay_Error (*v4.SetInfoStringProperties)(VDPOverlayClient_ContextId contextId, VDPOverlay_WindowId windowId, VDPOverlayClient_InfoStringProperties *pProperties);
```


Parameters

Parameter	Description
contextId	The ID returned from VDPOverlayClient.v1.Init().
windowId	The window ID that was cached from a prior OnWindowRegistered() event.
pProperties	A pointer to a VDPOverlayClient_InfoStringProperties structure. As a best practice, initialize the structure by calling v4.GetInfoStringProperties() if there are properties that you don't want to change.

Return Values

Value	Description
VDP_OVERLAY_ERROR_SUCCESS	No error. The function was successful.
VDP_OVERLAY_ERROR_NOT_INITIALIZED	Overlay API was not initialized.
VDP_OVERLAY_ERROR_INVALID_PARAMETER	One of the parameter entries was invalid.
VDP_OVERLAY_ERROR_HOST_OVERLAY_ERROR	There is an error with a low-level library. This error code should be treated as similar to INVALID_PARAMETER.

v5.SetRemoteWindowId

Associates a local overlay with a remote window.

Guest side overlays are automatically associated with the remote window they are tracking. An error is returned when trying to change the remote window ID for a guest side overlay.

Client side local overlays have an associated remote window ID of 0 by default.

When using the VDPScreenCapture ReadBackWindow function knowing which overlays are associated with a particular remote window is helpful in knowing which overlays to include and/or exclude in the image returned.

Only overlays with a matching remote window ID are eligible to be included in the image returned from ReadBackWindow() according to the VDPScreenCapture_OverlayOptions parameter.

Overlays with a non-matching remote window ID are always excluded from the image returned by ReadBackWindow().

This function is a member of VDPOverlayClient_Interface.

Method Signature

```
VDPOverlay_Error (*v5.SetRemoteWindowId)(VDPOverlayClient_ContextId contextId, VDPOverlay_OverlayId overlayId, uint64 hRemoteWindow);
```

Parameters

Parameter	Description
contextId	The ID returned from VDPOverlayClient.v1.Init().
overlayId	An overlay ID that was returned from a previous call to v2.CreateOverlay().
hRemoteWindow	The operating system window identifier to associate with this overlay.

Return Values

Value	Description
VDP_OVERLAY_ERROR_SUCCESS	The operating system window identifier is associated with the local overlay.
VDP_OVERLAY_ERROR_NOT_INITIALIZED	Overlay API was not initialized.
VDP_OVERLAY_ERROR_INVALID_PARAMETER	One of the parameter entries was invalid.
VDP_OVERLAY_ERROR_HOST_OVERLAY_ERROR	There is an error with a low-level library. This error code should be treated as similar to INVALID_PARAMETER.

v5.GetRemoteWindowId

Returns the remote desktop window ID associated with an overlay.

Guest side overlays are automatically associated with the remote window they are tracking. An error is returned when trying to change the remote window ID for a guest side overlay.

Client side local overlays have an associated remote window ID of 0 by default and can only be changed by calling v5.SetRemoteWindowId().

This function is a member of VDPOverlayClient_Interface.

Method Signature

```
VDPOverlay_Error (*v5.GetRemoteWindowId)(VDPOverlayClient_ContextId contextId, VDPOverlay_OverlayId overlayId, uint64 *phRemoteWindow);
```

Parameters

Parameter	Description
contextId	The ID returned from VDPOverlayClient.v1.Init().
overlayId	An overlay ID that was returned from a previous call to v2.CreateOverlay().
phRemoteWindow	Returns the operating system window identifier associated with the local overlay. Returns 0 if there isn't a remote window associated with the overlay.

Return Values

Value	Description
VDP_OVERLAY_ERROR_SUCCESS	The operating system window identifier is associated with the local overlay.
VDP_OVERLAY_ERROR_NOT_INITIALIZED	Overlay API was not initialized.
VDP_OVERLAY_ERROR_INVALID_PARAMETER	One of the parameter entries was invalid.
VDP_OVERLAY_ERROR_HOST_OVERLAY_ERROR	There is an error with a low-level library. This error code should be treated as similar to INVALID_PARAMETER.

VDPScreenCapture Functions

With VDPScreenCapture_Interface functions, you can access the image of the entire remote desktop, an area of the desktop or the windows that belong to a particular application. The vdpScreenCapture.h header file defines the set of functions to use in order to support screen capture functionality in an application.

v1.Init

This function initializes the client-side overlay library. This method must be the first method called in the client-side Overlay API.

This function is a member of VDPScreenCapture_Interface.

Method Signature

```
VDPScreenCapture_Error (*v1.Init)(const VDPScreenCapture_Sink* sink, void* userData, VDPScreenCapture_ContextId * pContextId);
```

Parameters

Parameter	Description
sink	Function pointers called when events are generated by the Screen Capture API.
userData	Parameter that is passed to sink callbacks. Can be NULL.
pContextId	Returns an ID that identifies the instance of the API for this plug-in instance.

Return Values

Value	Description
VDP_SCREEN_CAPTURE_ERROR_SUCCESS	Screen Capture API was initialized.
VDP_SCREEN_CAPTURE_ERROR_HOST_NOT_READY	The Screen Capture API has to exchange information with the Horizon Agent running on the remote desktop before it is ready to be used. Calling this function before a connection is in place or even too quickly after a connection has been made will result in this error. Try calling the function again after a few seconds.
VDP_SCREEN_CAPTURE_ERROR_INVALID_PARAMETER	The sink or pContextId parameter is NULL or sink reported an invalid version.

v1.Exit

Performs cleanup operations and releases all allocated resources.

This function is a member of VDPScreenCapture_Interface.

Method Signature

```
VDPScreenCapture_Error (*v1.Exit)(VDPScreenCapture_ContextId contextId);
```

Parameters

Parameter	Description
contextId	The ID returned from VDPScreenCapture.v1.Init().

Return Values

Value	Description
VDP_SCREEN_CAPTURE_ERROR_SUCCESS	Screen Capture API was properly shut down.
VDP_SCREEN_CAPTURE_ERROR_INVALID_PARAMETER	contextId is invalid.

v1.GetLocalTopology

Retrieves the topology of the local Horizon client.

This function is a member of VDPScreenCapture_Interface.

Method Signature

```
VDPScreenCapture_Error (*v1.GetLocalTopology)(VDPScreenCapture_ContextId contextId, VDPScreenCapture_Rect* pBounds, int32* pszTopology, VDPScreenCapture_Rect* pTopology);
```

Parameters

Parameter	Description
contextId	The ID returned from VDPScreenCapture.v1.Init().
pBounds	Returns a rectangle which contains the bounding box for the entire Horizon client. Pass NULL to avoid returning the information.
pszTopology	A pointer to an int32. On input the value is the size of the pTopology array which will return the Horizon client topology. On output the value is the number of rectangles required to hold the entire Horizon client topology; which may be larger than the size passed in if the pTopology array is too small to hold the entire topology. Passing NULL is treated the same as *pszTopology == 0.
pTopology	A pointer to an array which returns the rectangles that make up the Horizon client topology. Can be NULL only if pszTopology is NULL or *pszTopology == 0.

Return Values

Value	Description
VDP_SCREEN_CAPTURE_ERROR_SUCCESS	The topology information was returned.
VDP_SCREEN_CAPTURE_ERROR_INVALID_PARAMETER	contextId is invalid, szTopology is < 0 or pTopology is NULL when szTopology > 0.
VDP_SCREEN_CAPTURE_ERROR_NOT_ALLOWED_IN_APP_MODE	This function cannot be called when the Horizon client is in application mode.

v1.GetRemoteTopology

Retrieves the topology of the remote desktop.

This function is a member of VDPScreenCapture_Interface.

Method Signature

```
VDPScreenCapture_Error (*v1.GetRemoteTopology)(VDPScreenCapture_ContextId contextId, VDPScreenCapture_Rect* pBounds, int32* pszTopology, VDPScreenCapture_Rect* pTopology);
```

Parameters

Parameter	Description
contextId	The ID returned from VDPScreenCapture.v1.Init().
pBounds	Returns a rectangle which contains the bounding box for the entire remote desktop. Pass NULL to avoid returning the information.
pszTopology	A pointer to an int32. On input the value is the size of the pTopology array which will return the remote desktop topology. On output the value is the number of rectangles required to hold the entire remote desktop topology; which may be larger than the size passed in if the pTopology array is too small to hold the entire topology. Passing NULL is treated the same as *pszTopology == 0.
pTopology	A pointer to an array which returns the rectangles that make up the remote desktop topology. Can be NULL only if pszTopology is NULL or *pszTopology == 0.

Return Values

Value	Description
VDP_SCREEN_CAPTURE_ERROR_SUCCESS	The topology information was returned.
VDP_SCREEN_CAPTURE_ERROR_INVALID_PARAMETER	contextId is invalid, szTopology is < 0 or pTopology was NULL when szTopology > 0.
VDP_SCREEN_CAPTURE_ERROR_NOT_ALLOWED_IN_APP_MODE	This function can not be called when the Horizon client is in application mode.

v1.GetHostWindowByRect

Returns the host window ID that contains the given rectangle.

This function is a member of VDPScreenCapture_Interface.

Method Signature

```
VDPScreenCapture_Error (*v1.GetHostWindowByRect)(VDPScreenCapture_ContextId contextId, const VDPScreenCapture_Rect* pRect, uintptr_t* pHostWindow);
```

Parameters

Parameter	Description
contextId	The ID returned from VDPScreenCapture.v1.Init().
pRect	The rectangle that determines which host window to return in local coordinates. The host window must contain the entire rectangle.
pHostWindow	Returns the host window ID (0 on error). The host window ID can change while Horizon is running so it should not be cached.

Return Values

Value	Description
VDP_SCREEN_CAPTURE_ERROR_SUCCESS	The host window ID was returned.
VDP_SCREEN_CAPTURE_ERROR_INVALID_PARAMETER	contextId is invalid, pRect is NULL or pHostWindow is NULL.
VDP_SCREEN_CAPTURE_ERROR_NOT_ALLOWED_IN_APP_MODE	This function can not be called when the Horizon client is in application mode.
VDP_SCREEN_CAPTURE_ERROR_HOST_ERROR	A host window was not found.

v1.GetHostWindowByPoint

Returns the host window ID that contains the given point.

This function is a member of VDPScreenCapture_Interface.

Method Signature

```
VDPScreenCapture_Error (*v1.GetHostWindowByPoint)(VDPScreenCapture_ContextId contextId, int32 x, int32 y, uintptr_t* pHostWindow);
```

Parameters

Parameter	Description
contextId	The ID returned from VDPScreenCapture.v1.Init().
x,y	The point that determines which host window to return in local coordinates. The host window must contain this point.
pHostWindow	Returns the host window ID (0 on error). The host window ID can change while Horizon is running so it should not be cached.

Return Values

Value	Description
VDP_SCREEN_CAPTURE_ERROR_SUCCESS	The host window ID was returned.

Value	Description
VDP_SCREEN_CAPTURE_ERROR_INVALID_PARAMETER	contextId is invalid, pRect is NULL or pHostWindow is NULL.
VDP_SCREEN_CAPTURE_ERROR_NOT_ALLOWED_IN_APP_MODE	This function can not be called when the Horizon client is in application mode.
VDP_SCREEN_CAPTURE_ERROR_HOST_ERROR	A host window was not found.

v1.MapLocalToRemoteRect

Maps a rectangle from local coordinates to remote coordinates.

This function is a member of VDPScreenCapture_Interface.

Method Signature

```
VDPScreenCapture_Error (*v1.MapLocalToRemoteRect)(VDPScreenCapture_ContextId contextId, const VDPScreenCapture_Rect* pLocalRect, VDPScreenCapture_Rect* pRemoteRect);
```

Parameters

Parameter	Description
contextId	The ID returned from VDPScreenCapture.v1.Init().
pLocalRect	The rectangle, in local coordinates, that is mapped to remote coordinates. The rectangle can not span multiple local topology rectangles.
pRemoteRect	Returns the mapped rectangle in remote coordinates.

Return Values

Value	Description
VDP_SCREEN_CAPTURE_ERROR_SUCCESS	The local coordinates were mapped to remote coordinates.
VDP_SCREEN_CAPTURE_ERROR_INVALID_PARAMETER	contextId is invalid, pLocalRect is NULL or pRemoteRect is NULL.
VDP_SCREEN_CAPTURE_ERROR_NOT_ALLOWED_IN_APP_MODE	This function can not be called when the Horizon client is in application mode.
VDP_SCREEN_CAPTURE_ERROR_HOST_ERROR	The local coordinates could not be mapped to remote coordinates.

v1.MapRemoteToLocalRect

Maps a rectangle from remote coordinates to local coordinates.

This function is a member of VDPScreenCapture_Interface.

Method Signature

```
VDPScreenCapture_Error (*v1.MapRemoteToLocalRect)(VDPScreenCapture_ContextId contextId, const VDPScreenCapture_Rect* pRemoteRect, VDPScreenCapture_Rect* pLocalRect);
```


Parameters

Parameter	Description
contextId	The ID returned from VDPScreenCapture.v1.Init().
pRemoteRect	The rectangle, in remote coordinates, that is mapped to local coordinates. The rectangle can not span multiple remote topology rectangles.
pLocalRect	Returns the mapped rectangle in local coordinates.

Return Values

Value	Description
VDP_SCREEN_CAPTURE_ERROR_SUCCESS	The remote coordinates were mapped to local coordinates.
VDP_SCREEN_CAPTURE_ERROR_INVALID_PARAMETER	contextId is invalid, pRemoteRect is NULL or pLocalRect is NULL.
VDP_SCREEN_CAPTURE_ERROR_NOT_ALLOWED_IN_APP_MODE	This function can not be called when the Horizon client is in application mode.
VDP_SCREEN_CAPTURE_ERROR_HOST_ERROR	The remote coordinates could not be mapped to local coordinates.

v2.IsApplicationMode

Determines if the Horizon client is running in *'Application Mode'*. *'Application Mode'* also known as *'Unity Mode'* or *'Seamless Windows'* runs an application on a remote desktop and only displays the windows for the application. If not in *'Application Mode'* the Horizon client is running in *'Desktop Mode'* where the entire remote desktop is displayed.

This function is a member of VDPScreenCapture_Interface.

Method Signature

```
VDPScreenCapture_Error (*v2.IsApplicationMode)(VDPScreenCapture_ContextId contextId, Bool* pIsApplicationMode);
```

Parameters

Parameter	Description
contextId	The ID returned from VDPScreenCapture.v1.Init().
pIsApplicationMode	Returns TRUE if the Horizon Client is in application mode.

Return Values

Value	Description
VDP_SCREEN_CAPTURE_ERROR_SUCCESS	The application mode was returned.
VDP_SCREEN_CAPTURE_ERROR_INVALID_PARAMETER	contextId is invalid or pIsApplicationMode is NULL.

v2.ReadBackScreen

Returns an image with the remote desktop screen contents. When finished with the image it must be released by calling `v2.ReadBackRelease()`.

This function is a member of `VDPScreenCapture_Interface`.

Method Signature

```
VDPScreenCapture_Error (*v2.ReadBackScreen)(VDPScreenCapture_ContextId contextId, VDPScreenCapture_ReadBackParameters* pParams, VDPScreenCapture_ImageInfo* plmageInfo);
```

Parameters

Parameter	Description
contextId	The ID returned from <code>VDPScreenCapture.v1.Init()</code> .
pParams	The input parameters used to determine which part of the remote desktop to read.
plmageInfo	The resulting image. You must release the image by calling <code>v2.ReadBackRelease()</code> with the handle returned in <code>VDPScreenCapture_ImageInfo.v1.handle</code> .

Return Values

Value	Description
<code>VDP_SCREEN_CAPTURE_ERROR_SUCCESS</code>	An image of the remote desktop screen was returned.
<code>VDP_SCREEN_CAPTURE_ERROR_INVALID_PARAMETER</code>	<code>contextId</code> is invalid, <code>pParams</code> is NULL, <code>plmageInfo</code> is NULL or one of the values in <code>pParams</code> is not valid.
<code>VDP_SCREEN_CAPTURE_ERROR_NOT_ALLOWED_IN_APP_MODE</code>	This function can not be called when the Horizon client is in application mode.
<code>VDP_SCREEN_CAPTURE_ERROR_HOST_ERROR</code>	There was an error when attempting to capture an image of the remote desktop screen.

v2.ReadBackRelease

Releases resources held by the Horizon client to manage the image returned from `v2.ReadBackScreen()`.

This function is a member of `VDPScreenCapture_Interface`.

Method Signature

```
VDPScreenCapture_Error (*v2.ReadBackRelease)(VDPScreenCapture_ContextId contextId, VDPScreenCapture_ImageHandle hImage);
```

Parameters

Parameter	Description
contextId	The ID returned from <code>VDPScreenCapture.v1.Init()</code> .

Parameter	Description
hImage	The handle returned from v2.ReadBackScreen() in VDPScreenCapture_ImageInfo.v1.handle.

Return Values

Value	Description
VDP_SCREEN_CAPTURE_ERROR_SUCCESS	The image was successfully released.
VDP_SCREEN_CAPTURE_ERROR_INVALID_PARAMETER	contextId is invalid or hImage is invalid.
VDP_SCREEN_CAPTURE_ERROR_NOT_ALLOWED_IN_APP_MODE	This function can not be called when the Horizon client is in application mode.
VDP_SCREEN_CAPTURE_ERROR_HOST_ERROR	There was error when trying to release the image.

v3.IsReadBackWindowSupported

Determines if the ReadBackWindow interface is supported. The ReadBackWindow interface is dependent on the protocol and the version of the Horizon Agent running on the remote desktop to function properly.

This function is a member of VDPScreenCapture_Interface.

Method Signature

```
VDPScreenCapture_Error (*v3.IsReadBackWindowSupported)();
```

Return Values

Value	Description
VDP_SCREEN_CAPTURE_ERROR_SUCCESS	The ReadBackWindow interface is supported.
VDP_SCREEN_CAPTURE_ERROR_READBACK_WINDOW_NOT_READY	The ReadBackWindow interface has to exchange information with the Horizon Agent on the remote desktop before it is ready to be used. Calling this function before a connection is in place or even too quickly after a connection has been made will result in this error. Try calling the function again after a few seconds.
VDP_SCREEN_CAPTURE_ERROR_UNSUPPORTED_PROTOCOL	The ReadBackWindow interface requires the Blast protocol. Attempts to use the interface with other protocols will fail.
VDP_SCREEN_CAPTURE_ERROR_UNSUPPORTED_AGENT	The version of the Horizon Agent running on the remote desktop does not support the ReadbackWindow interface.

v3.ReadBackWindowBegin

Starts tracking a remote window; e.g. a window on the remote desktop.

- The API requires some time to begin tracking a remote window. v3.ReadBackWindow() will return the error code VDP_SCREEN_CAPTURE_ERROR_READBACK_WINDOW_NOT_READY until it is ready to return images.

- The callback `VDPScreenCapture_Sink.v3.OnReadBackWindowReady()` will be called when the API is ready to return images.
- The callback `VDPScreenCapture_Sink.v3.OnReadBackWindowDestroyed()` will be called if the remote window is destroyed while being tracked.
- The callback `VDPScreenCapture_Sink.v3.OnReadBackWindowRemoteError()` will be called if there is an error when setting up the tracking.
- When finished, you must release the resources used to track the remote window by calling `v3.ReadBackWindowEnd()`.
- If there are multiple calls to `v3.ReadBackWindowBegin()` to track the same window, each will return a different `ReadBackWindow` handle. Each handle will have to be released by calling `v3.ReadBackWindowEnd()`.

This function is a member of `VDPScreenCapture_Interface`.

Method Signature

```
VDPScreenCapture_Error (*v3.ReadBackWindowBegin)(VDPScreenCapture_ContextId contextId, VDPScreenCapture_HWND hWnd, uint32 processId, uint32 flags, VDPScreenCapture_ReadBackWindowHandle* phReadBackWindow);
```

Parameters

Parameter	Description
contextId	The ID returned from <code>VDPScreenCapture.v1.Init()</code> .
hWnd	The Windows HWND of the remote window to track.
processId	The Windows process ID associated with the HWND. An error will be returned if the HWND does not belong to this process.
flags	Options to apply to the read back. See <code>VDP_SCREEN_CAPTURE_READBACK_BEGIN_*</code> flags for details.
phReadBackWindow	Returns a handle to be used with <code>v3.ReadBackWindow()</code> and <code>v3.ReadBackWindowEnd()</code> . This handle is also passed to the <code>ReadBackWindow</code> callbacks.

Return Values

Value	Description
<code>VDP_SCREEN_CAPTURE_ERROR_SUCCESS</code>	The tracking of the remote window is being initialized by asking the Horizon Agent to track the window. One of the sink functions will be called to notify you if setting up the tracking was successful or not.
<code>VDP_SCREEN_CAPTURE_ERROR_INVALID_PARAMETER</code>	contextId is invalid, hWnd is 0, processId is 0 or phReadBackWindow is NULL.
<code>VDP_SCREEN_CAPTURE_ERROR_UNSUPPORTED_PROTOCOL</code>	The <code>ReadBackWindow</code> interface requires the Blast protocol. Attempts to use the interface with other protocols will fail.
<code>VDP_SCREEN_CAPTURE_ERROR_UNSUPPORTED_AGENT</code>	The version of the Horizon Agent running on the remote desktop does not support the <code>ReadbackWindow</code> interface.
<code>VDP_SCREEN_CAPTURE_ERROR_UNSUPPORTED_OPTION</code>	One of the options in the flags parameter is not supported.

Value	Description
VDP_SCREEN_CAPTURE_ERROR_HOST_ERROR	There was an error in setting up the tracking of the remote window.

v3.ReadBackWindowEnd

Stops tracking a remote window and releases the associated resources.

This function is a member of VDPScreenCapture_Interface.

Method Signature

```
VDPScreenCapture_Error (*v3.ReadBackWindowEnd)(VDPScreenCapture_ContextId contextId, VDPScreenCapture_ReadBackWindowHandle hReadBackWindow);
```

Parameters

Parameter	Description
contextId	The ID returned from VDPScreenCapture.v1.Init().
hReadBackWindow	The handle returned from v3.ReadBackWindowBegin().

Return Values

Value	Description
VDP_SCREEN_CAPTURE_ERROR_SUCCESS	The window is no longer being tracked.
VDP_SCREEN_CAPTURE_ERROR_INVALID_PARAMETER	contextId is invalid or hReadBackWindow is invalid.
VDP_SCREEN_CAPTURE_ERROR_HOST_ERROR	There was an error when attempting to stop tracking the window.

v3.ReadBackWindow

Returns an image with the remote window contents. When finished with the image it must be released by calling v2.ReadBackRelease().

This function is a member of VDPScreenCapture_Interface.

Method Signature

```
VDPScreenCapture_Error (*v3.ReadBackWindow)(VDPScreenCapture_ContextId contextId, VDPScreenCapture_ReadBackWindowHandle hReadBackWindow, VDPScreenCapture_ReadBackParameters* pParams, VDPScreenCapture_ImageInfo* pImageInfo);
```

Parameters

Parameter	Description
contextId	The ID returned from VDPScreenCapture.v1.Init().

Parameter	Description
hReadBackWindow	The handle returned from v3.ReadBackWindowBegin().
pParams	The input parameters used to determine which part of the window to read. Set VDPScreenCapture_ReadBackParameters.v1.srcRect to zeros to read the entire window.
pImageInfo	The resulting image. You must release the image by calling v2.ReadBackRelease() with the handled return in VDPScreenCapture_ImageInfo.v1.handle.

Return Values

Value	Description
VDP_SCREEN_CAPTURE_ERROR_SUCCESS	The image was returned.
VDP_SCREEN_CAPTURE_ERROR_INVALID_PARAMETER	contextId is invalid, hReadBackWindow is invalid, pParams is NULL, pImageInfo is NULL or one of the values in pParams is invalid.
VDP_SCREEN_CAPTURE_ERROR_READBACK_WINDOW_NOT_READY	v3.ReadBackWindow() was called before the tracking is fully initialized.
VDP_SCREEN_CAPTURE_ERROR_READBACK_WINDOW_DESTROYED	The window being tracked on the remote desktop has been destroyed.
VDP_SCREEN_CAPTURE_ERROR_HOST_ERROR	There was an error when attempting to capture the image of the window.

v4.GetReadBackCapabilities

Determines the ReadBack API's capabilities; e.g. which features / options are available. The features that are available are determined by the Horizon Agent and Horizon Client versions as well as the remote protocol being used.

This function is a member of VDPScreenCapture_Interface.

Method Signature

```
VDPScreenCapture_Error (*v4.GetReadBackCapabilities)(uint32* pReadbackCaps);
```

Parameters

Parameter	Description
pReadbackCaps	The return capabilities. See VDP_SCREEN_CAPTURE_READBACK_CAPS for more information.

Return Values

Value	Description
VDP_SCREEN_CAPTURE_ERROR_SUCCESS	The capabilities were returned.
VDP_SCREEN_CAPTURE_ERROR_READBACK_WINDOW_NOT_READY	The Screen Capture API has to exchange information with the Horizon Agent running on the remote desktop before it is ready

Value	Description
	to be used. Calling this function before a connection is in place or even too quickly after a connection has been made will result in this error. Try calling the function again after a few seconds.

v4.ReadBackWindowBegin

Starts tracking a remote window; e.g. a window on the remote desktop. This function is similar to `v3.ReadBackWindowBegin()` but it provides a way to include auxiliary windows along with the primary window. Auxiliary windows are top level window that will be included with the readback image if they overlap the primary window.

The API requires some time to begin tracking a remote window. `v3.ReadBackWindow()` will return the error code `VDP_SCREEN_CAPTURE_ERROR_READBACK_WINDOW_NOT_READY` until it is ready to return images.

The callback `VDPScreenCapture_Sink.v3.OnReadBackWindowReady()` will be called when the API is ready to return images.

The callback `VDPScreenCapture_Sink.v3.OnReadBackWindowDestroyed()` will be called if the remote window is destroyed while being tracked.

The callback `VDPScreenCapture_Sink.v3.OnReadBackWindowRemoteError()` will be called if there is an error when setting up the tracking.

When finished, you must release the resources used to track the remote window by calling `v3.ReadBackWindowEnd()`.

If there are multiple calls to `v3/v4.ReadBackWindowBegin()` to track the same window, each will return a different `ReadBackWindow` handle. Each handle will have to be released by calling `v3.ReadBackWindowEnd()`. But the first call defines the parameters and the subsequent calls must provide the same parameters, including the list of auxiliary windows.

This function is a member of `VDPScreenCapture_Interface`.

Method Signature

```
VDPScreenCapture_Error (*v4.ReadBackWindowBegin)(VDPScreenCapture_ContextId contextId, VDPScreenCapture_HWND hWnd, uint32 processId, VDPScreenCapture_HWND* hWndAux, uint32* processIdAux, int32 auxWindowCount, uint32 flags, VDPScreenCapture_ReadBackWindowHandle* phReadBackWindow);
```

Parameters

Parameter	Description
<code>contextId</code>	The ID returned from <code>VDPScreenCapture.v1.Init()</code> .
<code>hWnd</code>	The Windows HWND of the remote window to track.
<code>processId</code>	The Windows process ID associated with the HWND. An error will be returned if the HWND does not belong to this process.

Parameter	Description
hWndAux	An array of Windows HWNDs to include with the readback image. These images will only be included with the readback image if they overlap the primary window.
processIdAux	An array of process IDs associated with the corresponding hWndAux. An error will be returned if the HWND does not belong to the corresponding process.
auxWindowCount	The count of auxiliary windows in the hWndAux array and process IDs in the processIdAux array. hWndAux and processIdAux may be NULL if auxWindowCount is 0.
flags	Options to apply to the read back. See VDP_SCREEN_CAPTURE_READBACK_BEGIN_* flags for more information.
phReadBackWindow	Returns a handle to be used with v3.ReadBackWindow() and v3.ReadBackWindowEnd(). This handle is also passed to the ReadBackWindow callbacks.

Return Values

Value	Description
VDP_SCREEN_CAPTURE_ERROR_SUCCESS	The tracking of the remote window is being initialized by asking the Horizon Agent to track the window. One of the sink functions will be called to notify you if setting up the tracking was successful or not.
VDP_SCREEN_CAPTURE_ERROR_INVALID_PARAMETER	contextId is invalid, hWnd is 0, processId is 0 , hWndAux/auxWindowCount is NULL (and auxWindowCount > 0), or phReadBackWindow is NULL.
VDP_SCREEN_CAPTURE_ERROR_UNSUPPORTED_PROTOCOL	The ReadBackWindow interface requires the Blast protocol. Attempts to use the interface with other protocols will fail.
VDP_SCREEN_CAPTURE_ERROR_UNSUPPORTED_AGENT	The version of the Horizon Agent running on the remote desktop does not support the ReadbackWindow interface.
VDP_SCREEN_CAPTURE_ERROR_UNSUPPORTED_OPTION	One of the options in the flags parameter is not supported or auxiliary windows are not supported.
VDP_SCREEN_CAPTURE_ERROR_HOST_ERROR	There was an error in setting up the tracking of the remote window.

v4.GetReadBackWindowInfo

Retrieves information about the given readback window including state, location, etc.

This function is a member of VDPScreenCapture_Interface.

Method Signature

```
VDPScreenCapture_Error (*v4.GetReadBackWindowInfo)(VDPScreenCapture_ContextId contextId,
VDPScreenCapture_ReadBackWindowHandle hReadBackWindow, VDPScreenCapture_ReadBackWindowInfo* pReadbackWindowInfo);
```

Parameters

Parameter	Description
contextId	The ID returned from VDPScreenCapture.v1.Init().
hReadBackWindow	The handle returned from v3/v4.ReadBackWindowBegin().

Parameter	Description
pReadbackWindowInfo	Returns the information about the readback window.

Return Values

Value	Description
VDP_SCREEN_CAPTURE_ERROR_SUCCESS	The information about the readback window is returned.
VDP_SCREEN_CAPTURE_ERROR_INVALID_PARAMETER	contextId is invalid, hReadBackWindow is invalid or pReadbackWindowInfo is NULL.
VDP_SCREEN_CAPTURE_ERROR_HOST_ERROR	There was an error in getting the information on the readback window.

v4.RegisterForReadBackRequests

Registers for screen and window read back requests. When v2.ReadBackScreen() or v3.ReadBackWindow() is called the Screen Capture API will issue a callback where an image can be provided so that it is included in the read back image.

VDPScreenCapture_Sink.v4.OnReadBackRequest() will be invoked when a read back of a screen or window is being done. The callback can then provide an image to be included in the read back image returned by calling v4.ReadBackRequestUpdate().

Each registration can provide an image for a single area of the screen. If you have multiple areas of the screen that need to be updated then you will need to register for multiple callbacks. This applies even if there are multiple areas associated with the same window; in this case that window would have to be registered multiple times.

This function is a member of VDPScreenCapture_Interface.

Method Signature

```
VDPScreenCapture_Error (*v4.RegisterForReadBackRequests)(VDPScreenCapture_ContextId contextId, VDPScreenCapture_HWND hWnd, VDPScreenCapture_ReadBackRequestId* pReadBackRequestId);
```

Parameters

Parameter	Description
contextId	The ID returned from VDPScreenCapture.v1.Init().
hWnd	The Windows HWND of a window on the remote desktop. If set to NULL then you will only receive callbacks when v2.ReadBackScreen() is called. If set to non-NULL you will receive callbacks for v2.ReadBackScreen() as well as calls to v3.ReadBackWindow() with a matching window handle.
pReadBackRequestId	Returns an ID to be used with v4.CaptureRequestUpdate() and v4.UnregisterForReadBackRequests(). This ID is also passed to the VDPScreenCapture_Sink.v4.OnReadBackRequest() callback.

Return Values

Value	Description
VDP_SCREEN_CAPTURE_ERROR_SUCCESS	The window is registered for callbacks.
VDP_SCREEN_CAPTURE_ERROR_INVALID_PARAMETER	contextId is invalid or pReadBackRequestId is NULL.
VDP_SCREEN_CAPTURE_ERROR_HOST_ERROR	There was an error registering the window.

v4.UnregisterForReadBackRequests

Unregisters a read back request and releases its resources.

This function is a member of VDPScreenCapture_Interface.

Method Signature

```
VDPScreenCapture_Error (*v4.UnregisterForReadBackRequests)(VDPScreenCapture_ContextId contextId,  
VDPScreenCapture_ReadBackRequestId readBackRequestId);
```

Parameters

Parameter	Description
contextId	The ID returned from VDPScreenCapture.v1.Init().
readBackRequestId	The ID returned from VDPScreenCapture.v4.RegisterForReadBackRequests().

Return Values

Value	Description
VDP_SCREEN_CAPTURE_ERROR_SUCCESS	The window is unregistered for callbacks.
VDP_SCREEN_CAPTURE_ERROR_INVALID_PARAMETER	contextId is invalid or readBackRequestId is invalid.
VDP_SCREEN_CAPTURE_ERROR_HOST_ERROR	There was an error unregistering the window.

v4.ReadBackRequestUpdate

Provides an image to include in a read back image. While this function can be called at any time, it is normally called during the v4.OnReadBackRequest() callback to provide an image. A copy of the image is made and is used in all further read back requests until the image is either replaced or removed by the next call to v4.ReadBackRequestUpdate().

This function is a member of VDPScreenCapture_Interface.

Method Signature

```
VDPScreenCapture_Error (*v4.ReadBackRequestUpdate)(VDPScreenCapture_ContextId contextId, VDPScreenCapture_ReadBackRequestId  
readBackRequestId, VDPScreenCapture_ReadBackRequestParams* pRequestParams);
```

Parameters

Parameter	Description
contextId	The ID returned from VDPScreenCapture.v1.Init().
readBackRequestId	The ID returned from VDPScreenCapture.v4.RegisterForReadBackRequests().
pRequestParams	A structure that contains the image to be included in a read back screen or window capture image. Pass NULL to remove any previously set image.

Return Values

Value	Description
VDP_SCREEN_CAPTURE_ERROR_SUCCESS	The window is unregistered for callbacks.
VDP_SCREEN_CAPTURE_ERROR_INVALID_PARAMETER	contextId is invalid, readBackRequestId is invalid or one of the values in pRequestParams is invalid.
VDP_SCREEN_CAPTURE_ERROR_HOST_ERROR	There was an error applying the image to the read back image.

Channel Sinks

To interact with and receive notifications of changes, you must register sinks with the Horizon Session Enhancement APIs. Channel sinks are common sinks.

These functions are members of `VDPService_ChannelNotifySink`.

v1.OnChannelStateChanged

This method is invoked when there is a change in the channel connection that this plug-in instance uses.

This function is a member of `VDPService_ChannelNotifySink`.

Method Signature

```
void (*v1.OnChannelStateChanged)(void *userData, VDPService_ChannelState currentState, VDPService_ChannelState transientState, void *reserved);
```

Parameters

Parameter	Description
userData	Parameter passed in to the <code>v1.RegisterChannelNotifySink</code> method. May be NULL.
currentState	The current state of the connection.
transientState	The state change that caused the callback. This can be different from <code>currentState</code> if other state changes have already taken place and are waiting to be processed.
reserved	Unused parameter.

Return Values

None

v1.OnConnectionStateChanged

This method is invoked when the connection in the underlying user session has changed its state.

This function is a member of `VDPService_ChannelNotifySink`.

Method Signature

```
void (*v1.OnConnectionStateChanged)(void *userData, VDPService_ConnectionState currentState, VDPService_ConnectionState transientState, void *reserved);
```

Parameters

Parameter	Description
userData	Parameter passed in to the v1.RegisterChannelNotifySink method. May be NULL.
currentState	The current state of the connection.
transientState	The state change that caused the callback. This can be different from currentState if other state changes have already taken place and are waiting to be processed.
reserved	Unused parameter.

Return Values

None

v1.OnPeerObjectCreated

This method is invoked when an object was created on the other side of the channel connection, and no object with the same name exists locally.

This function is a member of VDPService_ChannelNotifySink.

Method Signature

```
void (*v1.OnPeerObjectCreated)(void *userData, const char *objName, void *reserved);
```

Parameters

Parameter	Description
userData	Parameter passed in to the v1.RegisterChannelNotifySink method. May be NULL.
objName	The name of the object created by the peer.
reserved	Unused parameter.

Return Values

None

RPC Sinks

You must register RPC sinks to interact with and receive notifications of changes to RPC-specific Horizon Session Enhancement APIs.

v1.OnAbort

This method is called when the Invoke call that this sink is registered with fails due to a Horizon Session Enhancement error.

This function is a member of `VDPRPC_RequestCallback`.

Method Signature

```
void (*v1.OnAbort)(void *userData, uint32 contextId, Bool userCancelled, uint32 reason);
```

Parameters

Parameter	Description
userData	The userData parameter passed to the Invoke method.
contextId	ID of the context that was passed to the Invoke method.
userCancelled	FALSE.
reason	A <code>VDP_RPC_E_*</code> error code.

Return Values

None

v1.OnDone

This method is called when the Invoke method that this sink is registered with returns from the peer. The contextId parameter maps to the ID of the context that is passed to the Invoke call. This ID does not match the ID of the context that contextHandle points to. The contextHandle parameter holds all of the return codes and values given by the peer.

This function is a member of `VDPRPC_RequestCallback`.

Method Signature

```
void (*v1.OnDone)(void *userData, uint32 contextId, void *contextHandle);
```

Parameters

Parameter	Description
userData	The userData parameter passed to the Invoke method.
contextId	ID of the context that was passed to the Invoke method.
contextHandle	Handle for the context that holds all of the return data from the peer.

Return Values

None

v1.OnInvoke

This method is invoked when the peer on the other end of the channel calls Invoke. The contextHandle parameter is used to retrieve the data given by the peer, using

VDPService_ChannelContextInterface. This same context should be altered to hold the return values, and the context will be returned to the caller when this method returns.

This function is a member of VDRPC_ObjectNotifySink.

Method Signature

```
void (*v1.OnInvoke)(void *userData, void *contextHandle, void *reserved);
```

Parameters

Parameter	Description
userData	The userData parameter passed into the CreateChannelObject method. May be NULL.
contextHandle	Handle for the context that will contain the data for the call, and to hold the return values.
reserved	Unused parameter.

Return Values

None

v1.OnObjectStateChanged

Called when the state of the object this sink was registered with has changed.

This function is a member of VDRPC_ObjectNotifySink.

Method Signature

```
void (*v1.OnObjectStateChanged)(void *userData, void *reserved);
```

Parameters

Parameter	Description
userData	The userData parameter passed in to the CreateChannelObject method. May be NULL.
reserved	Unused parameter.

Return Values

None

Overlay Sinks

You must register overlay sinks to interact with and receive notifications of changes to overlay specific Horizon Session Enhancement APIs.

VDPOverlayGuest_Sink Functions

The following sections describe the VDPOverlayGuest_Sink functions.

v1.OnOverlayCreateError

This event handler is called when the client-side overlay is not created due to an error. Note that the window that is associated with the overlay is automatically unregistered.

This function is a member of VDPOverlayGuest_Sink.

Method Signature

```
void (*v1.OnOverlayCreateError)(void *userData, VDPOverlay_WindowId windowId, VDPOverlay_Error error);
```

Parameters

Parameter	Description
userData	The userData parameter passed in to the Init call.
windowId	The window ID that this callback corresponds to.
error	The error that was encountered.

Return Values

None

v1.OnOverlayReady

This event handler is called when the client-side overlay is ready to be displayed. It does not mean that the overlay is enabled or even that the client-side has loaded an image into the overlay. It means only that the overlay was properly created and is ready to display an image.

This function is a member of VDPOverlayGuest_Sink.

Method Signature

```
void (*v1.OnOverlayReady)(void *userData, VDPOverlay_WindowId windowId, uint32 response);
```

Parameters

Parameter	Description
userData	The userData parameter that was passed to the Init call.
windowId	The window ID that this callback corresponds to.
response	Client-side plug-in response.

Return Values

None

v1.OnOverlayRejected

This event handler is called when the client-side overlay is not created because the client-side plug-in rejected it. Note that the window that is associated with the overlay is automatically unregistered.

This function is a member of VDPOverlayGuest_Sink.

Method Signature

```
void (*v1.OnOverlayRejected)(void *userData, VDPOverlay_WindowId windowId, uint32 reason);
```

Parameters

Parameter	Description
userData	The userData parameter passed in to the Init call.
windowId	The window ID that this callback corresponds to.
reason	The client-side plug-in reason given for rejecting the overlay.

Return Values

None

v1.OnUserMsg (Guest Sink)

This event handler is called in response to a call to v1.SendMsg from the client.

This function is a member of VDPOverlayGuest_Sink.

Method Signature

```
void (*v1.OnUserMsg)(void *userData, VDPOverlay_WindowId windowId, void *msg, uint32 msgLen);
```

Parameters

Parameter	Description
userData	The userData parameter passed in to the Init call.

Parameter	Description
windowId	The window ID that this message is sent to, or VDP_OVERLAY_WINDOW_ID_NONE if the message was not sent to a particular window.
msg	The message data. Not valid after the call returns.
msgLen	Length of msg, in bytes.

Return Values

None

VDPOverlayClient_Sink Functions

The following sections describe the VDPOverlayClient_Sink functions.

v1.OnLayoutModeChanged

This event handler is called when the layout mode for the overlay is changed. This event handler is for information only. No action is required by the plug-in.

This function is a member of VDPOverlayClient_Sink.

Method Signature

```
void (*v1.OnLayoutModeChanged)(void *userData, VDPOverlayClient_ContextId contextId, VDPOverlay_WindowId windowId, VDPOverlay_LayoutMode layoutMode);
```

Parameters

Parameter	Description
userData	The userData parameter passed to the Init method.
contextId	The context ID returned from the Init call.
windowId	Window ID that corresponds to the referenced overlay.
layoutMode	The new layout mode.

Return Values

None

v1.OnOverlayDisabled

This event handler is called when the guest side deactivates the overlay using the DisableOverlay method, causing the current image in the overlay to be hidden. The overlay image data is maintained and will be re-displayed when the overlay is re-activated.

This function is a member of VDPOverlayClient_Sink.

Method Signature

```
void (*v1.OnOverlayDisabled)(void *userData, VDPOverlayClient_ContextId contextId, VDPOverlay_WindowId windowId, VDPOverlay_UserArgs userArgs);
```

Parameters

Parameter	Description
userData	The userData parameter passed to the Init method.
contextId	The context ID returned from the Init call.
windowId	Window ID that corresponds to the deactivated overlay.
userArgs	Value passed by the guest side to the DisableOverlay call.

Return Values

None

v1.OnOverlayEnabled

This event handler is called when the guest side activates the overlay using the EnableOverlay method. This event handler causes the current image in the overlay to be displayed.

This function is a member of VDPOverlayClient_Sink.

Method Signature

```
void (*v1.OnOverlayEnabled)(void *userData, VDPOverlayClient_ContextId contextId, VDPOverlay_WindowId windowId, VDPOverlay_UserArgs userArgs);
```

Parameters

Parameter	Description
userData	The userData parameter passed to the Init method.
contextId	The context ID returned from the Init call.
windowId	Window ID that corresponds to the activated overlay.
userArgs	Value passed by the guest side to the EnableOverlay call.

Return Values

None

v1.OnUserMsg (Client Sink)

This event handler is used when the guest-side application has called the SendMsg method.

This function is a member of VDPOverlayClient_Sink.

Method Signature

```
void (*v1.OnUserMsg)(void *userData, VDPOverlayClient_ContextId contextId, VDPOverlay_WindowId windowId, void *msg, uint32 msgLen);
```

Parameters

Parameter	Description
userData	The userData parameter passed in to the Init call.
contextId	The context ID returned from the Init call.
windowId	The window ID that this message is sent to, or VDP_OVERLAY_WINDOW_ID_NONE if the message was not sent to a particular window.
msg	The message data. Not valid after the call returns.
msgLen	Length of msg, in bytes.

Return Values

None

v1.OnWindowObscured

This event handler is called when the guest-side window that the overlay is tracking is completely obscured. The client-side can use this event as a hint to scale back drawing to the overlay.

This function is a member of VDPOverlayClient_Sink.

Method Signature

```
void (*v1.OnWindowObscured)(void *userData, VDPOverlayClient_ContextId contextId, VDPOverlay_WindowId windowId);
```

Parameters

Parameter	Description
userData	The userData parameter passed to the Init method.
contextId	The context ID returned from the Init call.
windowId	Window ID that corresponds to the obscured overlay.

Return Values

None

v1.OnWindowPositionChanged

This event handler is called when the guest-side window that the overlay is tracking changes position. The overlay is drawn at the new location. This event handler is for information only. No action is required by the plug-in.

This function is a member of VDPOverlayClient_Sink.

Method Signature

```
void (*v1.OnWindowPositionChanged)(void *userData, VDPOverlayClient_ContextId contextId, VDPOverlay_WindowId int32 x, int32 y);
```

Parameters

Parameter	Description
userData	The userData parameter passed to the Init method.
contextId	The context ID returned from the Init call.
windowId	Window ID that corresponds to the repositioned overlay.
x	New X position with the display.
y	New Y position with the display.

Return Values

None

v1.OnWindowRegistered

This event handler is called when the guest-side application registers a window using the RegisterWindow method. You can reject the request by setting reject to TRUE. Use the response parameter to return a reason to the guest. You can also use response to send a message to the guest in the non-rejected case.

Note Cache the windowId parameter because it is required to identify the overlay to the Overlay API.

This function is a member of VDPOverlayClient_Sink.

Method Signature

```
void (*v1.OnWindowRegistered)(void *userData, VDPOverlayClient_ContextId contextId, VDPOverlay_WindowId windowId, VDPOverlay_UserArgs userArgs, Bool *reject, uint32 *response);
```

Parameters

Parameter	Description
userData	The userData parameter passed in to the Init call.
contextId	The context ID returned from the Init call.
windowId	The window ID representing the new overlay.
userArgs	Value sent by the guest-side in the RegisterWindow call.
reject	Set to TRUE to deny the request to create an overlay.
response	Response sent back to the guest.

Return Values

None

v1.OnWindowSizeChanged

This event handler is called when the guest-side window that the overlay is tracking changes size. The old overlay image is redrawn according to the layout mode of the overlay. This event handler is for information only. No action is required by the plug-in.

This function is a member of VDPOverlayClient_Sink.

Method Signature

```
void (*v1.OnWindowSizeChanged)(void *userData, VDPOverlayClient_ContextId contextId, VDPOverlay_WindowId windowId, int32 width, int32 height);
```

Parameters

Parameter	Description
userData	The userData parameter passed to the Init method.
contextId	The context ID returned from the Init call.
windowId	Window ID that corresponds to the resized overlay.
width	New width of the window.
height	New height of the window.

Return Values

None

v1.OnWindowUnregistered

This event handler is called when the guest-side unregisters a window using the `UnregisterWindow` method. The window ID is no longer valid, and the overlay associated with the window ID is destroyed.

This function is a member of `VDPOverlayClient_Sink`.

Method Signature

```
void (*v1.OnWindowUnregistered)(void *userData, VDPOverlayClient_ContextId contextId, VDPOverlay_WindowId windowId, VDPOverlay_UserArgs userArgs);
```

Parameters

Parameter	Description
<code>userData</code>	The <code>userData</code> parameter passed to the <code>Init</code> method.
<code>contextId</code>	The context ID returned from the <code>Init</code> call.
<code>windowId</code>	Window ID for the window that was unregistered.
<code>userArgs</code>	Value sent by the guest-side application in the <code>UnregisterWindow</code> call.

Return Values

None

v1.OnWindowVisible

This event handler is called when the guest-side window that the overlay is tracking was obscured but now is at least partially visible.

This function is a member of `VDPOverlayClient_Sink`.

Method Signature

```
void (*v1.OnWindowVisible)(void *userData, VDPOverlayClient_ContextId contextId, VDPOverlay_WindowId windowId);
```

Parameters

Parameter	Description
<code>userData</code>	The <code>userData</code> parameter passed to the <code>Init</code> method.
<code>contextId</code>	The context ID returned from the <code>Init</code> call.
<code>windowId</code>	Window ID that corresponds to the overlay that is now partially visible.

Return Values

None

v3.OnLayerChanged

This event handler is called when the layer for the overlay is changed. This event handler is for information only. No action is required by the plugin.

This function is a member of VDPOverlayClient_Sink.

Method Signature

```
void (*v3.OnLayerChanged)(void* userData, VDPOverlayClient_ContextId contextId, VDPOverlay_WindowId windowId, uint32 layer);
```

Parameters

Parameter	Description
userData	The userData parameter passed to the Init method.
contextId	The context ID returned from the Init call.
windowId	The window ID.
layer	The new layer.

Return Values

None

v3.OnTopologyChanged

This event handler is called when the desktop topology of the Horizon client has changed. The desktopTopology array is only valid during the callback. This event handler is for information only.

No action is required by the plugin.

This function is a member of VDPOverlayClient_Sink.

Method Signature

```
void (*v3.OnTopologyChanged)(void* userData, VDPOverlayClient_ContextId contextId, const VDPOverlay_Rect* desktopBounds, int32 szDesktopTopology, const VDPOverlay_Rect* desktopTopology);
```

Parameters

Parameter	Description
userData	The userData parameter passed to the Init method.
contextId	The context ID returned from the Init call.
desktopBounds	The desktop bounding box.
szDesktopTopology	The size of the desktopTopology array.
desktopTopology	The desktop topology.

Return Values

None

Screen Capture Sinks

You must register screen capture sinks to interact with and receive notifications of changes to screen capture specific Horizon Session Enhancement APIs. This chapter describes the `VDPScreenCapture_Sink` functions.

v1.OnTopologyChanged

This event handler is called when the topology of the Horizon client has changed. This is for information only, no action is required by the plugin.

This function is a member of `VDPScreenCapture_Sink`.

Method Signature

```
void (*v1.OnTopologyChanged)(void* userData, VDPScreenCapture_ContextId contextId, uint32 flags);
```

Parameters

Parameter	Description
userData	The userData parameter passed in to <code>v1.Init()</code> .
contextId	The context ID returned from <code>v1.Init()</code> .
flags	Bit values that tell you what has changed. See the <code>VDP_SCREEN_CAPTURE_TOPOLOGY_CHANGED</code> flags for more information.

Return Values

None

v3.OnReadBackWindowReady

This event handler is called after `v3/v4.ReadBackWindowBegin()` is called to inform the application that the image for the window is now ready to be captured.

This function is a member of `VDPScreenCapture_Sink`.

Method Signature

```
void (*v3.OnReadBackWindowReady)(void* userData, VDPScreenCapture_ContextId contextId, VDPScreenCapture_ReadBackWindowHandle hReadBackWindow);
```

Parameters

Parameter	Description
userData	The userData parameter passed in to <code>v1.Init()</code> .

Parameter	Description
contextId	The context ID returned from v1.Init().
hReadBackWindow	The handle returned from v3/v4.ReadBackWindowBegin().

Return Values

None

v3.OnReadBackWindowDestroyed

This event handler is called if the remote desktop window being tracked is destroyed. It is not called when v3.ReadBackWindowEnd() is called.

This function is a member of VDPScreenCapture_Sink.

Method Signature

```
void (*v3.OnReadBackWindowDestroyed)(void* userData, VDPScreenCapture_ContextId contextId,
VDPScreenCapture_ReadBackWindowHandle hReadBackWindow);
```

Parameters

Parameter	Description
userData	The userData parameter passed in to v1.Init().
contextId	The context ID returned from v1.Init().
hReadBackWindow	The handle returned from v3/v4.ReadBackWindowBegin().

Return Values

None

v3.OnReadBackWindowRemoteError

This event handler is called if there is an error when setting up the window tracking usually due to an error returned by the Horizon Agent on the remote desktop. When this event handler is called, v3.OnReadBackWindowReady() will not be called. v3.ReadBackWindowEnd() must still be called to free up resources.

This function is a member of VDPScreenCapture_Sink.

Method Signature

```
void (*v3.OnReadBackWindowRemoteError)(void* userData, VDPScreenCapture_ContextId contextId,
VDPScreenCapture_ReadBackWindowHandle hReadBackWindow, VDPScreenCapture_RemoteError remoteError);
```

Parameters

Parameter	Description
userData	The userData parameter passed in to v1.Init().
contextId	The context ID returned from v1.Init().
hReadBackWindow	The handle returned from v3/v4.ReadBackWindowBegin().
remoteError	The reason the tracking failed. See the VDPScreenCapture_RemoteError values for more information.

Return Values

None

v4.OnReadBackWindowLocationChanged

This event handler is called if the remote desktop window being tracked is resized or moved. You must set the flag VDP_SCREEN_CAPTURE_READBACK_BEGIN_LOCATION_CHANGED/_EX in the call to v3/v4.ReadBackWindowBegin() to receive this event.

This function is a member of VDPScreenCapture_Sink.

Method Signature

```
void (*v4.OnReadBackWindowLocationChanged)(void* userData, VDPScreenCapture_ContextId contextId, VDPScreenCapture_ReadBackWindowHandle hReadBackWindow, const VMRect* pLocation);
```

Parameters

Parameter	Description
userData	The userData parameter passed in to v1.Init().
contextId	The context ID returned from v1.Init().
hReadBackWindow	The handle returned from v3/v4.ReadBackWindowBegin().
pLocation	The new location of the window.

Return Values

None

v4.OnReadBackWindowImageChanged

This event handler is called if the contents of the remote desktop window being tracked changes; i.e. the pixels in the window have changed. You must set the flag VDP_SCREEN_CAPTURE_READBACK_BEGIN_IMAGE_CHANGED in the call to v3.ReadBackWindowBegin()to receive this event.

This function is a member of VDPScreenCapture_Sink.

Method Signature

```
void (*v4.OnReadBackWindowImageChanged)(void* userData, VDPScreenCapture_ContextId contextId, VDPScreenCapture_ReadBackWindowHandle hReadBackWindow);
```

Parameters

Parameter	Description
userData	The userData parameter passed in to v1.Init().
contextId	The context ID returned from v1.Init().
hReadBackWindow	The handle returned from v3/v4.ReadBackWindowBegin().

Return Values

None

v4.OnReadBackRequest

This event handler is called when the Screen Capture API is requesting content to be included in a read back image.

This function is a member of VDPScreenCapture_Sink.

Method Signature

```
void (*v4.OnReadBackRequest)(void* userData, VDPScreenCapture_ContextId contextId, VDPScreenCapture_ReadBackRequestId readBackRequestId, VDPScreenCapture_HWND hWnd);
```

Parameters

Parameter	Description
userData	The userData parameter passed in to v1.Init().
contextId	The context ID returned from v1.Init().
readBackRequestId	The handle returned from v4.RegisterForReadBackRequests().
hWnd	The VDPScreenCapture_HWND passed to v4.RegisterForReadBackRequests().

Return Values

None