COMPLETEVIEW API

V7.3.244

SALIENT

Table of Contents

Introduction

The CompleteView APIs provide the means for third-party developers to write applications that administer and monitor CompleteView servers, and/or present streaming video (and audio) from such servers. There are two separate APIs for interfacing with CompleteView servers:

- The CVServerControl API, which consists of a set of COM interfaces for administering and monitoring CompleteView servers.
- The CVClientControl API, which consists of a set of COM interfaces for streaming video from CompleteView servers.

The CVServerControl API is a COM control that can be instantiated to establish a connection to a CompleteView server. The control presents an abstraction allowing programmatic third-party control over the server. The cameras administered by the server are accessible through this abstraction, but only to a limited degree. The primary focus is on the server itself.

The centerpiece of the CVClientControl API is an ActiveX control named CVVideo. An instance of the CVVideo control presents a (logical) connection to a CompleteView server-camera pair. The control provides an abstraction allowing programmatic third-party use and control of the camera. The other cameras administered by the server are accessible through this abstraction, but only to a limited degree.

There is a small amount of overlap in the functionalities exposed by these APIs (and their respective ActiveX controls). This overlap is more an accident of history than anything else.

Developer Audience

Developers using the CompleteView APIs must be generally familiar with the CompleteView family of products and their capabilities. They should also be familiar with COM interfaces in general and writing COM client software in particular.

The CompleteView APIs can be used by a variety of applications, from scripts embedded in HTML pages to full-fledged applications written in C++, C#, or Visual Basic.

The CompleteView API installer includes the COM APIs and client/server sample applications for 64-bit OS. The sample applications exemplify the use of the COM APIs to connect/monitor CompleteView servers.

Dependencies

CVClientControl and CVServerControl have runtime dependencies on Resources.Common.dll that is required for localization support. The libraries must be included within the language subfolders (.\en\Resources.Common.dll, .\es\Resources.Common.dll, and .\zh-CN\Resources.Common.dll) in the same directory as the CVClientControl and CVServerControl. In addition, CVClientControl has dependency on fisheye dewarping DLLs. Please refer to the sample applications for usage details. All the dependencies are included in the API installer.

CVServerControl API

The CVServerControl API provides a set of interfaces that allow applications to administer and monitor CompleteView servers. There are two primary interfaces:

- ICVServerControl, which represents an abstract connection to a CompleteView server. This interface is implemented by the CVServerControl and invoked by the application.
- ICVServerEvents, through which the CVServerControl may provide the application with asynchronous notifications of significant server events. This interface is (optionally) implemented by the application; it is invoked by the CVServerControl.

A secondary interface, ICVServerControl2, extends *ICVServerControl*. *ICVServerControl2* can be thought of as a newer version of *ICVServerControl*.

A third interface, ICVServerControl3, extends *ICVServerControl2*. *It adds new interface methods and functionality*.

A fourth interface, ICVServerControl4, extends ICVServerControl3. *It adds new methods and functionality.*

Using the CVServerControl API

Server Interaction Types

With respect to exchanging messages with the server, there is wide variety of behaviors exhibited by the methods in the CVServerControl API. The following table summarizes the server interaction types:

Server Interaction	Description
Always/Best-Effort	The method always conducts communication with the server (assuming a connection), but does not wait for the server's response. It is "best effort" in the sense that the client application is given no indication of the success or timing of the server's execution of the method.
Always/Synchronous	The method always conducts communication with the server (assuming a connection), and waits for the server's response.
Never	The method involves no communication with the server.
Never/Best-Effort	The method involves no communications with the server, and only provides the requested information if the server has already provided it.
Possibly/Synchronous	The method conducts communication with the server (and waits for the response), but only if the information requested has not already been received and cached locally.
Always/Asynchronous	The method always conducts communication with the server (assuming a connection), but does not wait for a response, instead creating an asynchronous handler to await server response when available.

Threading Model

CVServerControl is an apartment threaded COM server, and so each instance must be used on only a single thread, and that thread must have a Windows message loop.

Several methods in the CVServerControl API perform a synchronous message exchange with a CompleteView server. These methods are specially designed with an embedded Windows message loop and a visible (modal) dialog box, providing some visual user feedback and preventing "lockup" of the UI during lengthy operations. For many applications this usage pattern will be adequate. However, some application developers may want to consider putting the CVServerControl instance on a thread other than the main user interface thread. (As mentioned above, this second thread must have a message pump.)

CVServerControl API Reference

ICVServerControl

Application developers should not implement this interface. The CVServerControl provides this functionality.

Note: In C++, all properties are retrieved by a method that is named by prepending *get_* to the name of the property. For example, if the property's name is *Password*, then *get_Password* retrieves the property. Writable properties are set by a method that is named by prepending *put_* to the name of the property, e.g. *put_Password*.

The *ICVServerControl* interface inherits from the IDispatch interface, and has additional properties and methods, summarized in the following tables and subsequently described in detail.

Property	Description
Password	The user password when connecting to a server.
Server	Host name or IP address of the server.
UserName	The user name when connecting to a server.

Method	Description	Server Interaction
Connect	Establishes a connection to a server.	Always/Best-Effort
Disconnect	Tears down the current server connection.	Always/Synchronous
FlushCamera	Flushes the current video clip for a particular camera.	Always/Best-Effort
GetCameraInfo	Gets information about a camera.	Possibly/Synchronous
GetNumberOfCameras	Gets the number of cameras.	Possibly/Synchronous
GetNumberOfUsers	Gets the number of users.	Possibly/Synchronous
GetServerName	Gets the friendly name of the server.	Never/Best-Effort
GetServerVersion	Gets server version information.	Never/Best-Effort
GetStorageInfo	Gets server storage information.	Always/Synchronous
GetUserName	Gets a particular user name.	Possibly/Synchronous
GotoPreset	Moves a particular camera to a preset.	Always/Best-Effort
IsServerConnected	Indicates whether currently connected to a server.	Never
SetCameraOverlay	Sets the overlay characteristics of a particular camera.	Always/Best-Effort
SetServerTime	Sets the server's system time.	Always/Best-Effort
StartAlarm	Starts alarm recording with a particular camera.	Always/Best-Effort
StartAlarmReason	Starts alarm recording with a particular camera (with a textual description/justification).	Always/Best-Effort
StartMotion	Starts motion recording with a particular camera.	Always/Best-Effort
StartMotionReason	Starts motion recording with a particular camera (with a textual description/justification).	Always/Best-Effort
StopAlarm	Stops alarm recording with a particular camera.	Always/Best-Effort
StopMotion	Stops motion recording with a particular camera.	Always/Best-Effort
TimedAlarm	Starts alarm recording with a particular camera for a particular duration.	Always/Best-Effort
TimedAlarmReason	Starts alarm recording with a particular camera for a particular duration (with a textual description/justification).	Always/Best-Effort
TimedMotion	Starts motion recording with a particular camera for a particular duration.	Always/Best-Effort
TimedMotionReason	Starts motion recording with a particular camera for a particular duration (with a textual description/justification).	Always/Best-Effort

ICVServerControl::Connect

The *Connect* method starts the process of establishing a connection to the server identified by the Server property, using the credentials represented by the UserName and Password properties.

Syntax (C++)

```
HRESULT Connect(
);
```

Parameters

The *Connect* method has no parameters.

Return Value

The return value is *S_OK*.

Remarks

The connection is established asynchronously. The application may poll for the connection status by calling the IsServerConnected method.

If the CVServerControl was previously connected to a server, that connection is torn down, followed by initiation of a new connection.

ICVServerControl::Disconnect

The *Disconnect* method takes down the current connection to the server.

Syntax (C++)

```
HRESULT Disconnect(
);
```

Parameters

The *Disconnect* method has no parameters.

Return Value

The return value is *S_OK*.

Remarks

Disconnect takes down the current connection to the server, if one has been established. If a connection has been initiated (via Connect) but not completed, then that connection attempt is aborted.

ICVServerControl::FlushCamera

The *FlushCamera* method instructs the server to flush the current video clip and start recording a new video clip, for a particular camera.

Syntax (C++)

```
HRESULT FlushCamera(
    [in]  SHORT Camera
);
```

Parameters

Camera [in]

Camera identifier (one-based index).

Return Value

The return value is S_OK if the appropriate command is successfully sent to the server, S_FAIL if no server is connected or a problem is detected with the connection.

Remarks

FlushCamera is a best-effort method, providing no indication of whether or when the server executes the command.

ICVServerControl::GetCameraInfo

The *GetCameraInfo* method returns information about a particular camera.

Syntax (C++)

```
HRESULT GetCameraInfo(
   [in]  SHORT CameraID,
   [out] BSTR *CameraName
   [out] UINT *CameraStatus
);
```

Parameters

CameraID [in]

Camera identifier (one-based index).

CameraName [out]

Receives the camera name, or is unset. See the remarks below.

CameraStatus [out]

Receives the camera status vector, or is unset. See the remarks below.

Return Value

The return value is *S_OK*.

Remarks

The camera name and status, if reported by this method, are not necessarily current. (One might reasonably expect the camera name to remain static, but the status is more likely to vary over time.) To guarantee up-to-date information, the client application should first call the PurgeCache method.

The camera status is a bit field with the following status bits:

Bit Position	Meaning when Set
0 (least significant)	Scheduled recording is active.
1	External alarm detected, external alarm recording is active.
2	Motion recording is active.
3	Motion has been detected.
4	Camera is a PTZ camera.
5	Prealarm recording is active.
6	Not connected to server (video acquisition failure).

7	Tour is enabled.
8	Camera is inactive/stopped (no attempt by server to acquire video), though it may be enabled.
9	Camera supports audio.
10	Camera is an IP camera.
11	Camera is an ACTI model.
13	Camera is transitioning to or from the stopped state. Bit 8 is the intended next state (0 = started, 1 = stopped) but the camera is not there yet.
14	Camera is set to use Digital PTZ.
15	Recording failure.
16	Camera preset and/or preset name changed.

The *CameraName* and *Status* output parameters will be unset if there is no connection to the server, or a problem is experienced with the connection, or if the camera identifier is invalid.

GetCameraInfo may perform a synchronous message exchange with the currently connected server. During this exchange Windows messages will continue to be dispatched.

ICVServerControl::GetNumberOfCameras

The *GetNumberOfCameras* method returns the number of cameras with which the currently connected server is configured.

Syntax (C++)

```
HRESULT GetNumberOfCameras(
   [out] SHORT *Count
);
```

Parameters

Count [out]

Receives the camera count, or is unset. See the remarks below.

Return Value

The return value is *S_OK*.

Remarks

The camera count, if any is reported by this method, is not necessarily current. If up-to-date information is required, the client application should first call the PurgeCache method.

The *Count* output parameter will be unset if there is no connection to a server, or a problem is experienced with the connection.

GetNumberOfCameras may perform a synchronous message exchange with the currently connected server. During this exchange Windows messages will continue to be dispatched.

ICVServerControl::GetNumberOfUsers

The *GetNumberOfUsers* method returns the number of users with which the currently connected server is configured.

Syntax (C++)

```
HRESULT GetNumberOfUsers(
   [out] SHORT *Count
);
```

Parameters

Count [out]

Receives the user count, or is unset. See the remarks below.

Return Value

The return value is *S_OK*.

Remarks

The user count, if any is reported by this method, is not necessarily current. If up-to-date information is required, the client application should first call the PurgeCache method.

The *Count* output parameter will be unset if there is no connection to a server, or a problem is experienced with the connection.

GetNumberOfUsers may perform a synchronous message exchange with the currently connected server. During this exchange Windows messages will continue to be dispatched.

ICVServerControl::GetServerName

The *GetServerName* method returns the friendly name of the currently connected server.

Syntax (C++)

```
HRESULT GetServerName(
   [out] BSTR *ServerName
);
```

Parameters

ServerName [out]

Receives the friendly name of the currently connected server, or is set to the empty string. See the remarks below.

Return Value

The return value is *S_OK*.

Remarks

GetServerName provides the friendly server name if the CVServerControl is connected to a server and that server has provided its friendly name. Otherwise the returned string is empty.

ICVServerControl::GetServerVersion

The *GetServerVersion* method retrieves version information about the currently connected server.

Syntax (C++)

```
HRESULT GetServerVersion(
   [out] LONG *Major,
   [out] LONG *Minor,
   [out] LONG *Release,
```

```
   [out] LONG *Build
);
```

Parameters

Major [out]

Receives the major version number, or is left unset. See the remarks below.

Minor [out]

Receives the minor version number, or is left unset. See the remarks below.

Release [out]

Receives the release number, or is left unset. See the remarks below.

Build [out]

Receives the build number, or is left unset. See the remarks below.

Return Value

The return value is *S_OK* if version information is returned, *S_FALSE* otherwise.

Remarks

GetServerVersion provides version information if the CVServerControl is connected to a server and the server has provided its version information.

ICVServerControl::GetStorageInfo

The *GetStorageInfo* method retrieves up-to-date storage information from the currently connected server.

Syntax (C++)

```
HRESULT GetStorageInfo(
   [out] LONG *UsedSpace,
   [out] LONG *AvailableSpace
);
```

Parameters

UsedSpace [out]

Receives the used space in megabytes, summed over all volumes on the server, or is set to zero if an error occurs.

AvailableSpace [out]

Receives the available space in megabytes, summed over all volumes on the server, or is set to zero if an error occurs.

Return Value

The return value is *S_OK* if the storage information is successfully retrieved from the server, *S_FALSE* otherwise.

Remarks

GetStorageInfo normally conducts a synchronous message exchange with the currently connected server. During this exchange, Windows messages will continue to be dispatched.

ICVServerControl::GetUserName

The *GetUserName* method retrieves a particular user name from the currently connected server.

Syntax (C++)

```
HRESULT GetUserName(
   [in]  SHORT UserID,
   [out] BSTR *UserName
);
```

Parameters

UserID [in]

User identifier (one-based index).

UserName [out]

Receives the user name, or is set to the empty string. See the remarks below.

Return Value

The return value is *S_OK*.

Remarks

The user name, if any is reported by this method, is not necessarily current. To ensure up-to-date information, the client application should first call the PurgeCache method.

The *UserName* output parameter will be set to the empty string if there is no connection to a server, a problem occurs with the connection, or the user identifier is invalid.

GetUserName may perform a synchronous message exchange with the currently connected server. During this exchange Windows messages will continue to be dispatched.

ICVServerControl::GotoPreset

The *GotoPreset* method instructs the currently connected server to direct a particular camera to a particular preset location.

Syntax (C++)

```
HRESULT GotoPreset(
   [in]  SHORT CameraID,
   [in]  SHORT Preset
);
```

Parameters

CameraID [in]

Camera identifier (one-based index).

Preset [in]

Camera preset number.

Return Value

The return value is *S_OK* if the command is successfully sent to the server, *S_FALSE* if not connected to a server or a problem occurs with the connection.

Remarks

GotoPreset only works for mechanical PTZ cameras and not on cameras using digital PTZ driver. This function is a best-effort method providing no indication of whether or when the server actually executes the command.

ICVServerControl::IsServerConnected

The *IsServerConnected* method returns an indication of whether the CVServerControl is connected to a server.

Syntax (C++)

```
HRESULT IsServerConnected(
   [out] VARIANT_BOOL *Connected
);
```

Parameters

Connected [out]

Receives a Boolean indicating whether the CVServerControl is connected to a server.

Return Value

The return value is *S_OK*.

ICVServerControl::Password

The *Password* property determines the user password that is used on the next connection to a server. Changing this property has no effect on the current connection (if any).

Syntax (C++)

```
HRESULT get_Password(
   [out] BSTR *pVal
);
HRESULT put_Password(
   [in]  BSTR newVal
);
```

Parameters

pVal [out]

Receives the current value of the property.

newVal [in]

Password to use in the next connection to a server.

Return Value

The return value is *S_OK*.

ICVServerControl::Server

The *Server* property determines the identity of the server to which connection is attempted when the Connect method is called. This property should be a host name, or an IP address in dotted-quad format, or the string *localhost*. Changing this property has no effect on the current connection (if any).

Syntax (C++)

```
HRESULT get_Server(
   [out] BSTR *pVal
```

```
);
HRESULT put_Server(
   [in]  BSTR newVal
);
```

Parameters

pVal [out]

Receives the current value of the Server property.

newVal [in]

Identifies the next server to which a connection should be attempted (on call of Connect).

Return Value

The return value is *S_OK*.

ICVServerControl::SetCameraOverlay

The *SetCameraOverlay* method instructs the currently connected sever to change the overlay parameters for a particular camera.

Syntax (C++)

```
HRESULT SetCameraOverlay(
   [in]  SHORT CameraID,
   [in]  BSTR OverlayText,
   [in]  VARIANT_BOOL ShowTimeStamp,
   [in]  VARIANT_BOOL ShowCameraName
);
```

Parameters

CameraID [in]

Camera identifier (one-based index).

OverlayText [in]

Overlay text. A null value or an empty string clears the overlay text.

ShowTimeStamp [in]

A Boolean that specifies whether to include timestamps in the overlay.

ShowCameraName [in]

A Boolean that specifies whether to include the camera name in the overlay.

Return Value

The return value is *S_OK* if the command was successfully sent to the server, *S_FALSE* if not connected to a server or a problem occurs with the connection.

Remarks

SetCameraOverlay is a best-effort method providing no indication of whether or when the server actually executes the command.

ICVServerControl::SetServerTime

The *SetServerTime* method instructs the currently connected server to change its system time.

Syntax (C++)

```
HRESULT SetServerTime(
   [in]  DATE Time
   [in]  VARIANT_BOOL UTC
);
```

Parameters

Time [in]

The target server date/time.

UTC [in]

A Boolean specifying whether the *Time* parameter is in UTC.

Return Value

The return value is *S_OK* if the command was successfully sent to the server, *S_FALSE* if not connected to a server or a problem occurred with the connection.

Remarks

SetServerTime is a best-effort method providing no indication of whether or when the server actually executes the command.

ICVServerControl::StartAlarm

The *StartAlarm* method instructs the currently connected server to begin alarm recording with a specific camera.

Syntax (C++)

```
HRESULT StartAlarm(
   [in]  SHORT CameraID
);
```

Parameters

CameraID [in]

Camera identifier (one-based index).

Return Value

The return value is *S_OK*.

Remarks

StartAlarm is a best-effort method providing no indication of whether or when the server actually executes the command.

The StartAlarmReason method performs the same function as *StartAlarm* but allows the client application to attach a textual description/justification to the recording event.

ICVServerControl::StartAlarmReason

The *StartAlarmReason* method instructs the currently connected server to begin alarm recording with a specific camera. Unlike the StartAlarm method, it allows the client application to provide a textual description/justification for the recording.

Syntax (C++)

```
HRESULT StartAlarmReason(
   [in]  SHORT CameraID,
   [in]  BSTR Reason
);
```

Parameters

CameraID [in]

Camera identifier (one-based index).

Reason [in]

Description/justification for the recording.

Return Value

The return value is *S_OK*.

Remarks

StartAlarmReason is a best-effort method providing no indication of whether or when the server actually executes the command.

ICVServerControl::StartMotion

The *StartMotion* method instructs the currently connected server to begin motion recording with a specific camera.

Syntax (C++)

```
HRESULT StartMotion(
   [in]  SHORT CameraID
);
```

Parameters

CameraID [in]

Camera identifier (one-based index).

Return Value

The return value is *S_OK*.

Remarks

StartMotion is a best-effort method providing no indication of whether or when the server actually executes the command.

The StartMotionReason method performs the same function as *StartMotion* but allows the client application to attach a textual description/justification to the recording event.

ICVServerControl::StartMotionReason

The *StartMotionReason* method instructs the currently connected server to begin motion recording with a specific camera. Unlike the StartMotion method, the client application can provide a textual description/justification for the recording.

Syntax (C++)

```
HRESULT StartMotionReason(
   [in]  SHORT CameraID,
   [in]  BSTR Reason
);
```

Parameters

CameraID [in]

Camera identifier (one-based index).

Reason [in]

Description/justification for the recording.

Return Value

The return value is *S_OK*.

Remarks

StartMotionReason is a best-effort method providing no indication of whether or when the server actually executes the command.

ICVServerControl::StopAlarm

The *StopAlarm* method instructs the server to stop alarm recording with a particular camera.

Syntax (C++)

```
HRESULT StopAlarm(
   [in]  SHORT CameraID
);
```

Parameters

CameraID [in]

Camera identifier (one-based index).

Return Value

The return value is *S_OK*.

Remarks

StopAlarm is a best-effort method providing no indication of whether or when the server actually executes the command.

ICVServerControl::StopMotion

The *StopMotion* method instructs the server to stop motion recording with a particular camera.

Syntax (C++)

```
HRESULT StopMotion(
   [in]  SHORT CameraID
);
```

Parameters

CameraID [in]

Camera identifier (one-based index).

Return Value

The return value is *S_OK*.

Remarks

StopMotion is a best-effort method providing no indication of whether or when the server actually executes the command.

ICVServerControl::TimedAlarm

The *TimedAlarm* method instructs the server to begin alarm recording with a specific duration.

Syntax (C++)

```
HRESULT TimedAlarm(
   [in]  SHORT CameraID
   [in]  SHORT Seconds
);
```

Parameters

CameraID [in]

Camera identifier (one-based index).

Seconds [in]

Length of time to record (in seconds). This value overrides the value configured on the server for alarm recording. The value is clamped to a minimum of one second.

Return Value

The return value is *S_OK*.

Remarks

TimedAlarm is a best-effort method providing no indication of whether or when the server actually executes the command.

The TimedAlarmReason method provides the same functionality, but also allows the client application to attach a textual description/justification to the recording.

ICVServerControl::TimedAlarmReason

The *TimedAlarmReason* method instructs the currently connected server to begin alarm recording with a specific duration. Unlike the TimedAlarm method, it allows the client application to specify a textual description/justification for the recording.

Syntax (C++)

```
HRESULT TimedAlarmReason(
   [in]  SHORT CameraID,
   [in]  SHORT Seconds,
   [in]  BSTR Reason
);
```

Parameters

CameraID [in]

Camera identifier (one-based index).

Seconds [in]

Length of time to record (in seconds). This value overrides the value configured on the server for alarm recording. The value is clamped to a minimum of one second.

Reason [in]

Description/justification for the recording.

Return Value

The return value is *S_OK*.

Remarks

TimedAlarmReason is a best-effort method providing no indication of whether or when the server actually executes the command.

ICVServerControl::TimedMotion

The *TimedMotion* method instructs the server to begin motion recording with a specific duration.

Syntax (C++)

```
HRESULT TimedMotion(
   [in]  SHORT CameraID
   [in]  SHORT Seconds
);
```

Parameters

CameraID [in]

Camera identifier (one-based index).

Seconds [in]

Length of time to record (in seconds). This value overrides the value configured on the server for motion recording. The value is clamped to a minimum of one second.

Return Value

The return value is *S_OK*.

Remarks

TimedMotion is a best-effort method providing no indication of whether or when the server actually executes the command.

The TimedMotionReason method provides the same functionality, but also allows the client application to attach a textual description/justification to the recording.

ICVServerControl::TimedMotionReason

The *TimedMotionReason* method instructs the currently connected server to begin motion recording with a specific duration. Unlike the TimedMotion method, it allows the client application to specify a textual description/justification for the recording.

Syntax (C++)

```
HRESULT TimedMotionReason(
   [in]  SHORT CameraID,
   [in]  SHORT Seconds,
   [in]  BSTR Reason
);
```

Parameters

CameraID [in]

Camera identifier (one-based index).

Seconds [in]

Length of time to record (in seconds). This value overrides the value configured on the server for motion recording. The value is clamped to a minimum of one second.

Reason [in]

Description/justification to attach to the recording.

Return Value

The return value is *S_OK*.

Remarks

TimedMotionReason is a best-effort method providing no indication of whether or when the server actually executes the command.

ICVServerControl::UserName

The *UserName* property determines the user name that is used on the next connection to a server. Changing this property has no effect on the current connection (if any).

Syntax (C++)

```
HRESULT get_UserName(
   [out] BSTR *pVal
);
HRESULT put_UserName(
   [in]  BSTR newVal
);
```

Parameters

pVal [out]

Receives the current value of the property.

newVal [in]

User name to use in the next connection to a server.

Return Value

The return value is *S_OK*.

ICVServerControl2

The *ICVServerControl2* interface inherits from the ICVServerControl interface. *ICVServerControl2* offers additional methods, summarized in the following table and described in detail in the text that follows.

Method	Description	Server Interaction
GetCameraPermissions	Gets the access permissions for a particular (user, camera) pair.	Possibly/Synchronous
PurgeCache	Clears the local cache of camera and user information.	Never
SearchVideo	Gets information about the time intervals for which video exists.	Always/Synchronous
TestCameraPermissions	Tests a camera permissions bitmap for a particular permissions bit.	Never
SetClientName	Sets client's name for the underlying TCP connection	Always/Best-Effort
SetClientVersion	Sets client's version for the underlying TCP connection	Always/Best-Effort

ICVServerControl2::GetCameraPermissions

The *GetCameraPermissions* method retrieves the access permissions for a particular user and camera.

Syntax (C++)

```
HRESULT GetCameraPermissions(
   [in]  SHORT UserID,
   [in]  SHORT CameraID,
   [out] LONG *Permissions
);
```

Parameters

UserID [in]

User identifier (one-based index).

CameraID [in]

Camera identifier (one-based index).

Permissions [out]

Receives the permissions bitmap for the (user, camera) pair. This bitmap is a bitwise-OR combination of the permissions flags enumerated by CameraPermission.

Return Value

The return value is *S_OK*.

Remarks

The client application may test the individual bits of the permissions bitmap through the usual language-specific means, or may use the TestCameraPermissions method.

GetCameraPermissions may conduct a synchronous message exchange with the currently connected server. During this exchange, Windows messages will continue to be dispatched.

ICVServerControl2::PurgeCache

The *PurgeCache* method clears the CVServerControl's locally cached information about cameras and users.

Syntax (C++)

```
HRESULT PurgeCache(
);
```

Parameters

The *PurgeCache* method has no parameters.

Return Value

The return value is *S_OK*.

ICVServerControl2::SearchVideo

The *SearchVideo* method returns an indication of when video exists from particular cameras during a particular time interval.

Syntax (C++)

```
HRESULT SearchVideo(
    [in]  VARIANT Cameras,
    [in]  DATE Start,
    [in]  DATE End,
    [out] VARIANT *ClipStart,
    [out] VARIANT *ClipEnd,
    [out] VARIANT_BOOL *Result
);
```

Parameters

Cameras [in]

A safe array of *SHORT* (*VT_ARRAY* | *VT_I2*), or a single *SHORT* (*VT_I2*), designating the camera(s) to include in the search. An empty array instructs the server to include all cameras in the search.

Start [in]

A date/time value giving the beginning of the temporal range to search, expressed in client local time.

End [in]

A data/time value giving the end of the temporal range to search, expressed in client local time.

ClipStart [out]

Receives a safe array of *DATE* (*VT_ARRAY* | *VT_DATE*). This array contains the start timestamps for each "clip" found by the search. The *i*th element of the array contains the start time of a clip, whose end time is the *i*th element of *ClipEnd*.

ClipEnd [out]

Receives a safe array of *DATE* (*VT_ARRAY* | *VT_DATE*). This array contains the end timestamps for each "clip" found by the search. The *i*th element of the array contains the end time of the clip whose start time is the *i*th element of *ClipStart*.

Result [out]

Receives a Boolean indicating whether the search was successful. Reasons for failure include not being connected to a server, or a problem with the connection. Note that a search returning no clips (i.e. returning empty *ClipStart* and *ClipEnd* arrays) may still have been a successful search.

Return Value

The return value is *E_INVALIDARG* if the input Cameras variant is of the wrong type. Otherwise the return value is *S_OK*.

Remarks

SearchVideo returns time intervals indicating the presence of video. More precisely, it returns a sequence of *n* time intervals $[s_1, t_1]$, $[s_2, t_2]$, … $[s_n, t_n]$ such that

$$s_i \leq t_i \text{ for each } i \in \{1, \dots n\}, \text{ i.e. each interval is non-empty;}$$

$$t_i < s_{i+1} \text{ for each } i \in \{1, \dots, n-1\}, \text{ i.e. the intervals are sorted and have non-empty spacing;}$$

$$s_1 \geq Start \text{ and } t_n \leq End; \text{ i.e. the results are limited to the specified range;}$$

$$\text{There exists video from } \textit{at least one of} \text{ the designated cameras for all } t \in \bigcup_{i=1}^{n} [s_i, t_i]; \text{ and}$$

$$\text{If } t \in [Start, End] \cap \sim \bigcup_{i=1}^{n} [s_i, t_i] \text{, then none of the designated cameras have video for time } t.$$

All times returned via *ClipStart* and *ClipEnd* are in client local time.

SearchVideo normally performs a synchronous message exchange with the server. During this exchange Windows messages will continue to be dispatched.

ICVServerControl2::TestCameraPermissions

The *TestCameraPermissions* method tests a camera permissions bitmap for a particular type of access permission.

Syntax (C++)

```
HRESULT TestCameraPermissions(
   [in]  LONG Permissions,
   [in]  CameraPermission Permission,
   [out] VARIANT_BOOL *Authorized
);
```

Parameters

Permissions [in]

A camera permissions bitmap returned by GetCameraPermissions.

Permission [in]

The specific permission to test.

Authorized [out]

Receives a Boolean indicating whether the specific permission is set in the permissions bitmap.

Return Value
The return value is *S_OK*.

Remarks
TestCameraPermissions is the recommended way to test a permissions bitmap in client applications that do not have programmatic access to the CameraPermission enumeration.

ICVServerControl2::SetClientName
The *SetClientName* method sets the client for the underlying TCP connection. Setting this value helps identify client connections originate from CVServerControl.

Syntax (C++)
```
HRESULT SetClientName(
   [in]  BSTR ClientName
);
```

Parameters
ClientName [in]
String identifying the client connecting through CVServerControl interface.

Return Value
The return value is *S_OK*.

ICVServerControl2::SetClientVersion
The *SetClientVersion* method sets the version information of the client for the underlying TCP connection.

Syntax (C++)
```
HRESULT SetClientVersion(
   [in]  BSTR ClientVersion
);
```

Parameters
ClientVersion [in]
String containing the version of the client.

Return Value
The return value is *S_OK*.

ICVServerControl2::SetTimeout
The *SetTimeout* method sets timeout value used when attempting to connect to a given host.

Syntax (C++)
```
HRESULT SetTimeout(
   [in]  UINT64 Timeout
);
```

Parameters

Timeout [in]

Timeout in milliseconds to be used on a connection attempt.

Return Value

The return value is *S_OK*.

Remarks

Typical timeout values could range from 10 to 60 seconds. If not set, connections will timeout based on Windows default socket timeout setting.

ICVServerControl3

The *ICVServerControl3* interface inherits from the ICVServerControl2 interface. *ICVServerControl3* offers additional properties and methods, summarized in the following tables and described in detail in the text that follows.

Property	Description
FlushOldEvents	Set to true to receive all events logged since the server started. Set to false to receive events logged only from when the connection to server has been established. By default, this property is set to false.
HeartbeatInterval	Recording service heartbeat interval.

Method	Description	Server Interaction
SearchVideoWithInfo	Retrieves video clip information based on camera indexes.	Possibly/Synchronous
GetServerTimeZone	Retrieves the server time zone standard name.	Always/Synchronous
GetCameraTimeZone	Retrieves the camera time zone standard name.	Always/Synchronous
AddSmartSearchZone	Adds a smart search zone for given coordinates.	Never
StartSmartSearch	Begins the smart search action.	Always/Asynchronous
StopSmartSearch	Stops the smart search action.	Always/Best-Effort
SaveSnapshot	Saves a snapshot to a file.	Always/Synchronous
GetCameraDetails	Retrieves the camera details for specified camera index.	Always/Synchronous
GetSnapshot	Retrieves a live snapshot with the supplied width and height.	Always/Synchronous
EnableServerInfo	Allows the client to setup ongoing request for the supplied server information.	Always/Best-Effort
DisableServerInfo	Allows the client to disable a request for the supplied server information.	Always/Best-Effort
GetSnapshotsAsync	Retrieves snapshots for the requested list of cameras with the supplied width, height and quality.	Always/Asynchronous
StopSnapshotsAsynch	Ends the asynchronous snapshot request if there is one occurring.	Always/Best-Effort
GetServerPlatform	Retrieves the OS platform for the currently connected server.	Always/Synchronous
GetEventDaysAsync	Request a list of the days in the event log that contain events	Always/Asynchronous
GetVideoDaysAsync	Request a list of the days that contain video.	Always/Asynchronous
GetLogEventsAsync	Request a list of log events for the specified time period.	Always/Asynchronous
StopLogEventsAsync	Stops a currently started request for log events.	Always/Best-Effort
DeleteVideoFilesAsync	Delete video files for a given camera, volume and date combination.	Always/Asynchronous
StartRecordingServiceAsync	Starts the video recording service of the connected server.	Always/Asynchronous
StopRecordingServiceAsync	Stops the video recording service of the connected server.	Always/Asynchronous
RestartRecordingServiceAsync	Restarts the video recording service of the connected server.	Always/Asynchronous
FilterCameraEventsByType	Attempts to set subscription filtering by event type preference.	Always/Best-Effort

GetNumberOfIoDevices	Gets the number of IO devices that have been configured on the server.	Always/Synchronous
GetIoDetailByGuid	Gets information on an IO device identified using the device's GUID.	Always/Synchronous
GetIoDetailByIndex	Gets information on an IO device identified using the device's index.	Always/Synchronous
TriggerOutputByGuid	Trigger an output on an IO device, using the GUID to identify the IO device.	Always/Best-Effort
TriggerOutputByIndex	Trigger an output on an IO device, using the device index to identify it.	Always/Best-Effort
GetFeatureKeys	Get information about feature keys available in server's license.	Always/Synchronous
ExportToServerQueue	Pushes a video clip to the Recording Server export queue.	Always/Synchronous
StartServiceListener	Starts the broadcast Recording Server service listener.	Always/Synchronous
StopServiceListener	Stops the broadcast Recording Server service listener.	Always/Synchronous
ResetRecordingServiceAsync	Resets a Recording Server to the default configuration.	Always/Asynchronous
SecureConnect	Connects to a server via Secure TCP connection. By default, the underlying class will use "localhost", "admin" and "" for Server, UserName and Password properties if they were not set.	Always/Synchronous

ICVServerControl3:: SearchVideoWithInfo

The *SearchVideoWithInfo* method retrieves video clip information based on the camera indexes supplied. This is a synchronous operation.

Syntax (C++)

```
HRESULT SearchVideoWithInfo (
   [in] VARIANT Cameras,
   [in] DATE Start,
   [in] DATE End,
   [in] BSTR TimeZoneStandardName,
   [in] RecordingCategory Category,
   [in] VARIANT Volumes,
   [out] VARIANT* ResultCameras,
   [out] VARIANT* ResultCatergories,
   [out] VARIANT* ResultStartDateTimes,
   [out] VARIANT* ResultEndDateTimes,
   [out] VARIANT* ResultStartActDateTimes,
   [out] VARIANT* ResultEndActDateTimes,
   [out] VARIANT* ResultDescriptions,
   [out] VARIANT* ResultClipSizes,
   [out] VARIANT* Reserved1,
   [out] VARIANT* Reserved2,
   [out] VARIANT* Reserved3,
   [out, retval] VARIANT_BOOL *Result
```

```
);
```

Parameters

Cameras [in]

List of camera indexes for video search. Empty list indicates all cameras

Start [in]

Search start time.

End [in]

Search end time.

TimeZoneStandardName [in]

Time zone standard name for search operation (see Remarks).

Category [in]

Recording category to filter search results. See RecordingCategory enumeration.

Volumes [in]

List of volumes for video search.

ResultCameras [out]

Result list of camera indexes pertaining to video clips found.

ResultCategories [out]

Result list of video category pertaining to video clips found.

ResultStartDateTimes [out]

Result list of start times pertaining to result video clips found.

ResultEndDateTimes [out]

Result list of end times pertaining to result video clips found.

ResultStartActDateTimes [out]

Result list of actual start times pertaining to result video clips found.

ResultEndActDateTimes [out]

Result list of actual end times pertaining to result video clips found.

ResultDescriptions [out]

Result list of descriptions pertaining to result video clips found.

ResultClipSizes [out]

Result list of clip sizes pertaining to result video clips found.

Result [out]

Receives a result code indicating the success or failure of the operation. A value of zero indicates success. Any other value is an internal implementation-defined error code and indicates failure.

Return Value
The return value is *S_OK*.

Remarks
It is strongly recommended to use a language-invariant time zone name, rather than a localized time zone name, for the *TimeZoneStandardName* input value.

ICVServerControl3:: GetServerTimeZone
The *GetServerTimeZone* method retrieves the server time zone standard name for the currently connected server. This is a synchronous operation.

Syntax (C++)
```
HRESULT GetServerTimeZone(
   [out] LONG *OffsetSeconds,
   [out] BSTR *Name,
   [out, retval] VARIANT_BOOL *Result
);
```

Parameters
OffsetSeconds [out]
Server time zone offset (from UTC), in seconds.

Name [out]
Server time zone standard name (see Remarks). e.g. Pacific Standard Time

Result [out]
Receives a result code indicating the success or failure of the operation. A value of zero indicates success. Any other value is an internal implementation-defined error code and indicates failure.

Return Value
The return value is *S_OK*.

Remarks
The value of the *Name* output will be in the Windows display language in use on the Recording Server system. When the Recording Server runs as a Windows Service, the language for the SYSTEM account is used.

ICVServerControl3:: GetCameraTimeZone
The *GetCameraTimeZone* method retrieves the camera time zone standard name for the requested camera. This is a synchronous operation.

Syntax (C++)
```
HRESULT GetCameraTimeZone (
   [in] INT Camera,
   [out] BSTR *Name,
   [out, retval] VARIANT_BOOL *Result
);
```

Parameters

Camera [in]

Camera identifier (one-based index).

Name [out]

Camera time zone standard name (see Remarks). e.g. Pacific Standard Time

Result [out]

Receives a result code indicating the success or failure of the operation. A value of zero indicates success. Any other value is an internal implementation-defined error code and indicates failure.

Return Value

The return value is *S_OK*.

Remarks

The value of the *Name* output may be localized, depending on various Windows system settings in effect when the camera was configured.

ICVServerControl3:: AddSmartSearchZone

The *AddSmartSearchZone* method adds a smart search zone with the supplied user parameters. This is a synchronous operation.

Syntax (C++)

```
HRESULT AddSmartSearchZone (
    [in] INT Left,
    [in] INT Top,
    [in] INT Right,
    [in] INT Bottom,
    [in] INT Sensitivity
);
```

Parameters

Left [in]

X-coordinate of the upper-left corner of a rectangle

Top [in]

Y-coordinate of the upper-left corner of a rectangle

Right [in]

X-coordinate of the lower-right corner of a rectangle

Bottom [in]

Y-coordinate of the lower-left corner of a rectangle

Sensitivity [in]

Rectangular region and sensitivity value: 0 (low) <= sensitivity <= 100 (high)

Result [out]

Receives a result code indicating the success or failure of the operation. A value of zero indicates success. Any other value is an internal implementation-defined error code and indicates failure.

Return Value

The return value is *S_OK*.

ICVServerControl3:: StartSmartSearch

The *StartSmartSearch* method begins the smart search action with the supplied user parameters. This is an asynchronous operation. The callback method handlers to monitor the progress of the search are OnSmartSearchResult and OnSmartSearchProgress.

Syntax (C++)

```
HRESULT StartSmartSearch (
   [in] INT Camera,
   [in] DATE Start,
   [in] DATE End,
   [in] BSTR TimeZoneStandardName,
   [out, retval] VARIANT_BOOL *Result
);
```

Parameters

Camera [in]

Camera identifier (one-based index).

Start [in]

Query start date and time in specified time zone.

End [in]

Query end date and time in specified time zone.

TimeZoneStandardName [in]

Camera time zone standard name (see Remarks). e.g. Pacific Standard Time

Result [out]

Receives a result code indicating the success or failure of the operation. A value of zero indicates success. Any other value is an internal implementation-defined error code and indicates failure.

Return Value

The return value is *S_OK*.

Remarks

It is strongly recommended to use a language-invariant time zone name, rather than a localized time zone name, for the *TimeZoneStandardName* input value.

ICVServerControl3:: StopSmartSearch

The *StopSmartSearch* method stops the smart search action. This is a synchronous operation. There is no indication sent from the server as to the ultimate success of failure of this action.

Syntax (C++)

```
HRESULT StopSmartSearch (
);
```

Parameters

The *StopSmartSearch* method has no parameters.

Return Value

The return value is *S_OK*.

ICVServerControl3:: SaveSnapshot

The *SaveSnapshot* method saves a snapshot to a file with the supplied user parameters. This is a synchronous operation. If width and height are not provided then the value of the original frame width and height will be utilized.

Syntax (C++)

```
HRESULT SaveSnapshot (
   [in] BSTR Filename,
   [in] INT Camera,
   [in, defaultvalue(-1)] INT Width,
   [in, defaultvalue(-1)] INT Height,
   [out, retval] VARIANT_BOOL *Result
);
```

Parameters

Filename [in]

File to which the snapshot needs to be saved.

Camera [in]

Camera identifier (one-based index).

Width [in]

Snapshot width.

Height [in]

Snapshot height.

Result [out]

Receives a result code indicating the success or failure of the operation. A value of zero indicates success. Any other value is an internal implementation-defined error code and indicates failure.

On failure the resultant codes are outlined in the following table:

Failure Code	Description
E_INVALIDARG	One or more arguments are invalid
E_ACCESSDENIED	General access denied error
E_ABORT	Operation aborted
E_FAIL	Unspecified error

Return Value

The return value is *S_OK*.

ICVServerControl3:: GetCameraDetails

The *GetCameraDetails* method retrieves the camera details for specified camera index. This is a synchronous operation.

Syntax (C++)

```
HRESULT GetCameraDetails (
   [in] INT Camera,
   [ref, in, out] CameraDetails* CameraDetails,
   [out] BSTR* Error,
   [out, retval] VARIANT_BOOL *Result
);
```

Parameters

Camera [in]

Camera identifier (one-based index).

CameraDetails [in/out]

Pointer to camera details object, see CameraDetails. This must be passed in by API client.

Error [out]

Description of error if any error is returned from the Result.

Result [out]

Receives a result code indicating the success or failure of the operation. A value of zero indicates success. Any other value is an internal implementation-defined error code and indicates failure.

Return Value

The return value is *S_OK*.

ICVServerControl3:: GetSnapshot

The *GetSnapshot* method retrieves a live snapshot with the supplied width and height. This is a synchronous operation. If width and height are not provided then the value of the original frame width and height will be utilized.

Syntax (C++)

```
HRESULT GetSnapshot (
   [in] INT Camera,
   [out] VARIANT* Buffer,
   [out] BSTR* Error,
   [in, defaultvalue(-1)] INT Width,
   [in, defaultvalue(-1)] INT Height,
   [out, retval] VARIANT_BOOL *Result
);
```

Parameters

Camera [in]

Camera identifier (one-based index).

Buffer [out]

Buffer containing the requested snapshot.

Error [out]

Description of error if any error is returned from the Result.

Width [in]

Snapshot width.

Height [in]

Snapshot height.

Result [out]

Receives a result code indicating the success or failure of the operation. A value of zero indicates success. Any other value is an internal implementation-defined error code and indicates failure.

Return Value

The return value is *S_OK*.

ICVServerControl3:: EnableServerInfo

The *EnableServerInfo* method allows the client to setup ongoing requests for the supplied server information. This is an asynchronous operation. The method handlers for the requested server information are OnSystemStatsUpdate, OnVolumeInfoUpdate, OnLicenseInfoUpdate, OnCameraInfoUpdate, OnConnectionInfoUpdate and OnStoragePoolInfoUpdate.

Syntax (C++)

```
HRESULT EnableServerInfo (
   [in]enum ServerInfoType InfoType,
   [in]UINT PollIntervalMs);
);
```

Parameters

InfoType [in]

ServerInfoType for which to poll the server.

PollIntervalMs [in]

Interval in milliseconds with which to poll the server for information.

Return Value

The return value is *S_OK*.

ICVServerControl3:: DisableServerInfo

The *DisableServerInfo* method allows the client to disable a request for the supplied server information.

Syntax (C++)

```
HRESULT DisableServerInfo (
   [in] enum ServerInfoType InfoType
);
```

Parameters

InfoType [in]

ServerInfoType for which to disable polling to the server.

Return Value

The return value is *S_OK*.

ICVServerControl3:: GetSnapshotsAsync

The *GetSnapshotsAsync* method retrieves snapshots for the requested list of cameras with the supplied width, height and quality. This is an asynchronous operation. The method handler to identify retrieval completion is OnSnapshotResult. If width and height are not provided then the value of the original frame width and height will be utilized.

Syntax (C++)

```
HRESULT GetSnapshotsAsync (
   [in] VARIANT Cameras,
   [in, out] BSTR* Error,
   [in, defaultvalue(-1)] INT Width,
   [in, defaultvalue(-1)] INT Height,
   [in, defaultvalue(100)]   INT Quality,
   [out, retval] VARIANT_BOOL *Result
);
```

Parameters

Cameras [in]

List of camera indexes for snapshot retrieval. Empty list indicates all cameras

Error [out]

Description of error if any error is returned from the Result.

Width [in]

Snapshot width.

Height [in]

Snapshot height.

Quality [in]

Snapshot quality. The range is 0 (worst quality) to 100 (best quality).

Result [out]

Receives a result code indicating the success or failure of the operation. A value of zero indicates success. Any other value is an internal implementation-defined error code and indicates failure.

Return Value

The return value is *S_OK*.

ICVServerControl3:: StopSnapshotsAsync

The *StopSnapshotsAsync* method ends the asynchronous snapshot request if there is one occurring. This is a synchronous operation.

Syntax (C++)

```
HRESULT StopSnapshotsAsync (
);
```

Parameters

The *StopSnapshotsAsync* method has no parameters.

Return Value

The return value is *S_OK*.

ICVServerControl3:: GetServerPlatform

The *GetServerPlatform* method retrieves the OS platform for the currently connected server. This is a synchronous operation.

Syntax (C++)

```
HRESULT GetServerPlatform (
   [out] BSTR* Platform
);
```

Parameters

Platform [out]

Value indicating the platform of the server. i.e. “x64” or “x32”

Return Value

The return value is *S_OK*.

ICVServerControl3:: GetEventDaysAsync

The *GetEventDaysAsync* method requests a list of the days in the event log that contain events. This is an asynchronous operation. The method handler to identify retrieval completion is OnEventDaysUpdate.

Syntax (C++)

```
HRESULT GetEventDaysAsync (
   [out] BSTR* Error,
   [out, retval] VARIANT_BOOL *Result
);
```

Parameters

Error [out]

Description of error if any error is returned from the Result.

Result [out]

Receives a result code indicating the success or failure of the operation. A value of zero indicates success. Any other value is an internal implementation-defined error code and indicates failure.

Return Value

The return value is *S_OK*.

ICVServerControl3:: GetVideoDaysAsync

The *GetVideoDaysAsync* method requests a list of the days that contain video. This is an asynchronous operation. The method handler to identify retrieval completion is OnVideoDaysUpdate.

Syntax (C++)

```
HRESULT GetVideoDaysAsync (
   [in]VARIANT Cameras,
   [in]VARIANT Volumes,
   [out] BSTR* Error,
   [out, retval] VARIANT_BOOL *Result
);
```

Parameters

Cameras [in]

List of camera indexes for video day retrieval. Empty list indicates all cameras.

Volumes [in]

List of volume indexes for video day retrieval.

Error [out]

Description of error if any error is returned from the Result.

Result [out]

Receives a result code indicating the success or failure of the operation. A value of zero indicates success. Any other value is an internal implementation-defined error code and indicates failure.

Return Value

The return value is *S_OK*.

ICVServerControl3:: GetLogEventsAsync

The GetLogEventsAsync method requests a list of log events for the specified time period. This method requires that at least one event type is requested. This is an asynchronous operation. The method handler to identify retrieval completion is OnLogEventsResult.

Syntax (C++)

```
HRESULT GetLogEventsAsync (
   [in] DATE Start,
   [in] DATE End,
   [in] BSTR TimeZoneStandardName,
   [in] UINT MaxResults,
   [in] VARIANT_BOOL IncludeServerEvents,
   [in] VARIANT EventTypes,
   [out] BSTR* Error,
   [out, retval] VARIANT_BOOL *Result
);
```

Parameters

Start [in]

Start time.

End [in]

End time.

TimeZoneStandardName [in]

String representing the standard timezone to use for this request (see Remarks). e.g. Pacific Standard Time. If no time zone is supplied it will be assumed that the "Local" time zone is requested.

MaxResults [in]

Specifies the maximum number of events to return. A value of -1 specifies that there is no limit.

IncludeServerEvents [in]

Set to true to include server events.

EventTypes [in]

List of events to include. See EventType enumeration definition for a list of supported types.

Error [out]

Description of error if any error is returned from the Result.

Result [out]

Receives a result code indicating the success or failure of the operation. A value of zero indicates success. Any other value is an internal implementation-defined error code and indicates failure.

Return Value

The return value is *S_OK*.

Remarks

It is strongly recommended to use a language-invariant time zone name, rather than a localized time zone name, for the *TimeZoneStandardName* input value.

ICVServerControl3:: StopLogEventsAsync

The *StopLogEventsAsync* method stops a running request for log events. This is a synchronous operation.

Syntax (C++)

```
HRESULT StopLogEventsAsync (
);
```

Parameters

The *StopLogEventsAsync* method has no parameters.

Return Value

The return value is *S_OK*.

ICVServerControl3:: DeleteVideoFilesAsync

The *DeleteVideoFilesAsync* method deletes video files for a given camera, volume and date combination. This is an asynchronous operation. The method handler to identify completion is OnDeleteVideoFilesResult.

Syntax (C++)

```
HRESULT DeleteVideoFilesAsync (
   [in] INT CameraId,
   [in] INT VolumeId,
   [in] VARIANT Date,
   [out] BSTR* Error,
   [out, retval] VARIANT_BOOL *Result
);
```

Parameters

CameraId [in]

Camera identifier (one-based index). A value of -1 indicates all cameras.

VolumeId [in]

Volume identifier (one-based index). A value of -1 indicates all volumes.

Date [in]

Represented by a string of the type mm-dd-yyyy.

Error [out]

Description of error if any error is returned from the Result.

Result [out]

Receives a result code indicating the success or failure of the operation. A value of zero indicates success. Any other value is an internal implementation-defined error code and indicates failure.

Return Value

The return value is *S_OK*.

ICVServerControl3:: StartRecordingServiceAsync

The *StartRecordingServiceAsync* starts the video recording service of the connected server. This is an asynchronous operation. The callback method handler to identify completion is OnAdminServiceTaskResult.

Syntax (C++)

```
HRESULT StartRecordingServiceAsync (
   [out] BSTR* Error,
   [out, retval] VARIANT_BOOL *Result
);
```

Parameters

Error [out]

Description of error if any error is returned from the Result.

Result [out]

Receives a result code indicating the success or failure of the operation. A value of zero indicates success. Any other value is an internal implementation-defined error code and indicates failure.

Return Value

The return value is *S_OK*.

ICVServerControl3:: StopRecordingServiceAsync

The *StopRecordingServiceAsync* stops the video recording service of the connected server. This is an asynchronous operation. The method handler to identify completion is OnAdminServiceTaskResult.

Syntax (C++)

```
HRESULT StopRecordingServiceAsync (
   [out] BSTR* Error,
   [out, retval] VARIANT_BOOL *Result
);
```

Parameters

Error [out]

Description of error if any error is returned from the Result.

Result [out]

Receives a result code indicating the success or failure of the operation. A value of zero indicates success. Any other value is an internal implementation-defined error code and indicates failure.

Return Value

The return value is *S_OK*.

ICVServerControl3:: RestartRecordingServiceAsync

The Restart*RecordingServiceAsync* restarts the video recording service of the connected server. This is an asynchronous operation. The method handler to identify completion is OnAdminServiceTaskResult.

Syntax (C++)

```
HRESULT RestartRecordingServiceAsync (
   [out] BSTR* Error,
   [out, retval] VARIANT_BOOL *Result
);
```

Parameters

Error [out]

Description of error if any error is returned from the Result.

Result [out]

Receives a result code indicating the success or failure of the operation. A value of zero indicates success. Any other value is an internal implementation-defined error code and indicates failure.

Return Value

The return value is *S_OK*.

ICVServerControl3:: FilterCameraEventsByType

The FilterCameraEventsByType accepts a list of EventTypes and attempts to set the camera event subscription based on the requested types. If no filter is set after initial connection or anytime thereafter, then all events are in the subscription. If an empty list is sent then the filter is reset to subscribe to all events.

Syntax (C++)

```
HRESULT FilterCameraEventsByType (
```

```
    [in] eventTypes,
    [out] BSTR* error,
    [out, retval] VARIANT_BOOL *result
);
```

Parameters

eventTypes [in]

List of event types. See EventTypes enumeration.

error [out]

Description of error if any error is returned from the Result.

result [out]

Receives a result code indicating the success or failure of the operation. A value of zero indicates success. Any other value is an internal implementation-defined error code and indicates failure.

Return Value

The return value is *S_OK*.

ICVServerControl3:: GetIoDetailByGuid

The *GetIoDetailByGuid* method retrieves I/O details for the specified I/O using the device's GUID to uniquely identify it. This is a synchronous operation.

Syntax (C++)

```
HRESULT GetIoDetailByGuid(
[in] GUID deviceGuid,
[ref, in, out] IoDetails* ioDetails,
[out] BSTR* error,
[out, retval] VARIANT_BOOL* result
);
```

Parameters

deviceGuid [in]

I/O device identifier.

ioDetails [in/out]

Pointer to I/O details object, see IoDetails. This must be passed in by API client.

Error [out]

Description of error if any error is returned from the Result.

Result [out]

Receives a result code indicating the success or failure of the operation. A value of zero indicates success. Any other value is an internal implementation-defined error code and indicates failure.

Return Value

The return value is *S_OK*.

ICVServerControl3:: GetIoDetailByIndex

The *GetIoDetailByIndex* method retrieves the I/O details for specified I/O device using the index to identify the device. This is a synchronous operation.

Syntax (C++)

```
HRESULT GetIoDetailByIndex(
[in] INT ioIndex,
[ref, in, out] IoDetails* ioDetails,
[out] BSTR* error,
[out, retval] VARIANT_BOOL* result
);
```

Parameters

ioIndex [in]

I/O device identifier.

ioDetails [in/out]

Pointer to I/O details object, see IoDetails. This must be passed in by API client.

Error [out]

Description of error if any error is returned from the Result.

Result [out]

Receives a result code indicating the success or failure of the operation. A value of zero indicates success. Any other value is an internal implementation-defined error code and indicates failure.

Return Value

The return value is *S_OK*.

ICVServerControl3:: TriggerOutputByGuid

The *TriggerOutputByGuid* method triggers or resets an output on an I/O device using the device's GUID to uniquely identify it. This is an asynchronous operation. If set is "true", the output specified will be triggered. If it is "false", the output specified will be reset. Whether an output should be closed when triggered, or open when triggered, is specified in the configuration and will be determined and processed by the server on receiving the command.

Syntax (C++)

```
HRESULT TriggerOutputByGuid (
[in] GUID deviceGuid,
[in] INT output,
VARIANT_BOOL set,
[out] BSTR* error);
```

Parameters

deviceGuid [in]

I/O device identifier.

output [in]

Output number to be triggered.

set [in]

True to set the output, and false to reset.

Error [out]

Description of error if any error is returned from the Result.

Return Value

The return value is *S_OK*.

ICVServerControl3:: TriggerOutputByIndex

The *TriggerOutputByIndex* method triggers or resets an output on an I/O device using the device's index to uniquely identify it. This is an asynchronous operation. If set is "true", the output specified will be triggered. If it is "false", the output specified will be reset. Whether an output should be closed when triggered, or open when triggered, is specified in the configuration and will be determined and processed by the server on receiving the command.

Syntax (C++)

```
HRESULT TriggerOutputByIndex (
[in] INT deviceNumber,
[in] INT output,
VARIANT_BOOL set,
[out] BSTR* error);
```

Parameters

deviceNumber [in]

I/O device identifier.

output [in]

Output number to be triggered.

set [in]

True to set the output, and false to reset.

Error [out]

Description of error if any error is returned from the Result.

Return Value

The return value is *S_OK*.

ICVServerControl3:: GetNumberOfIoDevices

The *GetNumberOfIoDevices* retrieves the number of I/O devices configured on the server. This is a synchronous operation.

Syntax (C++)

```
HRESULT GetNumberOfIoDevices (
[out] INT* count,
 BSTR *error
);
```

Parameters

count [out]

Number of I/O devices configured.

Error [out]

Description of error if any error is returned from the Result.

Return Value

The return value is *S_OK*.

ICVServerControl3:: GetFeatureKeys

The *GetFeatureKeys* method gets information about all available feature keys in the server license. This is a synchronous operation.

Syntax (C++)

```
HRESULT GetFeatureKeys (
[out] VARIANT* keys,
[out] VARIANT* descriptions,
[out] VARIANT* quantities,
[out] BSTR* error,
[out, retval] VARIANT_BOOL* result);
```

Parameters

keys [out]

List of feature keys as string.

descriptions [out]

List of descriptions of each feature key as string.

quantities [out]

List of quantity/feature information associated with each feature key as string.

error [out]

Description of error if any error is returned from the result.

Result [out]

Boolean value indicating success or failure.

Return Value

The return value is *S_OK*.

ICVServerControl3:: FlushOldEvents

The *FlushOldEvents* property can be set to true to enable receiving notifications of all events logged from the time the server started. By default, this property is set to false and the client application is notified of events logged only from the time connection to the server has been established. This property should be set before establishing connection to the server with the Connect call. Changing this property has no effect on the current connection.

Syntax (C++)

```
HRESULT get_FlushOldEvents(
[out] VARIANT_BOOL* result);

HRESULT put_FlushOldEvents (
[in] VARIANT_BOOL enabled);
```

Parameters

result [out]

Receives the current value of the property.

enabled [in]

Specifies whether to flush events from server start (VARIANT_TRUE) or not (VARIANT_FALSE).

Return Value

The return value is *S_OK*.

ICVServerControl3:: ExportToServerQueue

The *ExportToServerQueue* method pushes a video clip to the Recording Server export queue. The Recording Server must have an export volume configured. Video clips are queued on the Recording Server until the configured export time, at which a process is triggered which begins writing the clips to the export volume.

This method is synchronous. It returns the result of pushing the video clip to the export queue. It does not return any errors related to writing the video clip to the export volume.

Syntax (C++)

```
HRESULT ExportToServerQueue (
[in] QueueExportClipInfo clipInfo,
[out] BSTR* error,
[out, retval] VARIANT_BOOL* result);
```

Parameters

clipInfo [in]

Video clip settings. See QueueExportClipInfo.

error [out]

Description of error if any error is returned from the result.

result [out]

Boolean value indicating success or failure.

Return Value

The return value is *S_OK*.

ICVServerControl3::StartServiceListener

The *StartServiceListener* method starts a broadcast listener to listen for Recording Server service started events. The service endpoint port to listen on must match the service endpoint port configured on the Recording Server in order to receive these broadcast messages.

This method is synchronous.

Syntax (C++)

```
HRESULT StartServiceListener (
[in] Int endpoint,
[out] BSTR* error,
[out, retval] VARIANT_BOOL* result);
```

Parameters

endpoint [in]

Endpoint port to listen on.

error [out]

Description of error if any error is returned from the result.

result [out]

Boolean value indicating success or failure.

Return Value

The return value is *S_OK*.

ICVServerControl3::StopServiceListener

The *StopServiceListener* method stops the broadcast listener if it exists and has been started.

This method is asynchronous.

Syntax (C++)

```
HRESULT StopServiceListener (
[out] BSTR* error,
[out, retval] VARIANT_BOOL* result);
```

Parameters

error [out]

Description of error if any error is returned from the result.

result [out]

Boolean value indicating success or failure.

Return Value

The return value is *S_OK*.

ICVServerControl3::HeartbeatInterval

The HeartbeatInterval property. This value is configured on the Recording Server. The default value is 30 seconds. If this value is set to zero then no heartbeat will be sent out by the Recording Server.

Syntax (C++)

```
HRESULT get_HeartbeatInterval (
   [out] INT *heartbeatInterval
);

HRESULT put_HeartbeatInterval (
   [in] INT heartbeatInterval
```

```
);
```

ICVServerControl3::ResetRecordingServiceAsync

The *ResetRecordingServiceAsync* method resets the connected Recording Server to a clean install configuration. This method signals to the AdminService to stop the Recording Server, backup all required configuration files, clean those files and then start the Recording Server.

This method is asynchronous.

Syntax (C++)

```
HRESULT ResetRecordingServiceAsync (
[out] BSTR* error,
[out, retval] VARIANT_BOOL* result);
```

Parameters

error [out]

Description of error if any error is returned from the result.

result [out]

Boolean value indicating success or failure.

Return Value

The return value is S_OK.

ICVServerControl3::SecureConnect

The *SecureCOnnect* methods connects to a given server via a secure TCP connection. By default, the underlying class will use "localhost", "admin" and "" for Server, UserName and
Password properties if they were not set.
This method is synchronous.

Syntax (C++)

```
HRESULT SecureConnect (void);
```

Parameters

None..

Return Value

The return value is *S_OK*.

ICVServerControl4

The *ICVServerControl4* interface inherits from the ICVServerControl3 interface. *ICVServerControl4* offers additional properties and methods, summarized in the following tables and described in detail in the text that follows.

Property	Description
TimeZoneKeyName	Gets the time zone identifier. On a Windows machine, this corresponds to the registry key identifying the time zone on the server system.

Method	Description	Server Interaction
EnableEventSubscription	Allows for events subscription to be enabled or disabled. By default, event subscription is enabled. If connected this will also subscribe or unsubscribe from events.	Possibly/Asynchronous

ICVServerControl4::EnableEventSubscription

The *EnableEventSubscription* methods when connected will subscribe to events if enabled is "true" and unsubscribe from events if enabled is "false". When the event subscription has been disabled *Connect*, *SecureConnect*, and *FilterCameraEventsByType* to no longer subscribe to events. By default, event subscription is enabled.

This method is asynchronous when connected.

Syntax (C++)

```
HRESULT EnableEventSubscription (
[in] VARIANT_BOOL enabled);
```

Parameters

enabled [in]

Specifiec if the events subscription should be enabled.

Return Value

The return value is *S_OK*.

ICVServerControl4::TimeZoneKeyName

Read this property to get the name of the time zone in use on the server system when the connection to the server was established. This value corresponds to the Windows registry key identifying the time zone.

Syntax (C++)

```
HRESULT get_TimeZoneKeyName (
   [out] BSTR *timeZoneKeyName
);
```

Parameters

timeZoneKeyName [out]

Receives the time zone name

Return Value

The return value is S_OK upon success.

ICVServerControl5

The *ICVServerControl5* interface inherits from the ICVServerControl4 interface. *ICVServerControl5* offers additional properties and methods, summarized in the following tables and described in detail in the text that follows.

Method	Description	Server Interaction
SetAdminServiceLanguage	Allows for language to be updated on the Admin Service. By default, the language will be the system language if supported. If selected language is unavailable English will be used.	Possibly/Asynchronous

ICVServerControl5::SetAdminServiceLanguage

The *SetAdminServiceLanguage* method sets the current language for the Admin Service. When the Admin Service is started the default language will use the current system language. If language is not supported English will be used.

This method is asynchronous when connected.

Syntax (C++)

```
HRESULT SetAdminServiceLanguage (
[in] BSTR localeName, [out] BSTR* error);
```

Parameters

localeName [in]

Specifiec if the locale that the Admin Service should use.

error [out]

String describing any error that may be reported.

Return Value

The return value is *S_OK*.

ICVServerControl6

The *ICVServerControl6* interface inherits from the ICVServerControl5 interface. *ICVServerControl6* offers an additional method, summarized in the following table and described in detail in the text that follows.

Method	Description	Server Interaction
GetServerInfo	Retrieves a brief overview of the server information	Always/Synchronous
SaveStreamSnapshot	Saves a snapshot to a file using the supplied camera ID and stream ID.	Always/Synchronous
GetStreamSnapshot	Retrieves a live snapshot using the supplied camera ID and stream ID.	Always/Synchronous
GetStreamSnapshotsAsync	Retrieves snapshots for the requested list of cameras with the supplied stream ID, width, height and quality.	Always/Asynchronous

ICVServerControl6::GetServerInfo

The *GetServerInfo* method retrieves a brief overview of the server information. This is a synchronous operation.

Syntax (C++)

```
HRESULT GetServerInfo (
   [ref, in, out] ServerInfo* serverInfo,
   [out] BSTR* error,
   [out, retval] VARIANT_BOOL* result
);
```

Parameters

serverInfo [in/out]

Pointer to a server information object, see ServerInfo. This must be passed in by API client.

error [out]

Description of error if any error is returned from the Result.

result [out]

Receives a result code indicating the success or failure of the operation. A value of zero indicates success. Any other value is an internal implementation-defined error code and indicates failure.

Return Value

The return value is *S_OK* if the server information is successfully retrieved from the server. The return value can also be *E_POINTER* if a connection to the Recording Server does not exist or *S_FALSE* if another error occurs.

ICVServerControl6::SaveStreamSnapshot

The *SaveStreamSnapshot* method saves a snapshot to a file with the supplied user parameters. This is a synchronous operation. If width and height are not provided then the value of the original frame width and height will be utilized.

To save a snapshot to a file using an automatically selected stream ICVServerControl3::SaveSnapshot should be used.

Syntax (C++)

```
HRESULT SaveStreamSnapshot(
   [in] BSTR Filename,
   [in] INT Camera,
   [in] INT streamId,
   [in, defaultvalue(-1)] INT Width,
   [in, defaultvalue(-1)] INT Height,
   [out, retval] VARIANT_BOOL *Result
);
```

Parameters

Filename [in]

File to which the snapshot needs to be saved.

Camera [in]

Camera identifier (one-based index).

streamId [in]

Stream identifier (one-based index). Must be greater than or equal to 1.

Width [in]

Snapshot width.

Height [in]

Snapshot height.

Result [out]

Receives a result code indicating the success or failure of the operation. A value of zero indicates success. Any other value is an internal implementation-defined error code and indicates failure.

On failure the resultant codes are outlined in the following table:

Failure Code	Description
E_INVALIDARG	One or more arguments are invalid
E_ACCESSDENIED	General access denied error
E_ABORT	Operation aborted
E_FAIL	Unspecified error

Return Value

The return value is *S_OK*.

ICVServerControl6::GetStreamSnapshot

The *GetStreamSnapshot* method retrieves a live snapshot with the supplied camera ID, stream ID, width, and height. This is a synchronous operation. If width and height are not provided then the value of the original frame width and height will be utilized.

To retrieve a snapshot using an automatically selected stream ICVServerControl3::GetSnapshot should be used.

Syntax (C++)

```
HRESULT GetStreamSnapshot(
   [in] INT Camera,
   [in] INT streamId,
   [out] VARIANT *Buffer,
   [out] BSTR *Error,
   [in, defaultvalue(-1)] INT Width,
   [in, defaultvalue(-1)] INT Height,
   [out, retval] VARIANT_BOOL *Result
);
```

Parameters

Camera [in]

Camera identifier (one-based index).

streamId [in]

Stream identifier (one-based index). Must be greater than or equal to 1.

Buffer [out]

Buffer containing the requested snapshot.

Error [out]

Description of error if any error is returned from the Result.

Width [in]

Snapshot width.

Height [in]

Snapshot height.

Result [out]

Receives a result code indicating the success or failure of the operation. A value of zero indicates success. Any other value is an internal implementation-defined error code and indicates failure.

Return Value

The return value is *S_OK*.

ICVServerControl6::GetStreamSnapshotsAsync

The *GetStreamSnapshotsAsync* method retrieves snapshots for the requested list of cameras with the supplied camera ID, stream ID, width, height, and quality. This is an asynchronous operation. The method handler to identify retrieval completion is OnSnapshotResult. If width and height are not provided then the value of the original frame width and height will be utilized.

To asynchronously retrieve snapshots using an automatically selected stream ICVServerControl3::GetSnapshotsAsync should be used.

To stop the asynchronous snapshot request use ICVServerControl3::StopSnapshotsAsync.

Syntax (C++)

```
HRESULT GetStreamSnapshotsAsync (
   [in] VARIANT Cameras,
   [in] INT streamId,
   [in, out] BSTR *Error,
   [in, defaultvalue(-1)] INT Width,
   [in, defaultvalue(-1)] INT Height,
   [in, defaultvalue(100)] INT Quality,
   [out, retval] VARIANT_BOOL *Result
);
```

Parameters

Cameras [in]

List of camera indexes for snapshot retrieval. Empty list indicates all cameras.

streamId [in]

The stream identifier to use for the provided list of camera indexes (one-based). Must be greater than or equal to 1.

Error [out]

Description of error if any error is returned from the Result.

Width [in]

Snapshot width.

Height [in]

Snapshot height.

Quality [in]

Snapshot quality. The range is 0 (worst quality) to 100 (best quality).

Result [out]

Receives a result code indicating the success or failure of the operation. A value of zero indicates success. Any other value is an internal implementation-defined error code and indicates failure.

Return Value

The return value is *S_OK*.

ICVServerControl6::SaveStreamSnapshot

The *SaveStreamSnapshot* method saves a snapshot to a file with the supplied user parameters. This is a synchronous operation. If width and height are not provided then the value of the original frame width and height will be utilized.

To save a snapshot to a file using an automatically selected stream ICVServerControl3::SaveSnapshot should be used.

Syntax (C++)

```
HRESULT SaveStreamSnapshot(
   [in] BSTR Filename,
   [in] INT Camera,
   [in] INT streamId,
   [in, defaultvalue(-1)] INT Width,
   [in, defaultvalue(-1)] INT Height,
   [out, retval] VARIANT_BOOL *Result
);
```

Parameters

Filename [in]

File to which the snapshot needs to be saved.

Camera [in]

Camera identifier (one-based index).

streamId [in]

Stream identifier (one-based index). Must be greater than or equal to 1.

Width [in]

Snapshot width.

Height [in]

Snapshot height.

Result [out]

Receives a result code indicating the success or failure of the operation. A value of zero indicates success. Any other value is an internal implementation-defined error code and indicates failure.

On failure the resultant codes are outlined in the following table:

Failure Code	Description
E_INVALIDARG	One or more arguments are invalid
E_ACCESSDENIED	General access denied error
E_ABORT	Operation aborted
E_FAIL	Unspecified error

Return Value

The return value is *S_OK*.

ICVServerControl7

The *ICVServerControl7* interface inherits from the ICVServerControl6 interface. *ICVServerControl7* offers additional methods, summarized in the following table and described in detail in the text that follows.

Method	Description	Server Interaction
GetTrialLicenseInfo	Gets trial license keys and uses	Always/Synchronous
GetServerInfo2	Retrieves a brief overview of the server information.	Always/Synchronous

ICVServerControl7::GetTrialLicenseInfo

The *GetTrialLicenseInfo* method retrieves info about the keys and uses of features. This is a synchronous operation.

Syntax (C++)

```
HRESULT GetTrialLicenseInfo (
   [out] VARIANT* Types,
   [out] VARIANT* Keys,
   [out] VARIANT* KeysUsed
);
```

Parameters

Types [out]

Array of enumFeature defining what feature the matching index of Keys and KeysUsed describes.

Keys [out]

Array of ints defining the number of keys for the Recording Server.

KeysUsed [out]

Array of ints defining the number of keys used for the Recording Server.

ICVServerControl7::GetServerInfo2

The *GetServerInfo2* method retrieves a brief overview of the server information. This is a synchronous operation.

This method is similar to GetServerInfo except that it accepts a ServerInfo2 object which contains additional fields for failover license information, the product name, and brand information.

Syntax (C++)

```
HRESULT GetServerInfo2 (
   [ref, in, out] ServerInfo2* serverInfo,
   [out] BSTR* error,
   [out, retval] VARIANT_BOOL* result
);
```

Parameters

serverInfo [in/out]

Pointer to a server information object, see ServerInfo2. This must be passed in by API client.

error [out]

Description of error if any error is returned from the Result.

result [out]

Receives a result code indicating the success or failure of the operation. A value of zero indicates success. Any other value is an internal implementation-defined error code and indicates failure.

Return Value

The return value is *S_OK* if the server information is successfully retrieved from the server. The return value can also be *E_POINTER* if a connection to the Recording Server does not exist or *S_FALSE* if another error occurs.

ICVServerControl8

The *ICVServerControl8* interface inherits from the ICVServerControl7 interface. *ICVServerControl8* offers additional properties, summarized in the following table and described in detail in the text that follows.

Property	Description
Token	The user access token to use when connecting to the server.
ProxyMode	Whether or not proxy mode is enabled. Set to true if the server to connect to is a proxy/gateway. Set to false if the server to connect to is a Recording Server. By default, this property is set to false.
RecordingServerId	The Recording Server GUID to send to the proxy/gateway server if proxy mode is enabled. Required in order to use proxy mode so that the proxy/gateway server knows where to forward server control messages.
GetServerInfo3	Retrieves a brief overview of the server information.
GetVideoDaysFromPoolsAsync	Requests a list of days that contain video.

ICVServerControl8::Token

The *Token* property determines the user access token that is used on the next connection to a server. Changing this property has no effect on the current connection (if any). The format of this token must be in JSON Web Token (JWT) format (RFC7519). Use this as an alternative to the UserName and Password properties.

Syntax (C++)

```
HRESULT get_Token(
   [out] BSTR *pVal
```

```
);
HRESULT put_Token(
   [in]  BSTR newVal
);
```

Parameters

pVal [out]

Receives the current value of the property.

newVal [in]

User access token to use in the next connection to a server.

Return Value

The return value is *S_OK*.

ICVServerControl8::ProxyMode

The *ProxyMode* property determines whether or not proxy mode is enabled. Set to true if the server to connect to is a proxy/gateway. Set to false if the server to connect to is a Recording Server. By default, this property is set to false.

Changing this property has no effect on the current connection (if any). In order to start a new connection, a disconnect (if applicable) and a reconnect to the server is required.

Syntax (C++)

```
HRESULT get_ProxyMode(
   [out] VARIANT_BOOL *result
);
HRESULT put_ProxyMode(
   [in]  const VARIANT_BOOL newVal
);
```

Parameters

result [out]

Receives the current value of the property.

newVal [in]

True to enable proxy mode, false to disable proxy mode.

Return Value

The return value is *S_OK*.

ICVServerControl8::RecordingServerId

The *RecordingServerId* property specifies the Recording Server GUID to send to the proxy/gateway server if proxy mode is enabled (i.e. if the ProxyMode property is set to true). This value is required in order to use proxy mode so that the proxy/gateway server knows where to forward server control messages.

Changing this property has no effect on the current connection (if any). In order to start a new connection, a disconnect (if applicable) and a reconnect to the server is required.

Syntax (C++)

```
HRESULT get_RecordingServerId(
   [out] GUID *result
);
HRESULT put_RecordingServerId(
   [in]  const GUID newVal
);
```

Parameters

result [out]

Receives the current value of the property.

newVal [in]

The Recording Server GUID to send to the proxy/gateway server if proxy mode is enabled.

Return Value

The return value is *S_OK*.

ICVServerControl8::GetServerInfo3

The *GetServerInfo3* method retrieves a brief overview of the server information. This is a synchronous operation.

This method is similar to GetServerInfo except that it accepts a ServerInfo3 object which contains additional field for storage mode information.

Syntax (C++)

```
HRESULT GetServerInfo3 (
   [ref, in, out] ServerInfo3* serverInfo,
   [out] BSTR* error,
   [out, retval] VARIANT_BOOL* result
);
```

Parameters

serverInfo [in/out]

Pointer to a server information object, see ServerInfo3. This must be passed in by API client.

error [out]

Description of error if any error is returned from the Result.

result [out]

Receives a result code indicating the success or failure of the operation. A value of zero indicates success. Any other value is an internal implementation-defined error code and indicates failure.

Return Value

The return value is *S_OK* if the server information is successfully retrieved from the server. The return value can also be *E_POINTER* if a connection to the Recording Server does not exist or *S_FALSE* if another error occurs.

ICVServerControl8:: GetVideoDaysFromPoolsAsync

The *GetVideoDaysFromPoolsAsync* method requests a list of the days that contain video. This is an asynchronous operation. The method handler to identify retrieval completion is OnVideoDaysUpdate.

Syntax (C++)

```
HRESULT GetVideoDaysAsync (
    [in]  VARIANT Cameras
    [in]  VARIANT StoragePoolTypes,
    [in]  VARIANT NvrNames,
    [in]  StorageTypeToSearchCOM storageTypeToSearch,
    [out] BSTR* Error,
    [out, retval] VARIANT_BOOL *Result
);
```

Parameters

Cameras [in]

List of cameras to serach. Empty list indicates all cameras.

StoragePoolTypes [in]

List of pool types to serach. Empty list along with `StorageTypeToSearchCOM::Pools` flag indicates all pool types.

NvrNames [in]

List of NVR names to search. Empty list along with `StorageTypeToSearchCOM::Nvrs` flag indicates all NVR's.

StorageTypeToSearchCOM [in]

Type/s of storage to search in. See StorageTypeToSearchCOM

Error [out]

Description of error if any error is returned from the Result.

Result [out]

Receives a result code indicating the success or failure of the operation. A value of zero indicates success. Any other value is an internal implementation-defined error code and indicates failure.

Return Value

The return value is *S_OK*.

ICVServerControl9

The *ICVServerControl9* interface inherits from the ICVServerControl8 interface. *ICVServerControl9* offers an additional method, summarized in the following table and described in detail in the text that follows.

Property	Description
GetLogEventsAsync2	Request a list of log events for the specified time period.

ICVServerControl9:: GetLogEventsAsync2

The *GetLogEventsAsync2* method requests a list of log events for the specified camera(s) and time period. This method requires that at least one event type is requested. This is an asynchronous operation. The method handler to identify retrieval completion is OnLogEventsResult.

This method is similar to GetLogEventsAsync except that a list of cameras can be specified in addition to the time period.

Syntax (C++)

```
HRESULT GetLogEventsAsync2 (
   [in] DATE Start,
   [in] DATE End,
   [in] BSTR TimeZoneStandardName,
   [in] UINT MaxResults,
   [in] VARIANT_BOOL IncludeServerEvents,
   [in] VARIANT Cameras,
   [in] VARIANT EventTypes,
   [out] BSTR* Error,
   [out, retval] VARIANT_BOOL *Result
);
```

Parameters

Start [in]

Start time.

End [in]

End time.

TimeZoneStandardName [in]

String representing the standard timezone to use for this request (see Remarks). e.g. Pacific Standard Time. If no time zone is supplied it will be assumed that the "Local" time zone is requested.

MaxResults [in]

Specifies the maximum number of events to return. A value of -1 specifies that there is no limit.

IncludeServerEvents [in]

Set to true to include server events.

Cameras [in]

List of cameras to include, or empty to include all cameras.

EventTypes [in]

List of events to include. See EventType enumeration definition for a list of supported types.

Error [out]

Description of error if any error is returned from the Result.

Result [out]

Receives a result code indicating the success or failure of the operation. A value of zero indicates success. Any other value is an internal implementation-defined error code and indicates failure.

Return Value

The return value is *S_OK*.

Remarks

It is strongly recommended to use a language-invariant time zone name, rather than a localized time zone name, for the *TimeZoneStandardName* input value.

_ICVServerEvents

Client application developers should implement this interface if it is desirable to receive asynchronous notifications from a CVServerControl.

_ICVServerEvents has methods summarized in the following table and described in the text that follows.

Method	Description
OnConnectionLost	Notifies the client application that the connection to the server has been lost.
OnMotion	Notifies the client application that a motion event has started or ended.
OnAlarm	Notifies the client application that an alarm event has started or ended.
OnSyncLost	Notifies the client application that the server has lost synchronization with respect to a particular camera.
OnSyncGained	Notifies the client application that the server has gained synchronization with respect to a particular camera.
OnConnectionGained	Notifies the client when the connection to the server is successfully established.
OnSmartSearchProgress	Notifies the client periodically once a smart search has been initiated and returns the current date and the current progress in percent complete.
OnSmartSearchResult	Notifies the client when the smart search completes and returns the results of the requested search.
OnSystemStatsUpdate	Notifies the client periodically with updated system statistics.
OnVolumeInfoUpdate	Notifies the client periodically with updated volume information.
OnLicenseInfoUpdate	Notifies the client periodically with updated license information.
OnConnectionInfoUpdate	Notifies the client periodically with updated connection information.
OnEventInfoUpdate	Notifies the client after a connection to the cameras is established. The result contains information about the events as they occur.
OnCameraInfoUpdate	Notifies the client periodically with updated information for a given list of cameras.
OnSnapshotResult	Notifies the client once a requested snapshot is available.
OnEventDaysUpdate	Notifies the client on completion of GetEventDaysAsync query and returns the list of days in the event log that contain events.
OnVideoDaysUpdate	Notifies the client on completion of GetVideoDaysAsync query and returns the list of days that contain video.
OnLogEventsResult	Notifies the client on completion of log event actions and returns the results of the query.
OnDeleteVideoFilesResult	Notifies the client on completion of delete video file actions.
OnAdminServiceTaskResult	Notifies the client on completion of admin service actions.
OnServiceStartup	Notifies the client when a Recording Server service starts up.
OnServiceHeartbeat	Notifies the client with a simple notification that the Recording Server is still running.
OnServiceShutdown	Notifies the client when a Recording Server service gracefully shuts down.

For all of these methods, the return value is ignored by the CVServerControl.

_ICVServerEvents::OnConnectionLost

The *OnConnectionLost* method is called when the connection to the server is lost.

Syntax (C++)

```
HRESULT OnConnectionLost(
    [in] BSTR ServerIP,
    [in] DATE EventTime
);
```

Parameters

ServerIP [in]

The host name or IP address of the server. (This is the same string used when connecting to the server.)

EventTime [in]

Event timestamp, in server local time.

_ICVServerEvents::OnMotion

The *OnMotion* method is called when a motion event starts or ends.

Syntax (C++)

```
HRESULT OnMotion(
    [in]  VARIANT_BOOL  Start,
    [in]  BSTR ServerIP,
    [in]  SHORT CameraID,
    [in]  DATE EventTime
);
```

Parameters

Start [in]

A Boolean value indicating whether the motion has started (*VARIANT_TRUE*) or ended (*VARIANT_FALSE*).

ServerIP [in]

The host name or IP address of the server. (This is the same string used when connecting to the server.)

CameraID [in]

Camera identifier (one-based index).

EventTime [in]

Event timestamp, in server local time.

_ICVServerEvents::OnAlarm

The *OnAlarm* method is called when an alarm starts or ends.

Syntax (C++)

```
HRESULT OnAlaram(
    [in]  VARIANT_BOOL Start,
    [in]  BSTR ServerIP,
    [in]  SHORT CameraID,
```

```
   [in]  DATE EventTime
);
```

Parameters

Start [in]

A Boolean value indicating whether the alarm has started (*VARIANT_TRUE*) or ended (*VARIANT_FALSE*).

ServerIP [in]

The host name or IP address of the server. (This is the same string used when connecting to the server.)

CameraID [in]

Camera identifier (one-based index).

EventTime [in]

Event timestamp, in server local time.

_ICVServerEvents::OnSyncLost

The *OnSyncLost* method is called when the server loses the video feed of a camera.

Syntax (C++)

```
HRESULT OnSyncLost(
   [in]  BSTR ServerIP,
   [in]  SHORT CameraID,
   [in]  DATE EventTime
);
```

Parameters

ServerIP [in]

The host name or IP address of the server. (This is the same string used when connecting to the server.)

CameraID [in]

Camera identifier (one-based index).

EventTime [in]

Event timestamp, in server local time.

_ICVServerEvents::OnSyncGained

The *OnSyncGained* method is called when the server successfully synchronizes to the video feed of a camera.

Syntax (C++)

```
HRESULT OnSyncGained(
   [in]  BSTR ServerIP,
   [in]  SHORT CameraID,
   [in]  DATE EventTime
);
```

Parameters

ServerIP [in]

The host name or IP address of the server. (This is the same string used when connecting to the server.)

CameraID [in]

Camera identifier (one-based index).

EventTime [in]

Event timestamp, in server local time.

_ICVServerEvents::OnConnectionGained

The *OnConnectionGained* method is called when the connection to the server is successfully established.

Syntax (C++)

```
HRESULT OnConnectionGained(
   [in] BSTR ServerIP,
   [in] DATE EventTime
);
```

Parameters

ServerIP [in]

The host name or IP address of the server. (This is the same string used when connecting to the server.)

EventTime [in]

Event timestamp, in server local time.

_ICVServerEvents:: OnSmartSearchProgress

The *OnSmartSearchProgress* event handler is called periodically (every 500ms) after a smart search has been initiated via the StartSmartSearch method. Returns the current date in the time zone specified by the TimeZoneStandard parameter provided when calling the StartSmartSearch, and the current progress in percent complete. Once the percent complete reaches 100 the smart search is ended and OnSmartSearchResult handler is called.

Syntax (C++)

```
HRESULT OnSmartSearchProgress (
   DATE Position,
   INT PercentComplete
);
```

Parameters

Position [in]

Current smart search position in terms of data time.

PercentComplete [in]

Percent smart search complete.

_ICVServerEvents:: OnSmartSearchResult

The *OnSmartSearchResult* event is called when the StartSmartSearch completes.

Syntax (C++)

```
HRESULT OnSmartSearchResult (
   DATE ClipStart,
   DATE ClipEnd,
```

```
   INT ClipCategory,
   BSTR ClipDescription
);
```

Parameters

ClipStart [in]

Actual start time of the result clip.

ClipEnd [in]

Actual end time of the result clip.

ClipCategory [in]

Video category, see RecordingCategories

ClipDescription [in]

Description for the smart search result.

_ICVServerEvents:: OnSystemStatsUpdate

The *OnSystemStatsUpdate* event is only called if the ServerInfoType::TypeSystemStats is enabled via the EnableServerInfo method call. The *OnSystemStatsUpdate* event is called periodically (poll interval is set via the parameter PollIntervalMs in the EnableServerInfo method).

Syntax (C++)

```
HRESULT OnSystemStatsUpdate (
   DOUBLE PhysicalMemoryAvailable,
   DOUBLE PhysicalMemoryUsed,
   DOUBLE VirtualMemoryAvail,
   DOUBLE VirtualMemoryUsed,
   DOUBLE SystemCpuUsage,
   DOUBLE ServerCpuUsage,
   DATE ServerStartTime
);
```

Parameters

PhysicalMemoryAvailable [in]

The amount of actual physical memory, in MB.

PhysicalMemoryUsed [in]

The amount of physical memory utilized, in MB.

VirtualMemoryAvail [in]

The size of the user-mode portion of the virtual address space of the calling process, in MB. This value depends on the type of process, the type of processor, and the configuration of the operating system.

VirtualMemoryUsed [in]

The size of the user-mode portion of the virtual address space of the calling process utilized, in MB.

SystemCpuUsage [in]

Total CPU utilized by the server.

ServerCpuUsage [in]

CPU utilized by the CV server service.

ServerStartTime [in]

Server start time (UTC) as an unsigned 64-bit integer.

_ICVServerEvents:: OnVolumeInfoUpdate

The *OnVolumeInfoUpdate* event is only called if the ServerInfoType::TypeVolume is enabled via the EnableServerInfo method call. The *OnVolumeInfoUpdate* event is called periodically (poll interval is set via the parameter PollIntervalMs in the EnableServerInfo method).

Syntax (C++)

```
HRESULT OnVolumeInfoUpdate (
   VARIANT PathList,
   VARIANT TypeList,
   VARIANT SizeList,
   VARIANT FreeSpaceList,
   VARIANT VolOfflineList,
   VARIANT ExtendedInfoList
);
```

Parameters

PathList [in]

Paths for current server volumes.

TypeList [in]

Types of each of the volumes.

SizeList [in]

Total size for each of the volumes, in KB.

FreeSpaceList [in]

KB free for each of the volumes.

VolOfflineList [in]

True if volume is offline.

ExtendedInfoList [in]

Bit mask for volume information.

_ICVServerEvents:: OnLicenseInfoUpdate

The *OnLicenseInfoUpdate* event is only called if the ServerInfoType::TypeLicense is enabled via the EnableServerInfo method call. The *OnLicenseInfoUpdate* event is called periodically (poll interval is set via the parameter PollIntervalMs in the EnableServerInfo method).

Syntax (C++)

```
HRESULT OnLicenseInfoUpdate (
   BSTR Server, BSTR ProductId,
   INT IpLicenses, INT UsedIpLicenses,
   INT AnalogLicenses, INT UsedAnalogLicenses,
```

```
   INT LicenseType, BSTR LicenseStatus,
   VARIANT FeatureKeys, VARIANT FeatureKeyDescs,
   VARIANT FeatureKeyQuantities);
```

Parameters

Server [in]

The host name or IP address of the server.

ProductId [in]

Product Id.

IpLicenses [in]

Number of IP licenses.

UsedIpLicenses [in]

Number of used IP licenses.

AnalogLicenses [in]

Number of analog licenses.

UsedAnalogLicenses [in]

Number of used analog licenses.

LicenseType [in]

License type.

LicenseStatus [in]

License status.

FeatureKeys [in]

List of feature key strings.

FeatureKeyDescs [in]

List of feature key descriptions.

FeatureKeyQuantities [in]

List of feature key quantities.

_ICVServerEvents:: OnConnectionInfoUpdate

The OnConnectionInfoUpdate event is only called if ServerInfoType::TypeConnection is enabled via the EnableServerInfo method call. The OnConnectionInfoUpdate event is called periodically (poll interval is set via the parameter PollIntervalMs in the EnableServerInfo method).

Syntax (C++)

```
HRESULT OnConnectionInfoUpdate (
   VARIANT IdList,
   VARIANT PeerList,
   VARIANT ClientList,
   VARIANT UserList,
```

```
   VARIANT DescriptionList
);
```

Parameters

IdList [in]

List of connection Ids.

PeerList [in]

List of connected peers.

ClientList [in]

List of clients.

UserList [in]

List of logged in users.

DescriptionList [in]

List of descriptions.

_ICVServerEvents:: OnEventInfoUpdate

The *OnEventInfoUpdate* event is called after connection to the cameras is established and contains information about the events as they occur.

Syntax (C++)

```
HRESULT OnEventInfoUpdate (
   BSTR Server,
   DATE EventTime,
   EventType Type,
   INT Camera,
   INT Connection,
   BSTR User,
   BSTR Description
);
```

Parameters

Server [in]

The host name or IP address of the server. (This is the same string used when connecting to the server.)

EventTime [in]

Event timestamp, in server local time.

Type [in]

Type of event generated by Recording Server. See EventType enumeration.

Camera [in]

Camera identifier (one-based index). -1 if server event.

Connection [in]

Connection number, or 0 if not from a connection.

User [in]

User that triggered event, or empty if no user did.

Description [in]

Description of event.

_ICVServerEvents:: OnCameraInfoUpdate

The *OnCameraInfoUpdate* event is only called if the ServerInfoType::TypeCameraInfo is enabled via the EnableServerInfo method call. The *OnCameraInfoUpdate* event is called periodically (poll interval is set via the parameter PollIntervalMs in the EnableServerInfo method) and contains information about a list of cameras.

Syntax (C++)

```
HRESULT OnCameraInfoUpdate (
   VARIANT IdList,
   VARIANT NameList,
   VARIANT TypeList,
   VARIANT VolumeList,
   VARIANT StatusMaskList
);
```

Parameters

IdList [in]

List of camera identifiers (one-based index).

NameList [in]

List of camera names.

TypeList [in]

Event timestamp, in server local time.

VolumeList [in]

List of volumes that the cameras are recording to. Volume is blank if camera is not currently recording.

StatusMaskList [in]

List of statuses for the cameras. Combination of status bits, see table below.

CameraStatusScheduled = 1 << 0	Scheduled recording is active
CameraStatusAlarm = 1 << 1	Ext alarm detected, ext alarm recording active
CameraStatusMotion = 1 << 2	Motion recording is active
CameraStatusMotionDetected = 1 << 3	Motion has been detected
CameraStatusPtz = 1 << 4	Camera is a PTZ
CameraStatusPrealarm = 1 << 5	Prealarm recording is active
CameraStatusSyncLost = 1 << 6	Camera not connected (analog or IP)
CameraStatusTourEnabled = 1 << 7	Tour enabled
CameraStatusStopped = 1 << 8	Camera is inactive (may or may not be enabled)
CameraStatusAudio = 1 << 9	**Camera has audio**
CameraStatusIp = 1 << 10	Camera is an IP camera
CameraStatusActiModel = 1 << 11	If the camera is ACTI model
CameraStatusNorestart = 1 << 12	Do not restart the camera if stopped
CameraStatusTransitioning = 1 << 13	Transitioning to stopped/started

CameraStatusDigitalptz = 1 << 14	Camera is using Digital PTZ protocol
CameraStatusRecordingFailed = 1 << 15	Recording failed
CameraStatusPresetToggle = 1 << 16	Camera Preset and/or Preset Name changed.
CameraStatusRecordingNvr = 1 << 17	Camera recording on NVR/DVR

_ICVServerEvents:: OnSnapshotResult

The *OnSnapshotResult* event is called on completion of the GetSnapshotAsync method.

Syntax (C++)

```
HRESULT OnSnapshotResult (
   BSTR Server,
   INT CameraId,
   VARIANT Snapshot,
   VARIANT_BOOL Success,
   BSTR Error
);
```

Parameters

Server [in]

The host name or IP address of the server. (This is the same string used when connecting to the server.)

CameraId [in]

Camera identifier (one-based index).

Snapshot [in]

Snapshot retrieved.

Success [in]

Boolean value representing success or failure of the retrieval operation.

Error [in]

Description of error if any.

_ICVServerEvents:: OnEventDaysUpdate

The *OnEventDaysUpdate* event is called to signal completion of the actions started utilizing the GetVideoDaysAsync method.

Syntax (C++)

```
HRESULT OnEventDaysUpdate (
   VARIANT EventDays,
   VARIANT_BOOL Success,
   BSTR Error
);
```

Parameters

EventDays [in]

List of days in the event log that contain events.

Success [in]

Boolean value representing success or failure of the retrieval operation.

Error [in]

Description of error if any.

_ICVServerEvents:: OnVideoDaysUpdate

The *OnVideoDaysUpdate* event is called to signal completion of the actions started utilizing the GetVideoDaysAsync method.

Syntax (C++)

```
HRESULT OnVideoDaysUpdate (
   VARIANT VideoDays,
   VARIANT_BOOL Success,
   BSTR Error
);
```

Parameters

VideoDays [in]

List of days that contain video.

Success [in]

Boolean value representing success or failure of the retrieval operation.

Error [in]

Description of error if any.

_ICVServerEvents:: OnLogEventsResult

The *OnLogEventsResult* event is called to signal completion of the actions started utilizing the GetLogEventsAsync or GetLogEventsAsync2 methods.

Syntax (C++)

```
HRESULT OnLogEventsResult (
   VARIANT EventTimes,
   VARIANT EventTypes,
   VARIANT CameraIds,
   VARIANT ConnectionIds,
   VARIANT Users,
   VARIANT Descriptions,
   VARIANT_BOOL Success,
   BSTR Error
);
```

Parameters

EventTimes [in]

Event timestamps, in server local time.

EventTypes [in]

Event types. See EventType enumeration.

CameraIds [in]

Camera identifier (one-based index). Or -1 if server event.

ConnectionIds [in]

Connection number, or 0 if not from a connection.

Users [in]

User that triggered event, or empty if no user did.

Descriptions [in]

Description of event.

Success [in]

Boolean value representing success or failure of the retrieval operation.

Error [in]

Description of error if any.

_ICVServerEvents:: OnDeleteVideoFilesResult

The *OnDeleteVideoFilesResult* event is called to signal completion of the actions started utilizing the *DeleteVideoFilesAsync* method.

Syntax (C++)

```
HRESULT OnDeleteVideoFilesResult (
   VARIANT_BOOL Success,
   BSTR Error
);
```

Parameters

Success [in]

Boolean value representing success or failure of the retrieval operation.

Error [in]

Description of error if any.

_ICVServerEvents:: OnAdminServiceTaskResult

The *OnAdminServiceTaskResult* event is called to signal completion of the actions started utilizing one of the following methods: *StartRecordingServiceAsync*, *StopRecordingServiceAsync*, or *RestartRecordingServiceAsync*.

Syntax (C++)

```
HRESULT OnAdminServiceTaskResult (
   BSTR Server,
   AdminServiceTask Task,
   VARIANT_BOOL Success,
   BSTR Error
);
```

Parameters

Server [in]

The host name or IP address of the server. (This is the same string used when connecting to the server.)

AdminServiceTask [in]

Complete task. See AdminServiceTask enumeration.

Success [in]

Boolean value representing success or failure of the retrieval operation.

Error [in]

Description of error if any.

_ICVServerEvents:: OnServiceStartup

The *OnServiceStartup* event is called to signal that a Recording Server service has started running. This event is sent via a broadcast message. To receive this message the broadcast listener must be started using the *StartServiceListener* method.

Syntax (C++)

```
HRESULT OnServiceStartup (
   BSTR server,
   DATE serviceStartupTime,
   BSTR serverId
);
```

Parameters

Server [in]

The host name or IP address of the server. (This is the same string used when connecting to the server.)

DATE [in]

Recording Server service startup time.

BSTR [in]

Recording Server identification, this is a GUID.

_ICVServerEvents:: OnServiceHeartbeat

The *OnServiceStartup* event is called to signal the Recording Server is running.

Syntax (C++)

```
HRESULT OnServiceHeartbeat (
   BSTR server,
   DATE heartbeatTime,
   BSTR serverId
);
```

Parameters

Server [in]

The host name or IP address of the server. (This is the same string used when connecting to the server.)

DATE [in]

Recording Server service heartbeat time.

BSTR [in]

Recording Server identification, this is a GUID.

_ICVServerEvents:: OnServiceShutdown

The *OnServiceShutdown* event is called to signal that the Recording Server has gracefully shutdown.

Syntax (C++)

```
HRESULT OnServiceStartup (
   BSTR server,
   DATE serviceShutdownTime,
   BSTR serverId
);
```

Parameters

Server [in]

The host name or IP address of the server. (This is the same string used when connecting to the server.)

DATE [in]

Recording Server service shutdown time.

BSTR [in]

Recording Server identification, this is a GUID.

_ICVServerEvents2

The *_ICVServerEvents2* interface inherits from the _ICVServerEvents interface. *_ICVServerEvents2* offers additional methods, summarized in the following table and described in detail in the text that follows.

Method	Description
OnLicenseInfoUpdate2	Notifies the client periodically with updated license information.
OnSystemStatsUpdate2	Notifies the client periodically with updated system statistics.

_ICVServerEvents2::OnLicenseInfoUpdate2

The *OnLicenseInfoUpdate2* event is only called if the ServerInfoType::TypeLicense is enabled via the EnableServerInfo method call. The *OnLicenseInfoUpdate2* event is called periodically (poll interval is set via the parameter PollIntervalMs in the EnableServerInfo method). Trial liceses will return no keys therefor to get the info call `GetTrialLicenseInfo`.

Syntax (C++)

```
HRESULT OnLicenseInfoUpdate2 (
   BSTR Server, BSTR ProductId,
   INT LicenseType, BSTR LicenseStatus,
   VARIANT FeatureKeys, VARIANT FeatureKeyDescs,
   VARIANT FeatureKeyQuantities,
   VARIANT FeatureKeyUsedQuantities);
```

Parameters

Server [in]

The host name or IP address of the server.

ProductId [in]

Product Id.

LicenseType [in]

License type.

LicenseStatus [in]

License status.

FeatureKeys [in]

List of feature key strings.

FeatureKeyDescs [in]

List of feature key descriptions.

FeatureKeyQuantities [in]

List of feature key quantities.

FeatureKeyUsedQuantities [in]

List of feature key used quantities.

_ICVServerEvents2:: OnSystemStatsUpdate2

The *OnSystemStatsUpdate2* event is only called if the ServerInfoType::TypeSystemStats is enabled via the EnableServerInfo method call. The *OnSystemStatsUpdate2* event is called periodically (poll interval is set via the parameter PollIntervalMs in the EnableServerInfo method). This method provides two additional parameters beyond those provided in *_ICVServerEvents::OnSystemStatsUpdate*.

Syntax (C++)

```
HRESULT OnSystemStatsUpdate2(
    DOUBLE physicalMemoryAvailable,
    DOUBLE physicalMemoryUsed,
    DOUBLE virtualMemoryAvail,
    DOUBLE virtualMemoryUsed,
    DOUBLE systemCpuUsage,
    DOUBLE serverCpuUsage,
    DOUBLE serverMemAvail,
    DOUBLE serverMemUsed,
    DATE ServerStartTime
);
```

Parameters

physicalMemoryAvailable [in]

The amount of actual physical memory, in MB.

physicalMemoryUsed [in]

The amount of physical memory utilized, in MB.

virtualMemoryAvail [in]

The size of the user-mode portion of the virtual address space of the calling process, in MB. This value depends on the type of process, the type of processor, and the configuration of the operating system.

virtualMemoryUsed [in]

The size of the user-mode portion of the virtual address space of the calling process utilized, in MB.

systemCpuUsage [in]

Total CPU utilized by the server.

serverCpuUsage [in]

CPU utilized by the CV server service.

serverMemAvail [in]

Server memoyr available for use, in MB.

serverMemUsed [in]

The amount of used server memory, in MB.

serverStartTime [in]

Server start time (UTC) as an unsigned 64-bit integer.

_ICVServerEvents3

The *_ICVServerEvents3* interface inherits from the _ICVServerEvents2 interface. *_ICVServerEvents3* offers additional methods, summarized in the following table and described in detail in the text that follows.

Method	Description
OnEntityConfigChange	Notifies the client of configuration changes applied to the Recording Server.

_ICVServerEvents3::OnEntityConfigChange

The *OnLicenseInfoUpdate3* event is raised when changes to the Recording Server's configuration are applied.

Syntax (C++)

```
HRESULT OnEntityConfigChange (
   BSTR server, EntityType entityType,
   DATE eventTime, BSTR entityId,
   BSTR description
);
```

Parameters

server [in]

The host name or IP address of the server on which the change occurred.

entityType [in]

The type of entity that changed. See EntityType.

eventType [in]

The time at which the change event occurred. This is the Recording Server's local time.

entityId [in]

This string contains the unique identifier of the item that changed. The string contains a GUID.

description [in]

A description of the entity that changed.

_ICVServerEvents3::OnStoragePoolInfoUpdate

The *OnStoragePoolInfoUpdate* event is raised when changes to the Recording Server's storage pool/s info is/are retrieved.

Syntax (C++)

```
HRESULT OnStoragePoolInfoUpdate (
    BSTR poolGuidStr,
    StoragePoolType poolType,
    StoragePoolStatus poolStatus,
    UINT64 poolDriveCapacity,
    UINT64 poolDriveFreeSpace,
    UINT64 poolVideoCapacity,
    UINT64 poolVideoFreeSpace,
    UINT64 outsideStorage,
    UINT64 camStorageNoMinRetention,
    UINT64 camStorageWithMinRetention,
    VARIANT drivePath,
    VARIANT driveEnabled,
    VARIANT online,
    VARIANT overflowDrive,
    VARIANT capacity,
    VARIANT freeSpace,
    VARIANT reserveSpace,
    VARIANT videoCapacity,
    VARIANT videoFreespace,
    VARIANT driveOutsideStorage,
    VARIANT driveCamStorageNoMinRetention,
    VARIANT driveCamStorageWithMinRetention,
    VARIANT_BOOL Success,
    BSTR Error
);
```

Parameters

poolGuidStr [in]

Storage Pool GUID, string version.

poolType [in]

Storage Pool type. See StoragePoolType

poolStatus [in]

Storage Pool status (Good/Degraded/Failed). See StoragePoolStatus

poolDriveCapacity [in]

Total drive capacity of all drives that make up the pool (excluding overflow drives).

PoolDriveFreespace [in]

Total drive freespace of all drives that make up the pool (excluding overflow drives).

PoolVideoCapacity [in]

Total video space of all drives that make up the pool (excluding overflow drives).

PoolVideoFreespace [in]

Total video freespace of all drives that make up the pool (excluding overflow drives).

OutsideStorage [in]

Combined space consumed by outside storage in the pool.

CamStorageNoMinRetention [in]

Storage consumed in this pool by cameras with no minimum retention specified.

CamStorageWithMinRetention [in]

Storage consumed in this pool by cameras with with a minimum retention specified.

Note: ***The fields below are all variant and hold an array of items. Only if drive information is requested will the fields below be populated. Otherwise, these will return with a null value.***

DrivePath [in]

Full path to the root for video storage. – Returns an array of BSTR items.

DriveEnabled [in]

Is the drive enabled? - Returns an array of VARIANT_BOOL items.

Online [in]

Is the drive online? - Returns an array of VARIANT_BOOL items.

OverflowDrive [in]

Is this drive an overflow drive? - Returns an array of VARIANT_BOOL items.

Capacity [in]

Total drive capacity of the physical drive. - Returns an array of UINT64 items.

Freespace [in]

Total drive freespace of the physical drive. - Returns an array of UINT64 items.

ReserveSpace [in]

The drives reserve (safty buffer) space. - Returns an array of UINT64 items.

VideoCapacity [in]

Total video space allocated for this drive. - Returns an array of UINT64 items.

VideoFreespace [in]

Total video freespace of this drive (driveFreespace - driveReserveSpace). - Returns an array of UINT64 items.

DriveOutsideStorage [in]

Combined space cousumed by outside storage (videoCapacity - videoFreespace - videoStorage). - Returns an array of UINT64 items.

DriveCamStorageNoMinRetention [in]

Storage consumed on this drive by cameras with no minimum retention specified. - Returns an array of UINT64 items.

DriveCamStorageWithMinRetention [in]

Storage consumed on this drive by cameras with with a minimum retention specified. - Returns an array of UINT64 items.

Success [in]

Boolean value representing success or failure of the retrieval operation.

Error [in]

Description of error if any.

CVServerControl API Enumerations

AdminServiceTask

The *AdminServiceTask* enumeration defines the types of Admin Service tasks.

Syntax (C++)

```
enum AdminServiceTask {
   None,
   StartRecordingService,
   StopRecordingService,
   RestartRecordingService
};
```

CameraPermission

The *CameraPermission* enumeration defines the types of per-camera access permissions for a user.

Syntax (C++)

```
enum CameraPermission {
   VIEW          = 1<<0,
   PLAYBACK      = 1<<1,
   PTZ           = 1<<2,
   EXPORT        = 1<<3,
   SNAPSHOT      = 1<<4,
   AUDIO         = 1<<5,
};
```

Remarks

The members of the *CameraPermission* enumeration are bit flags corresponding to the access permissions defined on a per-user, per-camera basis:

VIEW	The user is authorized to view the live output of the camera.
PLAYBACK	The user is authorized to view recorded video from the camera.
PTZ	The user is authorized to conduct pan-tilt-zoom control of the camera.
EXPORT	The user is authorized to export video from the camera.
SNAPSHOT	The user is authorized to export snapshots of the video from the camera.
AUDIO	The user is authorized to enable audio on the camera.

EventType

The EventType enumeration defines events generated by the Recording Server.

Syntax (C++)

```
enum EventType {
   Unknown                  = 0,
   Begin                    = 1,
   Login                    = 1,
   Logout                   = 2,
   FailedLogin              = 3,
   Start                    = 4,
   Stop                     = 5,
   Restart                  = 6,
   Error                    = 7,
   AlarmStart               = 8,
   AlarmEnd                 = 9,
   MotionStart              = 10,
   MotionEnd                = 11,
   SyncLost                 = 12,
   SyncGained               = 13,
   SaveConfig               = 14,
   SetTime                  = 15,
   Info                     = 16,
   TcpConnected             = 17,
   TcpDisconnected          = 18,
   StartLiveView            = 19,
   StartPlayback            = 20,
   Debug                    = 21,
   RecFailed                = 22,
   DftStart                 = 23,
   DftEnd                   = 24,
   VideoExport              = 25,
   VolumeOnline             = 26,
   VolumeOffline            = 27,
   VolumeFull               = 28,
   TriggerActivated         = 29,
   TriggerDeactivated       = 30,
```

```
    InputActivated            = 31,
    InputDeactivated          = 32,
    OutputActivated           = 33,
    OutputDeactivated         = 34,
    VolumeMinStorageViolated  = 35,
    RecStartedAfterFailed     = 36,
    RtspChannelCreated        = 37,
    RtspChannelDestroyed      = 38,
    RtspLiveConnected         = 39,
    RtspLiveDisconnected      = 40,
    InsufficientRetention     = 43,
    MinRetentionViolation     = 44,
    PoolDriveOffline          = 45,
    DeleteFailed              = 46,
    FreePoolSpaceFailed       = 47,
    OverflowDriveActive       = 48,
    MinCamRetentionActive     = 49,
    PoolDriveOnline           = 50,
    OverflowDriveInactive     = 51,
    StorageThresholdMet       = 52,
    WriteFailed               = 53,
    GenericCamera             = 54,
    DeviceCamera              = 55,
    VideoSourceCamera         = 56,
    VideoEncoderCamera        = 57,
    VideoAnalyticsCamera      = 58,
    RuleEngineCamera          = 59,
    PtzControllerCamera       = 60,
    AudioSourceCamera         = 61,
    AudioEncoderCamera        = 62,
    UserAlarmCamera           = 63,
    MediaControlCamera        = 64,
    RecordingConfigCamera     = 65,
    RecordingHistoryCamera    = 66,
    VideoOutputCamera         = 67,
    AudioOutputCamera         = 68,
    VideoDecoderCamera        = 69,
    AudioDecoderCamera        = 70,
    ReceiverCamera            = 71,
    MonitoringCamera          = 72,
    End                       = 73,
};
```

Remarks

These event types are not tied to logging, logging is one of actions as a result of these events.

Recording Server-specific Events

Unknown	Unrecognized event
Begin	Lower bound

Login	Login
Logout	Logout
FailedLogin	Login failed
Start	Main server starts
Stop	Main server stops
Restart	Main server restarts
Error	General error occurred
AlarmStart	Alarm started
AlarmEnd	Alarm ended
MotionStart	Motion detection started
MotionEnd	Motion detection ended
SyncLost	Synchronization to server lost
SyncGained	Synchronization to server gained
SaveConfig	Server configuration changes saved
SetTime	Set server time
Info	Informational message
TcpConnected	Accepted TCP connection
TcpDisconnected	Closed TCP connection
StartLiveView	Live view started
StartPlayback	Recorded playback started
Debug	Debug message
RecFailed	Error recording video to volume
DftStart	DFT started
DftEnd	DFT ended
VideoExport	Export of video occurred
VolumeOnline	Recording volume online
VolumeOffline	Recording volume offline
VolumeFull	Recording volume full
TriggerActivated	Trigger activated
TriggerDeactivated	Trigger deactivated
InputActivated	Input pin on the device was activated
InputDeactivated	Input pin on the device was deactivated
OutputActivated	Output activated
OutputDeactivated	Output deactivated
VolumeMinStorageViolated	Volume minimum storage time period violated due to free space delete
RecStartAfterFailed	Recording has restarted after a recording failure
RtspChannelCreated	An rtsp channel has been created
RtspChannelDestroyed	An rtsp channel has been removed
RtspLiveConnected	A live connection has been established by a client to an rtsp channel
RtspLiveDisconnected	A live connection has been closed by a client to an rtsp channel
InsufficientRetention	Storage pool drive space for a particular drive is projected to be insufficient to meet expected retention
MinRetentionViolation	Minimum retention for a particular storage pool has been violated
PoolDriveOffline	A drive in a storage pool has gone offline
DeleteFailed	Failed to delete video clip
FreePoolSpaceFailed	Failed to free up space for a particular storage pool
OverflowDriveActive	Writing to overflow drive is active
MinCamRetentionActive	Video is being deleted from cameras with a minimum retention policy
PoolDriveOnline	A drive in a storage pool has gone online
OverflowDriveInactive	Writing to overflow drive is inactive

StorageThresholdMet	Pool storage has fallen below it's set percent threshold
WriteFailed	Failed to write video clip
GenericCamera	A generic camera event occurred
DeviceCamera	A device camera event occurred
VideoSourceCamera	A video source camera event occurred
VideoEncoderCamera	A video encoder camera event occurred
VideoAnalyticsCamera	A video analytics camera event occurred
RuleEngineCamera	A rule engine camera event occurred
PtzControllerCamera	A PTZ controller camera event occurred
AudioSourceCamera	An audio source camera event occurred
AudioEncoderCamera	An audio encoder camera event occurred
UserAlarmCamera	A user alarm camera event occurred
MediaControlCamera	A media control camera event occurred
RecordingConfigCamera	A recording config camera event occurred
RecordingHistoryCamera	A recording history camera event occurred
VideoOutputCamera	A video output camera event occurred
AudioOutputCamera	An audio output camera event occurred
VideoDecoderCamera	A video decoder camera event occurred
AudioDecoderCamera	An audio decoder camera event occurred
ReceiverCamera	A receiver camera event occurred
MonitoringCamera	A monitoring camera event occurred
End	Upper bound

RecordingCategory

The *RecordingCategory* enumeration will be used by clients to specify what category of clips should be searched.

Syntax (C++)

```
enum RecordingCategory {
   Continuous    = 1<<0,
   Alarm         = 1<<1,
   Motion        = 1<<2,
   Exported      = 1<<3
};
```

Remarks

The members of the *RecordingCategory* enumeration are bit flags corresponding to the type of video category defined on a per-camera basis:

Continuous	Scheduled recording.
Alarm	Alarm recording.
Motion	Motion detection recording.
Exported	Exported clips.

ServerInfoType

The ServerInfoType enumeration will be used by clients to specify what types of server information to send to the subscribed client.

Syntax (C++)

```
enum ServerInfoType {
   TypeAll             = 0,
   TypeSystemStats     = 1,
   TypeVolume          = 2,
   TypeLicense         = 3,
   TypeCameraInfo      = 4,
   TypeConnection      = 5,
   TypeStoragePool     = 6,
};
```

Remarks

TypeAll	All types.
TypeSystemStats	System statistics.
TypeVolume	Volume information.
TypeLicense	License information.
TypeCameraInfor	Camera information.
TypeConnection	Connection information.
TypeStoragePool	Storage Pool/s information that is/are configured in server.

EntityType

The EntityType enumeration is used when reporting changes to entities within a Recording Server. It specifies the type of the entity that was changed so the the subscribed client can react accordingly.

Syntax (C++)

```
enum EntityType {
   Undefined     = -1,
   Server        =  0,
   Camera        =  1,
   Volume        =  2,
   AlarmType     =  3,
   User          =  4,
   Trigger       =  5,
   Logger        =  6,
   View          =  7,
   Map           =  8,
   Task          =  9,
   Program       = 10,
   Schedule      = 11,
   StoragePool   = 12
};
```

Remarks

The specific types of Recording Server entities whose configuration changes are reported.

Undefined	An unspecified entity type configuration changed.
Server	Server configuration changed.

Camera	Camera configuration changed.
Volume	Volume configuration changed.
Alarm	Alarm configuration changed.
User	User configuration changed.
Trigger	Trigger configuration changed.
Logger	Logger configuration changed.
View	View configuration changed.
Map	Map configuration changed.
Task	Not used for configuration changed events.
Program	Not used for configuration changed events.
Schedule	Schedule configuration changed.
StoragePool	Storage pool configuration changed.

StoragePoolType

The StoragePoolType enumeration is used when requesting storage pool statistics.

Syntax (C++)

```
enum StoragePoolType {
   Regular       =  0,
   Archive       =  1,
   Backup        =  2,
   ExportPool    =  3,
   Recorder      =  4,
   Failover      =  5,
   Overflow      =  6
};
```

Remarks

StorageModeManager

The StorageModeManager enumeration denotes types of storage modes supported by the Recording Server.

Syntax (C++)

```
enum StorageModeManager {
   ManagerNone         =  0,
   ManagerVolume       =  1,
   ManagerStoragePool  =  2,
};
```

Remarks

StoragePoolStatus

The StoragePoolStatus enumeration denotes types of storage status supported by the Recording Server.

Syntax (C++)

```
enum StoragePoolStatus {
   StatusUnknown = -1,
   Good          = 0,
   Degraded      = 1,
   Failed        = 2,
};
```

Remarks

StorageTypeToSearchCOM

The StorageTypeToSearchCOM enumeration denotes types of storage to search videoes in.

Syntax (C++)

```
enum class StorageTypeToSearchCOM : int {
   Pools = 1 << 0,
   Nvrs = 1 << 1
};
```

Remarks

CVServerControl API Structures

CameraDetails

The CameraDetails structure defines fields and data types. CameraDetails is required in order to use the GetCameraDetails method.

Syntax (C++)

```
typedef struct CameraDetails
{
   VARIANT_BOOL UserCanView;
   VARIANT_BOOL UserCanPlayback;
   VARIANT_BOOL UserCanMultiPlayback;
   VARIANT_BOOL UserCanExport;
   VARIANT_BOOL UserCanSnapshot;
   VARIANT_BOOL UserCanToggleLight;
   VARIANT_BOOL UserCanToggleAudio;
   VARIANT_BOOL UserCanBurnDvd;
   VARIANT_BOOL CameraHasLight;
   VARIANT_BOOL DeviceSupportsExport;
   VARIANT_BOOL IsDSPEnabled;

   VARIANT_BOOL HasImmervisionLens;
   VARIANT_BOOL UserCanPtz;
   VARIANT_BOOL ThumbnailSupportAvailable;
```

```
   UINT Status;
   INT CameraType;
   INT VideoSize;
   INT HotspotFPS;
   INT HotspotCompressorType;
   INT CameraPosition;
   INT LensProfile;
   INT DewarpLensType;
   INT FisheyeCenterX;
   INT FisheyeCenterY;
   INT FisheyeRadius;
   INT CameraId;
   BSTR Name;
   BSTR Volume;
   BSTR TimeZoneStandardName;
   VARIANT PresetNameList;
   VARIANT LensCurveData;
} CameraDetails;
```

Remarks

UserCanView	Logged in user is allowed to view this camera.
UserCanPlayback	Logged in user is allowed to play back this camera.
UserCanMultiPlayback	Logged in user is allowed multi-playback for this camera.
UserCanExport	Logged in user is allowed to play back and export this camera.
UserCanSnapshot	Logged in user is allowed to snapshot.
UserCanToggleLight	Logged in user is allowed to toggle camera light.
UserCanToggleAudio	Logged in user is allowed to toggle camera audio.
UserCanBurnDvd	Logged in user can burn clips to CD/DVD.
CameraHasLight	Camera light capability.
DeviceSupportsExport	Camera/NVR channel supports export.
IsDSPEnabled	Disable Server Processing. True if DSP is set for this camera.
HasImmervisionLens	True if the feature is present.
UserCanPtz	True if user has permissions to PTZ.
ThumbnailSupportAvailable	Camera supports thumbnails.
Status	Camera status bit mask.
CameraType	Camera type
VideoSize	Video size.
HotspotFPS	Include camera hot spot FPS.
HotspotCompressorType	Include camera hot spot compressor type.
CameraPosition	Fisheye lens position.
LensProfile	Immervision lens profile.
DewarpLensType	Fisheye lens type (STANDARD if non-fisheye).
FisheyeCenterX	Fisheye center's X coordinate.
FisheyeCenterY	Fisheye center's Y coordinate.
FisheyeRadius	Fisheye calibration circle radius.
CameraId	Camera ID.
Name	Name of the camera.
Volume	Volume path of the camera.

TimeZoneStandardName	Time zone standard name (may be localized).
PresetNameList	Preset Name List (list of string).
LensCurveData	Camera's Binary Large OBject (BLOB) Data / Lens curve data.

IoDetails

The IoDetails structure defines fields and data types used to describe I/O devices. IoDetails is required in order to use the GetIoDetailByIndex and GetIoDetailByGuid methods.

Syntax (C++)

```
typedef struct IoDetails
{
   GUID deviceGuid;
   BSTR deviceName;
   INT deviceNumber;
   VARIANT outputNumbers;
   VARIANT outputNames;
   VARIANT deviceTypes;
   VARIANT triggerStates;
} IoDetails;
```

Remarks

The outputNumbers, outputNames, deviceTypes and triggerStates are all parameters used to represent outputs on an I/O device. The parameters have been flattened into arrays of base types in order to enable marshaling of the nested user defined OutputInfo structure via the API. Once received, these parameters may be grouped together based on their indices within the array, to represent an OutputInfo object.

deviceGuid	Guid to uniquely identify I/O device.
deviceName	Device name.
deviceNumber	Index to identify I/O device.
outputNumbers	Indices of outputs associated with device. This is stored as an array of 4 byte integers, encapsulated into a VARIANT.
outputNames	Names of outputs associated with device. This is stored as an array of strings, encapsulated into a VARIANT.
deviceTypes	Device type of outputs associated with device. This is stored as an array of 4 byte integers, encapsulated into a VARIANT.
triggerStates	Whether outputs associated with device are set or reset. This is stored as an array of 4 byte integers, encapsulated into a VARIANT.

QueueExportClipInfo

This struct specifies the settings used when exporting a video clip to a Recording Server export volume.

Syntax (C++)

```
typedef struct QueueExportClipInfo
{
   BSTR Description;
   INT CameraId;
   DATE StartTime;
   DATE EndTime;
   BSTR ServerTimeZone;
   BSTR CameraTimeZone;
   UINT64 ClipSizeKb;
   INT LensType;
   INT LensOrientation;
   INT FisheyeCenterX;
   INT FisheyeCenterY;
   INT FisheyeRadius;
   INT LensProfile;
   INT ExportType;
} QueueExportClipInfo;
```

Remarks

Description	Description of the video clip.
CameraId	Camera ID.
StartTime	Start time of the video clip. This must be in the time zone of the local machine that issues this export request.
EndTime	End time of the video clip. This must be in the time zone of the local machine that issues this export request.
ServerTimeZone	Time zone of the Recording Server (may be localized).
CameraTimeZone	Time zone of the camera (may be localized).
ClipSizeKb	File size in kilobytes of the video clip.
LensType	Fisheye lens type (STANDARD if non-fisheye).
LensOrientation	Fisheye lens orientation.
FisheyeCenterX	Fisheye center's X coordinate.
FisheyeCenterY	Fisheye center's Y coordinate.
FisheyeRadius	Fisheye calibration circle radius.
LensProfile	Immervision lens profile.
ExportType	Export type (1 = queue, 2 = trailing).

ServerInfo

The ServerInfo structure defines fields that describe information about the Recording Server. ServerInfo is required in order to use the ICVServerControl6::GetServerInfo method.

Syntax (C++)

```
typedef struct ServerInfo
{
   BSTR FriendlyName;
   BSTR IpAddress;
```

```
   BSTR Version;
   BSTR Guid;
   BSTR ProductId;
   BSTR LicenseType;
   INT UsedIpLicenses;
   INT AvailableIpLicenses;
   INT UsedAnalogLicenses;
   INT AvailableAnalogLicenses;
} ServerInfo;
```

Remarks

FriendlyName	The server friendly name.
IpAddress	The server IP address.
Version	The server version.
Guid	The server GUID.
ProductId	The product ID.
LicenseType	The license type.
UsedIpLicenses	The number of IP licenses used on the server.
AvailableIpLicenses	The number of IP licenses available on the server.
UsedAnalogLicenses	The number of analog licenses used on the server.
AvailableAnalogLicenses	The number of analog licenses available on the server.

ServerInfo2

The ServerInfo2 structure defines fields that describe information about the Recording Server. ServerInfo2 is required in order to use the ICVServerControl7::GetServerInfo2 method.

The ServerInfo2 structure is similar to ServerInfo except that it contains additional fields for failover license information, the product name, and brand information.

Syntax (C++)

```
typedef struct ServerInfo2
{
   BSTR FriendlyName;
   BSTR IpAddress;
   BSTR Version;
   BSTR Guid;
   BSTR ProductId;
   BSTR LicenseType;
   INT UsedIpLicenses;
   INT AvailableIpLicenses;
   INT UsedAnalogLicenses;
   INT AvailableAnalogLicenses;
   INT UsedFailoverLicenses;
   INT AvailableFailoverLicenses;
   BSTR ProductName;
   VARIANT_BOOL IsBranded;
} ServerInfo2;
```

Remarks

FriendlyName	The server friendly name.
IpAddress	The server IP address.
Version	The server version.
Guid	The server GUID.
ProductId	The product ID.
LicenseType	The license type.
UsedIpLicenses	The number of IP licenses used on the server.
AvailableIpLicenses	The number of IP licenses available on the server.
UsedAnalogLicenses	The number of analog licenses used on the server.
AvailableAnalogLicenses	The number of analog licenses available on the server.
UsedFailoverLicenses	The number of failover licenses used on the server.
AvailableFailoverLicenses	The number of failover licenses available on the server.
ProductName	The product name.
IsBranded	Whether or not the product is branded. False for Salient CompleteView installations, true otherwise.

ServerInfo3

The ServerInfo3 structure defines fields that describe information about the Recording Server. ServerInfo3 is required in order to use the ICVServerControl8::GetServerInfo3 method.

The ServerInfo3 structure is similar to ServerInfo except that it contains additional field for storage mode information.

Syntax (C++)

```
typedef struct ServerInfo3
{
   BSTR FriendlyName;
   BSTR IpAddress;
   BSTR Version;
   BSTR Guid;
   BSTR ProductId;
   BSTR LicenseType;
   INT UsedIpLicenses;
   INT AvailableIpLicenses;
   INT UsedAnalogLicenses;
   INT AvailableAnalogLicenses;
   INT UsedFailoverLicenses;
   INT AvailableFailoverLicenses;
   BSTR ProductName;
   VARIANT_BOOL IsBranded;
   INT StorageMode;
} ServerInfo3;
```

Remarks

FriendlyName	The server friendly name.
IpAddress	The server IP address.
Version	The server version.
Guid	The server GUID.
ProductId	The product ID.
LicenseType	The license type.
UsedIpLicenses	The number of IP licenses used on the server.
AvailableIpLicenses	The number of IP licenses available on the server.
UsedAnalogLicenses	The number of analog licenses used on the server.
AvailableAnalogLicenses	The number of analog licenses available on the server.
UsedFailoverLicenses	The number of failover licenses used on the server.
AvailableFailoverLicenses	The number of failover licenses available on the server.
ProductName	The product name.
IsBranded	Whether or not the product is branded. False for Salient CompleteView installations, true otherwise.
StorageMode	An enum value of type "StorageModeManager".

CVClientControl API

The CVClientControl API provides a set of interfaces that allow applications to stream video from CompleteView servers (possibly with audio). There are two primary interfaces:

- ICVVideo, which represents an abstract connection to a particular camera on a CompleteView server. This interface is implemented by the CVVideo ActiveX control and invoked by the application.
- ICVVideoEvents, through which the CVVideo ActiveX control provides the application with asynchronous notifications of significant events. This interface is (optionally) implemented by the application.

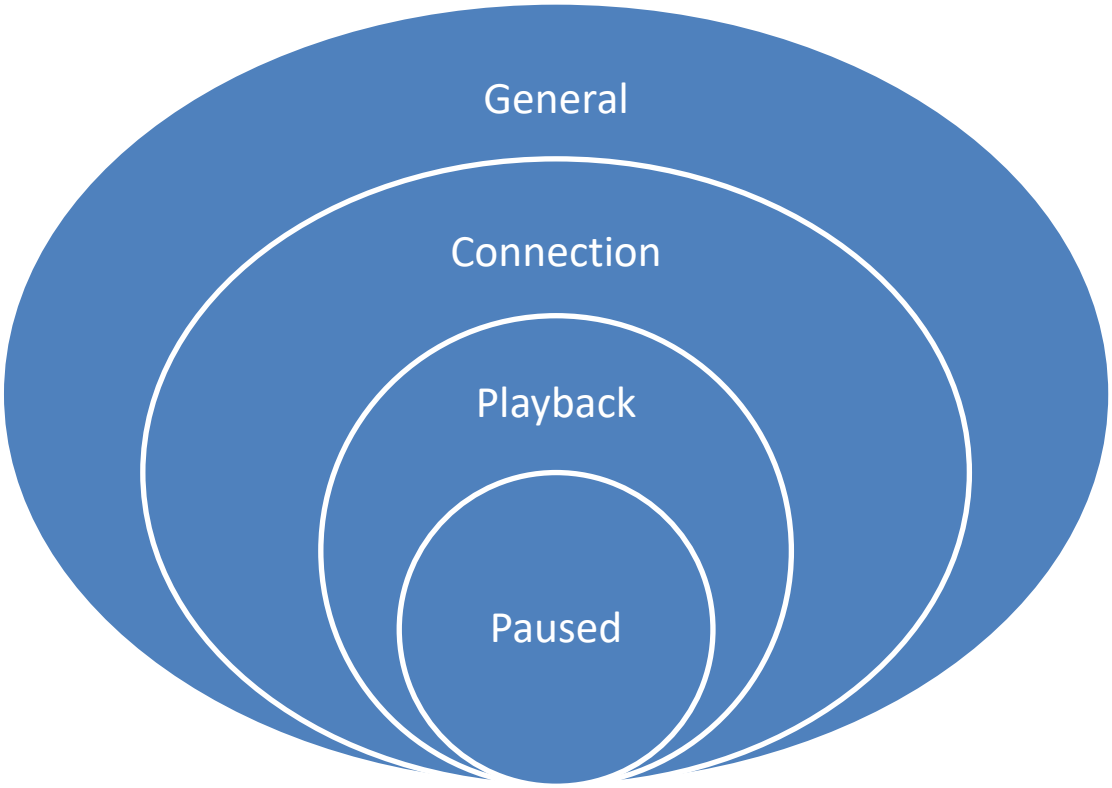
A secondary interface, ICVVideo2, extends *ICVVideo*. *ICVVideo2* can be thought of as a newer version of *ICVVideo*.

For the sake of brevity, in the remainder of this section we often refer to the CVVideo Active X control as (simply) the CVVideo control.

Using the CVClientControl API

Method Contexts

Every method in the CVClientControl API has a *context* in which it makes sense to call that method. These contexts roughly correspond to *states* of the CVVideo control. There are four such contexts, illustrated in the following Venn diagram:



Note that these contexts are fully nested. A contained context is a restriction of its containing contexts. In other words, a method associated with an outer (containing) context can always be used profitably in an inner (contained) context.

General Context

An application can sensibly call the methods associated with the **General** context at any time. The methods associated with the **General** context do not require the previous successful execution of any other methods.

There are three classes of methods associated with the General context:

1. The truly general methods do use the values of any properties.
2. The server-oriented methods rely on the values of the Server, Username, and Password properties, so it only makes sense to call these methods after setting these three properties.
3. The camera-oriented methods rely on the value of the Camera property in addition to the *Server*, *Username*, and *Password* properties. It only makes sense to call these methods after setting all four of these properties.

Here is the full list of CVClientControl methods associated with the **General** context (organized by class):

- GetFilePathName [truly general]
- GetViewOption [truly general]
- SetViewOption [truly general]
- ConnectSync [server-oriented]
- ConnectSyncResult [server-oriented]
- GetCameraInfo [server-oriented]

- GetNumberOfCameras [server-oriented]
- GetServerTime (*ICVVideo2*) [server-oriented]
- GetServerTimeZone (*ICVVideo2*) [server-oriented]
- Connect [camera-oriented]
- SearchVideo (*ICVVideo2*) [camera-oriented]
- SetTimeZone (*ICVVIdeo2*) [truly general]

Particularly noteworthy is the *Connect* method. The successful execution of *Connect* causes the CVVideo control to enter the **Connection** context, described next.

Connection Context

An application can sensibly invoke the methods associated with the **Connection** context only after a successful execution of the Connect method, and before any subsequent execution of the Disconnect method. In the **Connection** context, there is an established connection to a camera (and implicitly to a server administering that camera). We refer to this camera as the *current camera* (and the server as the *current server*.)

One important thing to note about the **Connection** context (and implicitly those contexts nested within it) are that they do not depend on the current values of the *Server*, *Username*, *Password*, and *Camera* properties, i.e. the properties that helped you get there. These properties can be changed while in the **Connection** context without affecting the context in any way, though it may be considered ill-advised to make such changes.

Here is the full list of CVClientControl methods associated with the **Connection** context:

- AutoFocus
- AutoIris
- Disconnect
- EnableAudio
- Focus
- GotoPreset
- Iris
- Playback
- SaveSnapshot
- SaveSnapshotScaled
- SetCameraOverlay (*DEPRECATED in version 5.X, use SetServerCameraOverlay*)
- SetPreset
- GetSnapshot (*ICVVideo2*)
- GetSnapshotScaled (*ICVVideo2*)
- SetClientCameraOverlay (*ICVVideo3*)
- SetServerCameraOverlay (*ICVVideo3*)
- GotoHomePreset (ICVVideo3)

The Disconnect method takes the CVVideo control out of the **Connection** context. The Playback method causes the CVVideo control to enter the **Playback** context, discussed next.

Playback Context

An application can sensibly invoke the methods associated with the **Playback** context only after a successful execution of the Playback method, and before any subsequent execution of the Disconnect or Connect method. In

the **Playback** context, there is an established connection to a camera, and a video playback interval has been established. In the **Playback** context the CVVideo control is oriented around playback of recorded video as opposed to viewing of live video.

Here is the full list of CVClientControl methods associated with the Playback context:

- EndPosition
- ExportClip
- ExportWizard
- PlaybackRate
- Position
- Seek
- Start
- StartPosition
- Stop

On entering the **Playback** context the video is paused at the first frame of the playback interval. The Start method initiates the actual playback. The Stop method pauses the playback and causes the CVVideo control to enter the **Paused** context.

The Connect method may be used to leave the **Playback** context. The Disconnect method may be used to leave the **Playback** context and **Connection** context.

Paused Context

An application can sensibly invoke the methods associated with the **Paused** context only after a successful execution of the Stop method, and before any subsequent execution of the Disconnect method or the Start method. In the **Paused** context, there is an established connection to a camera, an established playback interval, and the playback has been paused.

Here is the full list of CVClientControl methods associated with the **Paused** context:

- StepBack
- StepForward

The Start method causes the CVVideo control to leave the **Paused** context. The Connect method may be used to leave the **Paused** and **Playback** contexts. The Disconnect method may be used to leave the **Paused**, **Playback**, and **Connection** contexts.

Threading Model

CVClientControl is an apartment threaded COM server, so each CVVideo control instance must be used on only a single thread, and that thread must have a Windows message loop.

A few of the methods in the CVClientControl API perform a synchronous message exchange with a CompleteView server. These methods are specially designed with an embedded Windows message loop and a visible (modal) dialog box, providing user feedback and preventing “lockup” of the UI during abnormally lengthy operations. For many applications this usage pattern will be adequate. However, some application developers may want to consider putting the CVVideo control instance on a thread other than the main user interface thread. As mentioned above, such an alternate thread must have a message pump.

CVClientControl API Reference

ICVVideo

Application developers should not implement this interface. The CVVideo control provides this functionality.

Note: In C++, all properties are retrieved by a method whose name is formed by prepending *get_* to the name of the property. For example, if the property's name is *Password*, then *get_Password* retrieves the property. Similarly, writable properties are set by a method whose name is formed by prepending *put_* to the name of the property, e.g. *put_Password*.

The *ICVVideo* interface inherits from the IDispatch interface, and has additional properties and methods summarized in the following tables and described in detail in the text that follows.

Property	Description
Camera	Identifies the camera, as distinguished from the other cameras managed by the CompleteView server.
FPS	Frames per second in the streaming video.
HasAudio (read-only)	Indicates camera support for audio.
IsPTZ (read-only)	Indicates camera support for PTZ control.
PTZRepeat	Whether or not to periodically repeat the most recent PTZ command.
Password	Password with which to log on to the CompleteView server.
Quality	Quality of the streaming video.
Server	Identifies the CompleteView server.
ShowStatus	Determines whether status text is displayed when there is no video.
KeyframeRate	Key frames per second for streaming live video.
Username	User name with which to log on to the CompleteView server.

Method	Description	Context
AutoFocus	Turns on auto focus.	Connection
AutoIris	Turns on auto iris.	Connection
Connect	Initiates connection to a camera.	General (camera-oriented)
ConnectSync	Tests the accessibility of a server.	General (server-oriented)
ConnectSyncResult	Tests the accessibility of a server and provides one of several result codes.	General (server-oriented)
Disconnect	Tears down the current camera connection.	Connection
EnableAudio	Enables or disables audio.	Connection
EndPosition	Gets the end date/time of the current playback interval.	Playback
ExportClip	Exports the current playback interval to a file.	Playback
ExportWizard	Presents a user interface that guides the user through the exporting of a video clip.	Playback
Focus	Sets the camera's focus speed.	Connection
GetCameraInfo	Gets a camera name and status.	General (server-oriented)
GetFilePathName	Presents a standard file chooser dialog with JPEG (.JPG) as the primary, default file type.	General (truly general)
GetNumberOfCameras	Retrieves the number of cameras on the server.	General (server-oriented)
GetViewOption	Gets the current video viewing option.	General (truly general)
GotoPreset	Moves the camera to a preset location.	Connection
Iris	Sets the camera's iris speed.	Connection
Playback	Sets the video playback interval.	Connection
PlaybackRate	Adjusts the playback rate.	Playback
Position	Retrieves the current playback position.	Playback
SaveSnapshot	Saves the current video frame to a JPEG file.	Connection
SaveSnapshotScaled	Saves the current video frame to a JPEG file, with optional image scaling.	Connection
Seek	Seeks to a particular position in the current playback.	Playback
SetCameraOverlay	Changes the camera's overlay settings.	Connection
SetPreset	Sets a camera preset.	Connection
SetViewOption	Sets the current viewing option (stretching or letterbox).	General (truly general)
Start	Starts playback of recorded video.	Playback
StartPosition	Gets the start date/time of the current playback interval.	Playback

StepBack	Goes backward one video frame (while playback is paused).	Paused
StepForward	Goes forward one video frame (while playback is paused).	Paused
Stop	Pauses playback.	Playback

ICVVideo::AutoFocus

The *AutoFocus* method instructs the current camera to turn on auto focus, if this feature is supported by the camera.

Syntax (C++)

```
HRESULT AutoFocus(
);
```

Parameters

The AutoFocus method has no parameters.

Return Value

The return value is *S_OK*.

Remarks

AutoFocus is associated with the **Connection** context, i.e. it only makes sense to call it when there is a current camera.

AutoFocus is a best-effort method providing no indication of whether or when the camera actually executes the command.

ICVVideo::AutoIris

The *AutoIris* method instructs the current camera to turn on auto iris, if this feature is supported by the camera.

Syntax (C++)

```
HRESULT AutoIris(
);
```

Parameters

The AutoIris method has no parameters.

Return Value

The return value is *S_OK*.

Remarks

AutoIris is associated with the **Connection** context, i.e. it only makes sense to call it when there is a current camera.

AutoIris is a best-effort method providing no indication of whether or when the camera actually executes the command.

ICVVideo::Camera

The *Camera* property represents the camera identifier (a one-based index). This value determines the particular camera to connect to on the next connection attempt. It is necessary to set this property before calling Connect. Changing the value of this property has no effect on the connection after *Connect* has been called.

Syntax (C++)

```
HRESULT get_Camera(
   [out] SHORT *Result
);
HRESULT put_Camera(
   [in]  SHORT NewVal
);
```

Parameters

Result [out]

Receives the current value of the property.

NewVal [in]

Target camera identifier (one-based index) for the next connection attempt.

Return Value

The return value is *S_OK*.

ICVVideo::Connect

The *Connect* method initiates the process of connecting to the camera identified by the Camera property managed by the server identified by the Server property, using the logon credentials defined by the Username and Password properties.

Syntax (C++)

```
HRESULT Connect(
);
```

Parameters

The *Connect* method has no parameters.

Return Value

The return value is *S_OK*.

Remarks

The connection is established asynchronously. The CVVideo control will call either one of OnConnected or OnFailedConnection (in the *ICVVideoEvents* interface) to report the eventual outcome of the connection attempt.

Important: API access must be enabled on the server (for the user identified by the *UserName* property) for a connection to be made successfully.

If the CVVideo control was previously connected to a camera, that connection is torn down, followed by initiation of a new connection.

Connect is associated with the **General** context, meaning that it can be usefully called at any time. Note, however, that the successful establishment of a connection depends on the values of the *Server*, *Username*, *Password*, and *Camera* properties. On successful connection, the CVVideo control enters the **Connection** context.

ICVVideo::ConnectSync

The *ConnectSync* method attempts to synchronously connect to the server identified by the Server property, using credentials supplied by the Username and Password properties. When successful, the connection is immediately terminated, before this method returns.

Syntax (C++)

```
HRESULT ConnectSync(
   [out] VARIANT_BOOL *Result
);
```

Parameters

Result [out]

Receives a Boolean indicating the success (*VARIANT_TRUE*) or failure (*VARIANT_FALSE*) of the connection attempt.

Return Value

The return value is *S_OK*.

Remarks

Use *ConnectSync* to test whether a given server is accessible. Any existing connection established by the *Connect* method is unaffected.

Important: API access must be enabled on the server (for the user identified by the *UserName* property) for a connection to be made successfully.

This method does NOT cause the *OnConnected* or *OnFailedConnection* events to be fired. The returned Boolean is the only indication of success.

Windows messages will continue to be dispatched during the message exchange with the server.

ConnectSync is associated with the **General** context, meaning that it can be usefully called at any time. Note, however, that its successful (and meaningful) execution depends on the values of the *Server*, *Username*, and *Password* properties.

ICVVideo::ConnectSyncResult

The *ConnectSyncResult* method synchronously attempts to connect to the server identified by the Server property, using credentials supplied by the Username and Password properties. When successful, the connection is immediately terminated, before this method returns. This method returns more detailed connection status information than the ConnectSync method.

Syntax (C++)

```
HRESULT ConnectSync(
   [out] ConnectResults *Result
);
```

Parameters

Result [out]

Receives a ConnectResults value indicating the result of the connection attempt.

Return Value

The return value is *S_OK*.

Remarks

Use *ConnectSyncResult* to test whether a given server is accessible and when more than a Boolean success/failure indication is desired.

Any existing connection established by the *Connect* method is unaffected.

Important: API access must be enabled on the server (for the user identified by the *UserName* property) for a connection to be made successfully.

This method does NOT cause the *OnConnected* or *OnFailedConnection* events (*ICVVideoEvents*) to be fired. The returned *ConnectResults* value is the only indication of success.

Windows messages will continue to be dispatched during the message exchange with the server.

ConnectSyncResult is associated with the **General** context, meaning that it can be usefully called at any time. Note, however, that its successful (and meaningful) execution depends on the values of the *Server*, *Username*, and *Password* properties.

ICVVideo::Disconnect

The *Disconnect* method tears down the current connection to the camera.

Syntax (C++)

```
HRESULT Disconnect(
);
```

Parameters

The *Disconnect* method has no parameters.

Return Value

The return value is *S_OK*.

Remarks

Disconnect takes down the current connection, if one has been established. If a connection has been initiated (via *Connect*) but not completed, then that connection attempt is aborted.

Disconnect is associated with the **Connection** context, i.e. it only makes sense to call it when there is a current camera. Disconnect causes the CVVideo control to leave the **Connection** context.

ICVVideo::EnableAudio

The *EnableAudio* method instructs the current camera to enable or disable audio.

Syntax (C++)

```
HRESULT EnableAudio(
```

```
    [in]  VARIANT_BOOL Enable,
    [out] VARIANT_BOOL *Result
);
```

Parameters

Enable [in]

Boolean indicating whether to enable (*VARIANT_TRUE*) or disable (*VARIANT_FALSE*) audio.

Result [out]

Receives a Boolean indicating the success or failure of the operation. See the remarks below.

Return Value

The return value is *S_OK*.

Remarks

On output, Result is set to *VARIANT_FALSE* if there is no current camera, a problem is detected with the connection, or the current camera does not support audio. Otherwise it is set to *VARIANT_TRUE*. A value of *VARIANT_TRUE* provides no guarantee that the camera acted (or will act) upon the command.

Note: Audio is enabled or disabled only with respect to the CVVideo control instance. It is essentially a "mute" control for the CVVideo control. Recording of audio is unaffected.

EnableAudio is associated with the **Connection** context, i.e. it only makes sense to call it when there is a current camera.

EnableAudio nominally conducts a synchronous message exchange with the server. During this message exchange Windows messages will continue to be dispatched.

ICVVideo::EndPosition

The *EndPosition* method returns the end date/time of the current playback interval.

Syntax (C++)

```
HRESULT EndPosition(
    [out] DATE *Value
);
```

Parameters

Value [out]

Receives the end date/time, or zero if no playback interval has been established.

Return Value

The return value is *S_OK*.

Remarks

EndPosition is associated with the **Playback** context, i.e. it can only be usefully invoked after a playback interval has been established.

ICVVideo::ExportClip

The *ExportClip* method exports the current playback interval to a file.

Syntax (C++)

```
HRESULT ExportClip(
   [in]  BSTR FilePathName)
);
```

Parameters

FilePathName [in]

Path to the file to which the video clip should be exported.

Return Value

The return value is *S_OK*.

Remarks

ExportClip displays a (modal) dialog box that provides feedback on the progress of the operation. If the client application wants programmatic progress information, it may listen for the OnExportProgress event.

ExportClip is associated with the **Playback** context, i.e. it can only be usefully invoked after a playback interval has been established.

ExportClip is a best-effort method providing no indication of whether or when the clip is actually exported. Reasons for failure include there being no current camera or no current playback clip, an invalid file path, or an inability to write to the specified file.

ICVVideo::ExportWizard

The *ExportWizard* method presents a user interface that allows the user to choose part (or all) of the current playback interval for export, and a file name to which it should be exported. *ExportWizard* then exports the chosen video to the specified file.

Syntax (C++)

```
HRESULT ExportWizard(
);
```

Parameters

The *ExportWizard* method has no parameters.

Return Value

The return value is *S_OK*.

Remarks

If the client application wants programmatic progress information it may listen for the OnExportProgress event.

ExportWizard is associated with the **Playback** context, i.e. it can only be usefully invoked after a playback interval has been established.

ExportWizard is a best-effort method providing no indication of whether any errors occurred in the export. Reasons for failure include there being no current camera or no current playback clip, or an inability to write to the chosen file.

ICVVideo::FPS

The *FPS* property determines the frames per second of the video stream associated with the next camera connection. Changing the value of this property does not affect the current connection (if any). The default value is 4.0.

Syntax (C++)

```
HRESULT get_FPS(
   [out] DOUBLE *Result
);
HRESULT put_FPS(
   [in] DOUBLE NewVal
);
```

Parameters

Result [out]

Receives the current value of the property.

NewVal [in]

The frames per second to use with the next camera connection.

Return Value

The return value is *S_OK*.

ICVVideo::Focus

The *Focus* method instructs the current camera to set its focus speed, if the camera supports this feature.

Syntax (C++)

```
HRESULT Focus(
   [in]  SHORT FocusSpeed
);
```

Parameters

FocusSpeed [in]

Focus speed, the units of which are camera-dependent, but the range of which is always [-1000, +1000]. (The value is clamped into this range.)

Return Value

The return value is *S_OK*.

Remarks

Focus is associated with the **Connection** context, i.e. it only makes sense to call it when there is a current camera.

Focus is a best-effort method providing no indication of whether or when the camera actually executes the command.

ICVVideo::GetCameraInfo

The *GetCameraInfo* method retrieves information about a particular camera administered by a particular server.

Syntax (C++)

```
HRESULT GetCameraInfo(
```

```
    [in]  SHORT Camera,
    [out] BSTR *Name,
    [out] UINT *Status
);
```

Parameters

Camera [in]

Camera identifier (one-based index).

Name [out]

Receives a camera name, or is set to the empty string if an error occurs.

Status [out]

Receives a camera status, or is set to zero if an error occurs.

Return Value

The return value is *S_OK*.

Remarks

GetCameraInfo is associated with the **General** context, meaning that it may be usefully called at any time. However, its behavior is essentially unreliable when the CVVideo control is in the **Connection** context, so client applications should not call this method while there is a current camera.

When not in the **Connection** context, this method uses the current values of the Server, Username, and Password properties. Note, however, that the returned camera name and status are not necessarily current. The CVVideo control caches this camera information, and the cache is only cleared by (a) calling Disconnect or Connect or (b) resetting the *Server* property (even to the same value).

The camera status is a bit field with the following status bits:

Bit Position	Meaning when Set
0 (least significant)	Scheduled recording is active.
1	External alarm detected, external alarm recording is active.
2	Motion recording is active.
3	Motion has been detected.
4	Camera is a PTZ camera.
5	Prealarm recording is active.
6	Not connected to server (video acquisition failure).
7	Tour is enabled.
8	Camera is inactive/stopped (no attempt by server to acquire video), though it may be enabled.
9	Camera supports audio.
10	Camera is an IP camera.
11	Camera is an ACTI model.
13	Camera is transitioning to or from the stopped state. Bit 8 is the intended next state (0 = started, 1 = stopped), but the camera is not there yet.
14	Camera is set to use Digital PTZ.
15	Recording failure.
16	Camera preset and/or preset name changed.

GetCameraInfo may conduct a synchronous message exchange with the server. During this message exchange, Windows messages will continue to be dispatched.

ICVVideo::GetFilePathName

The *GetFilePathName* method presents a standard file chooser dialog with JPEG (.JPG) as the default file type, and returns the chosen file path (if any).

Syntax (C++)

```
HRESULT GetFilePathName(
   [in]  BSTR FileTypes,
   [out] BSTR *File
);
```

Parameters

FileTypes [in]

A file filter specification such as that used in the OPENFILENAME structure of the Win32 API.

File [out]

Receives the path to the chosen file, or the empty string if the user cancels the dialog.

Return Value

The return value is *S_OK*.

Remarks

GetFilePathName is associated with the **General** context, meaning that it can be usefully called at any time.

ICVVideo::GetNumberOfCameras

The *GetNumberOfCameras* method retrieves the number of cameras on a particular server.

Syntax (C++)

```
HRESULT GetNumberOfCameras(
   [out] SHORT *Result
);
```

Parameters

Result [out]

Receives the number of cameras, or is set to zero in the case of an error. See the remarks below.

Return Value

The return value is *S_OK*.

Remarks

GetNumberOfCameras is associated with the **General** context, meaning that it may be usefully called at any time. However, its behavior is essentially unreliable when the CVVideo control is in the **Connection** context, so client applications should not call this method while there is a current camera.

When not in the **Connection** context, this method uses the current values of the Server, Username, and Password properties. Note, however, that the returned number of cameras is not necessarily current. The CVVideo control

caches this camera information, and the cache is only cleared by (a) calling Disconnect or Connect or (b) resetting the *Server* property (even to the same value).

GetNumberOfCameras may conduct a synchronous message exchange with the server. During this message exchange, Windows messages will continue to be dispatched.

ICVVideo::GetViewOption

The *GetViewOption* method returns the current viewing option (letterbox or stretch to fit).

Syntax (C++)

```
HRESULT GetViewOption(
   [out] ViewOptions *Option
);
```

Parameters

Option [out]

Receives the current viewing option. See ViewOptions.

Return Value

The return value is *S_OK*.

Remarks

GetViewOption is associated with the **General** context, meaning that it can be usefully called at any time.

ICVVideo::GotoPreset

The *GotoPreset* method instructs the current camera to move (pan, tilt, zoom, focus, and/or iris) to a preset location.

Syntax (C++)

```
HRESULT GotoPreset(
   [in]  SHORT Preset
);
```

Parameters

Preset [in]

Camera preset number.

Return Value

The return value is *S_OK*.

Remarks

GotoPreset is associated with the **Connection** context, i.e. it only makes sense to call it when there is a current camera.

GotoPreset is a best-effort method providing no indication of whether or when the camera actually executes the command.

ICVVideo::HasAudio

The read-only *HasAudio* property indicates whether the current camera supports audio. The value of this property is *VARIANT_FALSE* when there is no current camera.

Syntax (C++)

```
HRESULT get_HasAudio(
   [out] VARIANT_BOOL *Result
);
```

Parameters

Result [out]

Receives the current value of the property.

Return Value

The return value is *S_OK*.

ICVVideo::Iris

The *Iris* method instructs the camera to set its iris speed, if the camera supports this feature.

Syntax (C++)

```
HRESULT Iris(
   [in]  SHORT IrisSpeed
);
```

Parameters

IrisSpeed [in]

Iris speed, the units of which are camera-dependent, but the range of which is always [-1000, +1000]. (The value is clamped to this range.)

Return Value

The return value is *S_OK*.

Remarks

Iris is associated with the **Connection** context, i.e. it only makes sense to call it when there is a current camera.

Iris is a best-effort method providing no indication of whether or when the camera actually executes the command.

ICVVideo::IsPTZ

The read-only *IsPTZ* property indicates whether the current camera supports pan-tilt-zoom control. The value of this property is VARIANT_FALSE if there is no current camera.

Syntax (C++)

```
HRESULT get_IsPTZ(
   [out] VARIANT_BOOL *Result
);
```

Parameters

Result [out]

Receives the current value of the property.

Return Value

The return value is *S_OK*.

ICVVideo::PTZ

The *PTZ* method instructs the current camera to set its pan-tilt-zoom speeds, if these features are supported by the camera.

Syntax (C++)

```
HRESULT PTZ(
   [in]  SHORT PanSpeed,
   [in]  SHORT TiltSpeed,
   [in]  SHORT ZoomSpeed
);
```

Parameters

PanSpeed [in]

Pan speed, the units of which are camera-dependent, but the range of which is always [-1000, +1000]. (The value is clamped to this range.)

TiltSpeed [in]

Tilt speed, the units of which are camera-dependent, but the range of which is always [-1000, +1000]. (The value is clamped to this range.)

ZoomSpeed [in]

Zoom speed, the units of which are camera-dependent, but the range of which is always [-1000, +1000]. (The value is clamped to this range.)

Return Value

The return value is *S_OK*.

Remarks

PTZ is associated with the **Connection** context, i.e. it only makes sense to call it when there is a current camera.

PTZ is a best-effort method providing no indication of whether or when the camera actually executes the command.

ICVVideo::PTZRepeat

The *PTZRepeat* property determines whether the CVVideo control will repeat the most recent PTZ command once per second.

Syntax (C++)

```
HRESULT get_PTZRepeat(
   [out] VARIANT_BOOL *Result
);
HRESULT put_PTZRepeat(
   [in]  VARIANT_BOOL NewVal
);
```

Parameters

Result [out]

Receives the current value of the property.

NewVal [in]

Specifies whether to enable (*VARIANT_TRUE*) or disable (*VARIANT_FALSE*) the PTZ repetition feature.

Return Value

The return value is *S_OK*.

ICVVideo::Password

The *Password* property determines the user password to use in the next connection to a server/camera. Changing this property has no effect on the current connection, if any.

Syntax (C++)

```
HRESULT get_Password(
   [out] BSTR *Result
);
HRESULT put_Password(
   [in]  BSTR NewVal
);
```

Parameters

Result [out]

Receives the value of the property.

NewVal [in]

The user password to use in the next camera connection.

Return Value

The return value is *S_OK*.

ICVVideo::Playback

The *Playback* method sets up the current server (and/or current camera) for playback of video captured by the current camera.

Syntax (C++)

```
HRESULT Playback(
   [in]  DATE Start,
   [in]  DATE End,
   [out] VARIANT_BOOL *Result
);
```

Parameters

Start [in]

The start date/time of the video to play back, in server local time. This parameter defines the beginning of the playback interval.

End [in]

The end date/time of the video to play back, in server local time. This parameter defines the end of the playback interval.

Result [out]

Not used.

Return Value

The return value is *S_OK*.

Remarks

Playback is associated with the **Connection** context, i.e. it only makes sense to call it when there is a current camera. A successful execution of this method takes the CVVideo control into the **Playback** context.

Playback is a best-effort method providing no indication of whether or when the server/camera actually executes the command.

If *Start* and *End* are not in server local time, specify the time zone by calling *SetTimeZone()* method.

ICVVideo::PlaybackRate

The *PlaybackRate* method instructs the server and/or current camera to adjust the rate of playback.

Syntax (C++)

```
HRESULT PlaybackRate(
   [in]  DOUBLE Rate
);
```

Parameters

Rate [in]

Playback rate, where 1.0 is normal speed, 2.0 is double the speed, etc. Positive values less than 1.0 are slower than normal speed. This value can't be negative or zero.

Return Value

The return value is *S_OK*.

Remarks

PlaybackRate is associated with the **Playback** context, i.e. it can only be usefully called after a playback interval has been established.

PlaybackRate is a best-effort method providing no indication of whether or when the server/camera actually executes the command.

ICVVideo::Position

The *Position* method retrieves the current position (as a date/time) in the current playback interval.

Syntax (C++)

```
HRESULT Position(
   [out] DATE *CurrentPos
);
```

Parameters

CurrentPos [out]

Receives the current playback position, or a time in the distant past if an error occurs. See the remarks below.

Return Value
The return value is *S_OK*.

Remarks
The returned *DATE* will be in the distant past if there is no current playback or an error occurs.

Position is associated with the **Playback** context, i.e. it only makes sense to call it after a playback interval has been established.

ICVVideo::Quality
The *Quality* property determines the video streaming quality. This is a technical parameter of the video encoding. The range is zero (worst quality) to 100 (best quality). The default value is 100. The primary benefit of lowering this value is reduced network bandwidth usage. Note that the BPS property (in the *ICVVideo2* interface) provides a different, more easily quantifiable means to this end.

This property only affects subsequent camera connections. Changing the value of this property has no effect on the current connection, if any.

Syntax (C++)
```
HRESULT get_Quality(
   [out] SHORT *Result
);
HRESULT put_Quality(
   [in]  SHORT NewVal
);
```

Parameters
Result [out]
Receives the current value of the property.

NewVal [in]
The video streaming quality to use in the next camera connection.

Return Value
The return value is *S_OK*.

ICVVideo::SaveSnapshot
The *SaveSnapshot* method saves the current video frame to a file as a JPEG image.

Syntax (C++)
```
HRESULT SaveSnapshot(
   [in]  BSTR Filename,
   [out] LONG *Result
);
```

Parameters
Filename [in]
The path to the output file.

Result [out]

Receives a CVVideo-internal status code indicating the success or failure of the operation. This value is zero if successful, nonzero if an error occurs.

Return Value

The return value is *S_OK*.

Remarks

The SaveSnapshotScaled method is similar, but allows the client application to specify the pixel resolution of the target image.

SaveSnapshot is associated with the **Connection** context, i.e. it only makes sense to call it when there is a current camera.

ICVVideo::SaveSnapshotScaled

The *SaveSnapshotScaled* method saves the current video frame to a file as a JPEG image, with (optional) scaling of the image.

Syntax (C++)

```
HRESULT SaveSnapshotScaled(
   [in]  BSTR Filename,
   [in]  SHORT Width,
   [in]  SHORT Height,
   [out] LONG *Result
);
```

Parameters

Filename [in]

Path to the target file.

Width [in]

Image width in pixels. Specify -1 for the default.

Height [in]

Image height in pixels. Specify -1 for the default.

Result [out]

Receives a CVVideo-internal status code indicating the success or failure of the operation. This value is zero if successful, nonzero if an error occurs.

Return Value

The return value is *S_OK*.

Remarks

Note: Calling this method with -1 for both the width and height is the same as calling the SaveSnapshot method.

SaveSnapshotScaled is associated with the **Connection** context, i.e. it only makes sense to call it when there is a current camera.

ICVVideo::Seek

The *Seek* method instructs the server (and/or current camera) to seek to a particular position in the current playback interval.

Syntax (C++)

```
HRESULT Seek(
   [in]  DATE NewPos
);
```

Parameters

NewPos [in]

The target date/time, in server local time.

Return Value

The return value is *S_OK*.

Remarks

Seek is associated with the **Playback** context, meaning that it can only be called usefully after a playback interval has been established.

Seek is a best-effort method providing no indication of whether or when the command is actually executed by the server/camera.

If *SetTimeZone()* was called prior to making a playback request, *NewPos* should be in the same time zone format.

ICVVideo::Server

The *Server* property determines which CompleteView server is used in the next camera connection. This value should be a host name, or an IP address in dotted-quad notation, or the string *localhost*. This property only affects subsequent camera connections. Changing the value of this property has no effect on the current connection, if any.

Syntax (C++)

```
HRESULT get_Server(
   [out] BSTR *Result
);
HRESULT put_Server(
   [in]  BSTR NewVal
);
```

Parameters

Result [out]

Receives the current value of the property.

NewVal [in]

The next server to connect to.

Return Value

The return value is *S_OK*.

ICVVideo::SetCameraOverlay

The *SetCameraOverlay* method instructs the server (and/or current camera) to adjust the overlay settings for the current camera.

Syntax (C++)

```
HRESULT SetCameraOverlay(
   [in]  SHORT CameraID,
   [in]  BSTR Overlay,
   [in]  VARIANT_BOOL ShowTimestamp,
   [in]  VARIANT_BOOL ShowCameraName
);
```

Parameters

CameraID [in]

Not used.

Overlay [in]

Overlay text.

ShowTimestamp [in]

Whether to show (*VARIANT_TRUE*) or hide (*VARIANT_FALSE*) the timestamp in the overlay.

ShowCameraName [in]

Whether to show (*VARIANT_TRUE*) or hide (*VARIANT_FALE*) the camera name in the overlay.

Return Value

The return value is *S_OK*.

Remarks

SetCameraOverlay is associated with the **Connection** context, i.e. it only makes sense to call it when there is a current camera.

SetCameraOverlay is a best-effort method providing no indication of whether or when the command is actually executed by the server/camera.

ICVVideo::SetPreset

The *SetPreset* method instructs the server (and/or current camera) to set a particular preset on the current camera. The preset become's associated with the camera's current "position" (pan, tilt, zoom, focus, and/or iris).

Syntax (C++)

```
HRESULT SetPreset(
   [in]  SHORT Preset
);
```

Parameters

Preset [in]

Camera preset number, this value is zero-based

Return Value

The return value is *S_OK*.

Remarks

SetPreset is associated with the **Connection** context, i.e. it only makes sense to call it when there is a current camera.

SetPreset is a best-effort method providing no indication of whether or when the server/camera actually executes the command.

ICVVideo::SetViewOption

The *SetViewOption* method sets the current viewing option (stretching or letterbox).

Syntax (C++)

```
HRESULT SetViewOption(
   [in]  ViewOptions Option
);
```

Parameters

Option [in]

New viewing option.

Return Value

The return value is *S_OK*.

Remarks

SetViewOption is associated with the **General** context, meaning that it can be usefully invoked at any time.

ICVVideo::ShowStatus

The *ShowStatus* property determines whether status text is shown when there is no video to display. The default value is *VARIANT_TRUE*.

Syntax (C++)

```
HRESULT get_ShowStatus(
   [out] VARIANT_BOOL *Result
);
HRESULT put_ShowStatus(
   [in]  VARIANT_BOOL NewVal
);
```

Parameters

Result [out]

Receives the current value of the property.

NewVal [in]

Whether to show (*VARIANT_TRUE*) or hide *(VARIANT_FALSE*) status text.

Return Value

The return value is *S_OK*.

ICVVideo::Start

The *Start* method instructs the server (and/or current camera) to start playback.

Syntax (C++)

```
HRESULT Start(
);
```

Parameters

The *Start* method has no parameters.

Return Value

The return value is *S_OK*.

Remarks

Start is associated with the **Playback** context, i.e. it only makes sense to call it after a playback interval has been established. When the CVVideo control is in the **Paused** context, a successful invocation of *Start* causes it to leave that context.

Start is a best-effort method providing no indication of whether or when the server/camera actually executes the command.

ICVVideo::StartPosition

The *StartPosition* method returns the start of the current playback interval.

Syntax (C++)

```
HRESULT StartPosition(
   [out] DATE *Val
);
```

Parameters

Val [out]

Receives the playback start position, in server local time or specified time zone.

Return Value

The return value is *S_OK*.

Remarks

StartPosition is associated with the **Playback** context, meaning that it only makes sense to call it after a playback interval has been established.

ICVVideo::StepBack

The *StepBack* method instructs the server (and/or current camera) to advance the video backward by one frame.

Syntax (C++)

```
HRESULT StepBack(
);
```

Parameters

The *StepBack* method has no parameters.

Return Value

The return value is *S_OK*.

Remarks

StepBack is associated with the **Paused** context, meaning that it only makes sense to call it while playback is paused.

StepBack is a best-effort method giving no indication of whether or when the server/camera actually executes the command.

ICVVideo::StepForward

The *StepForward* method instructs the server (and/or current camera) to advance the video forward by one frame.

Syntax (C++)

```
HRESULT StepForward(
);
```

Parameters

The *StepForward* method has no parameters.

Return Value

The return value is *S_OK*.

Remarks

StepForward is associated with the **Paused** context, meaning that it only makes sense to call it while playback is paused.

StepForward is a best-effort method giving no indication of whether or when the server/camera actually executes the command.

ICVVideo::Stop

The *Stop* method instructs the server (and/or current camera) to suspend (pause) playback.

Syntax (C++)

```
HRESULT Stop(
);
```

Parameters

The *Stop* method has no parameters.

Return Value

The return value is *S_OK*.

Remarks

Stop is associated with the **Playback** context, i.e. it only makes sense to call it after a playback interval has been established. Furthermore, *Stop* will have no effect if playback has not been started via Start.

A successful execution of *Stop* takes the CVVideo control into the **Paused** context.

Stop is a best-effort method providing no indication of whether or when the server/camera actually executes the command.

ICVVideo::Username

The *Username* property determines the user name to use in the next server/camera connection. This property only affects subsequent connections. Changing the value of this property has no effect on the current camera connection, if any.

Syntax (C++)

```
HRESULT get_Username(
   [out] BSTR *Result
);
HRESULT put_Username(
   [in]  BSTR NewVal
);
```

Parameters

Result [out]

Receives the current value of the property.

NewVal [in]

The user name to use in the next camera connection.

Return Value

The return value is *S_OK*.

ICVVideo2

Application developers should not implement this interface. The CVVideo control provides this functionality.

Note: In C++, all properties are retrieved by a method that is called by prepending *get_* to the name of the property. For example, if the property's name is *Password*, then *get_Password* retrieves the property. Similarly, writable properties are set by a method whose name is formed by prepending *put_* to the name of the property, e.g. *put_Password*.

The *ICVVideo2* interface inherits from the ICVVideo interface, and has additional properties and methods summarized in the following tables and described in the text that follows.

Property	Description
BPS	Maximum video streaming bitrate.
XResolution	Video width.
YResolution	Video height.
IsDigitalPTZ (read-only)	Indicates whether camera is set to use Digital PTZ.

Method	Description
GetServerTime	Retrieves the server's system time.
GetServerTimeZone	Retrieves information about the server's time zone.
GetSnapshot	Writes a snapshot to an image buffer.
GetSnapshotScaled	Writes a snapshot to an image buffer, with optional image scaling.
SearchVideo	Retrieves information about the time intervals for which video exists.
SetTimeZone	Sets the time zone for playback requests.
CheckFeature	Checks if a particular product feature or version is supported.
SilentExport	Exports the current playback interval to a file in AVI format without displaying the export dialog.
CancelSilentExport	Cancels the video export operation started by *SilentExport* API. Deletes the partially exported video.

ICVVideo2::BPS

The *BPS* property determines the maximum bitrate for the video streaming associated with the next camera connection. The default value is -1, meaning that the maximum bitrate is left up to the implementation. This property only affects subsequent connections. Changing the value of this property has no effect on the current camera connection, if any.

Syntax (C++)

```
HRESULT get_BPS(
   [out] LONG *Result
);
HRESULT put_BPS(
   [in]  LONG NewVal
);
```

Parameters

Result [out]

Receives the current value of the property.

NewVal [in]

The target bitrate for the next camera connection. Specify any non-positive integer to revert to the implementation's default.

Return Value

The return value is *S_OK*.

ICVVideo2::GetServerTime

The *GetServerTime* method retrieves the system time on the server identified by the Server property, using credentials supplied by the Username and Password properties.

Syntax (C++)

```
HRESULT GetServerTime(
   [out] DATE *CurrentTimeLocal,
   [out] DATE *CurrentTimeUTC,
   [out] VARIANT_BOOL *Result
```

```
);
```

Parameters

CurrentTimeLocal [out]

Receives the server's local time.

CurrentTimeUTC [out]

Receives the server's time in UTC.

Result [out]

Receives a Boolean value indicating success (*VARIANT_TRUE*) or failure (*VARIANT_FALSE*) in retrieving the server time.

Return Value

The return value is *S_OK*.

Remarks

GetServerTime is associated with the **General** context, meaning that it can be usefully called at any time. Note, however, that its successful (and meaningful) execution depends on the values of the *Server*, *Username*, and *Password* properties.

GetServerTime normally conducts a synchronous message exchange with the server. During this exchange, Windows messages will continue to be dispatched.

ICVVideo2::GetServerTimeZone

The *GetServerTimeZone* method retrieves information about the time zone of the server identified by the Server property, using credentials supplied by the Username and Password properties..

Syntax (C++)

```
HRESULT GetServerTimeZone(
    [out] LONG *OffsetSeconds,
    [out] BSTR *Name,
    [out] VARIANT_BOOL *Result
);
```

Parameters

OffsetSeconds [out]

Receives the server's time zone offset from UTC, in seconds.

Name [out]

Receives the server's time zone name (see Remarks).

Result [out]

Receives a Boolean value indicating success (*VARIANT_TRUE*) or failure (*VARIANT_FALSE*) in retrieving the server time zone information.

Return Value

The return value is *S_OK*.

Remarks

GetServerTimeZone is associated with the **General** context, meaning that it can be usefully called at any time. Note, however, that its successful (and meaningful) execution depends on the values of the *Server*, *Username*, and *Password* properties.

GetServerTimeZone normally conducts a synchronous message exchange with the server. During this exchange, Windows messages will continue to be dispatched.

NOTE: The value of the *Name* output will be in the Windows display language in use on the Recording Server system. When the Recording Server runs as a Windows Service, the language for the SYSTEM account is used.

ICVVideo2::GetSnapshot

The *GetSnapshot* method writes the current video frame into a buffer.

Syntax (C++)

```
HRESULT GetSnapshot(
   [in]  BSTR Filename,
   [out] VARIANT *Buffer,
   [out] LONG *Result
);
```

Parameters

Filename [in]

Reserved. Must be *NULL*.

Buffer [out]

Receives the snapshot image, as an array of bytes (*VT_SAFEARRAY* | *VT_UI1*).

Result [out]

Receives a result code indicating the success or failure of the operation. A value of zero indicates success. Any other value is an internal implementation-defined error code and indicates failure.

Return Value

The return value is *S_OK*.

Remarks

GetSnapshot is associated with the **Connection** context, meaning that it only makes sense to call it while there is a current camera.

ICVVideo2::GetSnapshotScaled

The *GetSnapshotScaled* method writes the current video frame into a buffer, with optional scaling of the image.

Syntax (C++)

```
HRESULT GetSnapshotScaled(
   [in]  BSTR Filename,
   [in]  SHORT Width,
   [in]  SHORT Height,
   [out] VARIANT *Buffer,
   [out] LONG *Result
);
```

Parameters

Filename [in]

Reserved. Must be *NULL*.

Width [in]

Image width in pixels. Specify -1 for the default image width.

Height [in]

Image height in pixels. Specify -1 for the default image height.

Buffer [out]

Receives the snapshot image, as an array of bytes (*VT_SAFEARRAY* | *VT_UI1*).

Result [out]

Receives a result code indicating the success or failure of the operation. A value of zero indicates success. Any other value is an internal implementation-defined error code and indicates failure.

Return Value

The return value is *S_OK*.

Remarks

Calling this method with width and height both equal to -1 is the same as calling GetSnapshot.

GetSnapshotScaled is associated with the **Connection** context, meaning that it only makes sense to call it while there is a current camera.

ICVVideo2::SearchVideo

The *SearchVideo* method returns an indication of the times for which video exists from the current camera during a particular time interval.

Syntax (C++)

```
HRESULT SearchVideo(
   [in]  DATE Start,
   [in]  DATE End,
   [out] VARIANT *ClipStart,
   [out] VARIANT *ClipEnd,
   [out] VARIANT_BOOL *Result
);
```

Parameters

Start [in]

A date/time value giving the beginning of the temporal range to search, in client local time.

End [in]

A data/time value giving the end of the temporal range to search, in client local time.

ClipStart [out]

Receives a safe array of *DATE* (*VT_ARRAY* | *VT_DATE*). This array contains the start timestamps for each "clip" found by the search. The *i*th element of the array contains the start time of a clip, whose end time is the *i*th element of *ClipEnd*.

ClipEnd [out]

Receives a safe array of *DATE* (*VT_ARRAY* | *VT_DATE*). This array contains the end timestamps for each "clip" found by the search. The *i*th element of the array contains the end time of the clip whose start time is the *i*th element of *ClipStart*.

Result [out]

Receives a Boolean value indicating whether the search was successful. Reasons for failure include not being connected to a camera, or a problem with the connection. Note that a search returning no clips (i.e. returning empty *ClipStart* and *ClipEnd* arrays) may still have been a successful search.

Return Value

The return value is *S_OK*.

Remarks

SearchVideo returns time intervals indicating the presence of video. More precisely, it returns a sequence of *n* time intervals $[s_1, t_1]$, $[s_2, t_2]$, … $[s_n, t_n]$ such that

$$s_i \le t_i \text{ for each } i \in \{1, \dots n\}, \text{ i.e. each interval is non-empty;}$$

$$t_i < s_{i+1} \text{ for each } i \in \{1, \dots, n-1\}, \text{ i.e. the intervals are sorted and have non-empty spacing;}$$

$$s_1 \ge Start \text{ and } t_n \le End; \text{ i.e. the results are limited to the requested range;}$$

$$\text{there exists video from the current camera for all } t \in \bigcup_{i=1}^{n} [s_i, t_i]; \text{ and}$$

$$\text{if } t \in [Start, End] \cap \sim \bigcup_{i=1}^{n} [s_i, t_i] \text{, then the current camera has no video for time } t.$$

All times returned in *ClipStart* and *ClipEnd* are in client local time. If *Start* and *End* are not in client local time, specify the time zone by calling *SetTimeZone()* method in which case, *ClipStart* and *ClipEnd* will be in the same time zone as *Start* and *End*.

SearchVideo is associated with the **Connection** context, meaning that it only makes sense to call it while there is a current camera.

SearchVideo normally performs a synchronous message exchange with the server. During this exchange Windows messages will continue to be dispatched.

ICVVideo2::XResolution

The *XResolution* property determines the x-resolution (video width) to use in the next camera connection. The default value is -1, meaning the default width. This property only affects subsequent connections. Changing the value of this property does not affect the current camera connection, if any.

Syntax (C++)

```
HRESULT get_XResolution(
   [out] LONG *Result
);
HRESULT put_XResolution(
   [in]  LONG NewVal
);
```

Parameters

Result [out]

Receives the current value of the property.

NewVal [in]

The x-resolution (video width) to use in the next camera connection.

Return Value

The return value is *S_OK*.

ICVVideo2::YResolution

The *YResolution* property determines the y-resolution (video height) to use in the next camera connection. The default value is -1, meaning the default height. This property only affects subsequent connections. Changing the value of this property does not affect the current camera connection, if any.

Syntax (C++)

```
HRESULT get_YResolution(
   [out] LONG *Result
);
HRESULT put_YResolution(
   [in]  LONG NewVal
);
```

Parameters

Result [out]

Receives the current value of the property.

NewVal [in]

The y-resolution (video height) to use in the next camera connection.

Return Value

The return value is *S_OK*.

ICVVideo2::IsDigitalPTZ

The read-only *IsDigitalPTZ* property indicates whether the current camera supports digital PTZ protocol. The value of this property is VARIANT_FALSE if there is no current camera.

Syntax (C++)

```
HRESULT get_IsDigitalPTZ(
   [out] VARIANT_BOOL *Result
);
```

Parameters

Result [out]

Receives the current value of the property.

Return Value

The return value is *S_OK*.

ICVVideo2:SetTimeZone

The *SetTimeZone* method sets the time zone from which all playback requests will be made.

Syntax (C++)

```
HRESULT SetTimeZone (
   [in] TimeZoneType TimeZoneType
);
```

Parameters

TimeZoneType [in]

Time zone type. See TimeZoneType.

Return Value

The return value is *S_OK*.

Remarks

SetTimeZone is associated with the **General** context, meaning that it can be usefully invoked at any time but prior to making a playback request.

ICVVideo2::CheckFeature

The *SilentExport* method exports the current playback interval to a file.

Syntax (C++)

```
HRESULT CheckFeature(
   [in]  BSTR FeatureKeyString
   [out] VARIANT_BOOL* Result
);
```

Parameters

FeatureKeyString [in]

A particular product feature or version of the CompleteView server.

Result [out]

Receives VARIANT_FALSE if the requested feature is not present, otherwise VARIANT_TRUE.

Return Value

The return value is *S_OK*.

Remarks

The API should be used after the connection to the CompleteView server has been established successfully.

ICVVideo2::SilentExport

The *SilentExport* method exports the current playback interval to a file.

Syntax (C++)

```
HRESULT SilentExport(
   [in]  BSTR FilePathName
   [out] VARIANT_BOOL* Result
);
```

Parameters

FilePathName [in]

Path to the file to which the video clip should be exported.

Result [out]

Receives VARIANT_FALSE if the export operation fails immediately due to no camera, no video, invalid export interval or empty *FilePathName* parameter.

Return Value

The return value is *S_OK*.

Remarks

SilentExport does not display a dialog box that provides the feedback on the progress of the export operation. However, if the client application wants programmatic progress information, it may listen for the OnExportProgress event.

SilentExport is associated with the **Playback** context, i.e. it can only be usefully invoked after a playback interval has been established.

SilentExport is a best-effort method providing no indication of whether or when the clip is actually exported. Reasons for failure include there being no current camera or no current playback clip, an invalid file path, or an inability to write to the specified file.

ICVVideo2::CancelSilentExport

The *CancelSilentExport* method cancels the export operation initiated by client using the *SilentExport* API.

Syntax (C++)

```
HRESULT CancelSilentExport();
```

Parameters

None.

Return Value

The return value is *S_OK*.

Remarks

The CancelSilentExport API terminates the export operation which is started using *SilentExport* API only. This API should not be used to cancel the export operation, which is started using *ExportClip* or *ExportWizard*.

CancelSilentExport is innocuous and will have no effect if used without calling *SilentExport.* This API is associated with the **Playback** context.

If the export operation initiated using the *SilentExport* is not completed, *CancelSilentExport* will delete the partially exported video clip.

ICVVideo3

Application developers should not implement this interface. The CVVideo control provides this functionality.

Note: In C++, all properties are retrieved by a method that is called by prepending *get_* to the name of the property. For example, if the property's name is *Password*, then *get_Password* retrieves the property. Similarly, writable properties are set by a method whose name is formed by prepending *put_* to the name of the property, e.g. *put_Password*.

The *ICVVideo3* interface inherits from the ICVVideo2 interface, and has additional properties and methods summarized in the following tables and described in the text that follows:

Property	Description
PtzSupports (read-only)	PTZ capabilities that this camera supports.

Method	Description
ProcessIFrameOnly	Enables the ActiveX control to process I-Frames only.
ForceToSeekTime	Enables playback from a desired position within specified playback interval.
SetTimeZoneName	Allows the API clients to specify custom timezone name.
SetDewarpOptions	Allows the API clients to specify dewarp/fisheye options pertaining to lens such as orientation, profile, type, co-ordinates of center, radius and any data pertaining to curvature.
SetLensDewarpType	Allows the API clients to specify type of fisheye view for the video renderer.
SetPresetName	Allows the API clients to specify a custom preset name.
GetPresetNames	Allows API clients to retrieve existing preset names in order.
GetDewarpPTZValues	Allows API clients to retrieve existing fisheye PTZ values with video renderer.
SetDewarpPTZValues	Allows API clients to retrieve existing fisheye PTZ values with video renderer.
SetQuickTrackOptions	Allows API clients to set Quick Track options.
StartQuickTrackRecording	Starts quick track recording asynchronously.
StopQuickTrackRecording	Stops on going quick track recording.
StartThumbnailSearch	Starts thumbnail search asynchronously.
EndThumbnailSearch	Stops on going thumbnail search operation.
SetClientCameraOverlay	Sets a client overlay based on the overlay type and in the position defined by the overlay position.
SetServerCameraOverlay	Instructs the currently connected sever to change the overlay parameters for the camera in the current CVVideo instance.
GotoHomePreset	Moves camera to a given home preset or the active home preset.
SetTranscoders	Sets the H.264 and MPEG4 decoder type.
SetGen2AnalogImageSettings	Sets the image settings for GenII cameras
SetVideoRenderer	Sets the video renderer option.
SetVideoCategories	Sets the video categories of clips for playback.
EnablePresetTour	Enables or disables preset tour on a PTZ camera.
SetMousePtrPtz	Enable or disable mouse based ptz. (AMAG requested)
EnableTls	Enable or disable TLS communitactions to and from Recording Server.
StartLocalRecording	Starts local recording
StopLocalRecording	Stops local recording

ICVVideo3::ProcessIFrameOnly

The *ProcessIFrameOnly* method enables the ActiveX control to process I-Frames only. This only applies to video playback.

Syntax (C++)

```
HRESULT ProcessIFrameOnly(
   [in]  VARIANT_BOOL Enable
);
```

Parameters

Enable [in]

VARIANT_TRUE indicates the renderer to process I-Frames only and skip P-Frames.
VARIANT_FALSE disabled I-Frame only processing.

Return Value

The return value is *S_OK*.

Remarks

ProcessIFrameOnly is associated with the **Connection** context. Meaning this API will take effect only after successful connection to the Recording Server.

ICVVideo3::ForceToSeekTime

The *ForceToSeekTime* method enables playback from a desired position within specified playback interval. The playback interval is specified by the 'Start' and 'End' times. The 'Seek' time must be between the 'Start' and 'End'.

Syntax (C++)

```
HRESULT ForceToSeekTime (
   [in]  DATE Start,
   [in]  DATE End,
   [in]  DATE Seek,
   [out,retval]  VARIANT_BOOL *RetVal
);
```

Parameters

Start [in]

Indicates beginning time of a playback interval.

End [in]

Indicates end time of a playback interval.

Seek [in]

Indicates position time at which the playback is desired to resume.
Must be between *Start* and *End*

Result [out]

Receives a result code indicating the success or failure of the operation. A value of zero indicates success. Any other value is an internal implementation-defined error code and indicates failure.

Return Value
The return value is *S_OK*.

Remarks
ForceToSeekTime is associated with the **Connection** context, i.e. it only makes sense to call it when there is a current camera. *ForceToSeekTime* is a best-effort method providing no indication of whether or when the server/camera actually executes the command. If *Start* and *End* are not in server local time, specify the time zone by calling *SetTimeZone* method.

ICVVideo3::SetTimeZoneName
The *SetTimeZoneName* method allows the API clients to specify custom timezone name. This is the timezone's standard name. See Remarks.

Syntax (C++)
```
HRESULT SetTimeZoneName(
   [in]  BSTR TimeZoneName
);
```

Parameters
TimeZoneName [in]
Time zone standard name (see Remarks).

Return Value
The return value is *S_OK*.

Remarks
Time zones can either be set using the SetTimeZoneName method or using the SetTimeZone method. The methods are mutually exclusive and it is not required that both be used. Selecting one of the two methods is sufficient.

NOTE: It is strongly recommended to use a language-invariant time zone name, such as the key name used to define the time zone in the Windows Registry. Doing so ensures correct operation in environments that may use multiple operating system languages in use across the deployment. In the Win32 programming API, this would correspond to the `TimeZoneKeyName` value in the `DYNAMIC_TIME_ZONE_INFORMATION` data structure.

ICVVideo3::SetDewarpOptions
The *SetDewarpOptions* method allows the API clients to specify dewarp/fisheye options pertaining to lens such as orientation, profile, type, co-ordinates of center, radius and any data pertaining to curvature.

Syntax (C++)
```
HRESULT SetDewarpOptions (
   [in]  struct DewarpOptions Options
);
```

Parameters
Options [in]
Dewarp options, see DewarpOptions data structure.

Return Value
The return value is *S_OK*.

Remarks
SetDewarpOptions updates the fisheye options with the video renderer. A connection update is required for these dewarp options to take effect. This method is associated with **Connection context** and an explicit call to *Connect* is required after SetDewarpOptions *is called.*

ICVVideo3::SetLensDewarpType
The *SetLensDewarpType* method allows the API clients to specify type of fisheye view for the video renderer.

Syntax (C++)
```
HRESULT SetLensDewarpType (
   [in]  enum DewarpViewType dewarpViewType
);
```

Parameters
dewarpViewType [in]
Type of fisheye view such as rectangular (DVT_RECT), quad (DVT_QUAD), dual panorama (DVT_PANODUAL), etc. See DewarpViewType enumeration.

Return Value
The return value is *S_OK*.

Remarks
SetLensDewarpType updates the fisheye view type with the video renderer. A connection update to the Recording Server is required for the view type to take effect. This method is associated with **Connection context** and an explicit call to *Connect* is required after *SetLensDewarpType is called.*

ICVVideo3::SetPresetName
The *SetPresetName* method allows the API clients to specify a custom preset name.

Syntax (C++)
```
HRESULT SetPresetName (
   [in]  INT     PresetIndex,
   [in]  BSTR    PresetName
);
```

Parameters
PresetIndex [in]
One based preset index.

PresetName [in]
Desired name of the preset

Return Value
The return value is *S_OK*.

Remarks

The *SetPresetName* method is “best effort” in the sense that the client application is given no indication of the success or failure.

ICVVideo3::GetPresetNames

The *GetPresetNames* method allows API clients to retrieve existing preset names for the connected camera in the current renderer.

Syntax (C++)

```
HRESULT GetPresetNames (
   [out] VARIANT* PresetNames,
   [out] BSTR*   Error,
   [out,retval] VARIANT_BOOL* Result
);
```

Parameters

PresetNames [in]

List of preset names, if present, ordered by id.

Error [out]

Error description for error while retrieving preset names.

Result [out]

VARIANT_TRUE if success.

Return Value

The return value is *S_OK*.

Remarks

If there are presets configured with no names on Recording Server, *PresetNames* will contain empty preset names. This is to notify the API client, the number of presets configured for this camera. This method is associated with **Connection** context.

ICVVideo3::GetDewarpPTZValues

The *GetDewarpPTZValues* method allows API clients to retrieve existing fisheye PTZ values with video renderer.

Syntax (C++)

```
HRESULT GetDewarpPTZValues (
   [out] VARIANT* PanValues,
   [out] VARIANT* TiltValues,
   [out] VARIANT* ZoomValues,
   [out, retval] VARIANT_BOOL *Result
);
```

Parameters

PanValues [out]

List of floating point pan values for four quadrants in the order – top-left, top-right, bottom-left and bottom-right.

TiltValues [out]

List of floating point tilt values for four quadrants in the order - top-left, top-right, bottom-left and bottom-right.

ZoomValues [out]

List of floating point zoom values for four quadrants in the order - top-left, top-right, bottom-left and bottom-right.

Result[out]

VARTIANT_TRUE if success.

Return Value

The return value is *S_OK*.

ICVVideo3::SetDewarpPTZValues

The *SetDewarpPTZValues* method allows API clients to set fisheye PTZ values for user with the video renderer.

Syntax (C++)

```
HRESULT SetDewarpPTZValues (
   [in]  VARIANT PanValues,
   [in]  VARIANT TiltValues,
   [in]  VARIANT ZoomValues,
   [out, retval] VARIANT_BOOL *Result
);
```

Parameters

PanValues [in]

Input list of floating point pan values for four quadrants in the order - top-left, top-right, bottom-left and bottom-right.

TiltValues [in]

Input list of floating point tilt values for four quadrants in the order - top-left, top-right, bottom-left and bottom-right.

ZoomValues [in]

Input list of floating point zoom values for four quadrants in the order - top-left, top-right, bottom-left and bottom-right.

Result[out]

VARIANT_TRUE if success.

Return Value

The return value is *S_OK*.

Remarks

SetDewarpPTZValues updates the PTZ values with the video renderer. A connection update to the Recording Server is required for the view type to take effect. This method is associated with **Connection context** and an explicit call to *Connect* is required after *SetDewarpPTZValues* is called.

ICVVideo3::SetQuickTrackOptions

The *SetQuickTrackOptions* method allows API clients to set QuickTrack options for QuickTrack recording.

Syntax (C++)

```
HRESULT SetQuickTrackOptions (
    [in] BSTR Server,
    [in] INT Camera,
    [in] BSTR Username,
    [in] BSTR Password,
    [in] enum CompressionType CompressionType,
    [in] INT Width,
    [in] INT Height,
    [in] INT FrameRate,
    [out] BSTR* Error,
    [out, retval] VARIANT_BOOL *Result
);
```

Parameters

Server [in]

QuickTrack Server hostname / IP address.

Camera [in]

Camera identifier (one-based index) of an existing QuickTrack camera on a QuickTrack Server.

Username [in]

QuickTrack Server username.

Password[in]

QuickTrack Server password.

CompressionType [in]

Compression type, see CompressionType enumeration.

Width [in]

Image width in pixels.

Height [in]

Image height in pixels.

FrameRate [in]

Frames per second for QuickTrack Recording.

Error [out]

Error while starting the QuickTrack recording, if any.

Result [out, retval]

VARIANT_TRUE is QuickTrack recording starts successfully.

Return Value

The return value is *S_OK*.

ICVVideo3::StartQuickTrackRecording

This method starts quick track recording asynchronously.

Syntax (C++)

```
HRESULT StartQuickTrackRecording (
   [in] BSTR RecordingServer,
   [in] INT RecordingCamera,
   [in] BSTR Username,
   [in] BSTR Password,
   [out] BSTR* Error,
   [out, retval] VARIANT_BOOL *Result
);
```

Parameters

RecordingServer [in]

Recording Server hostname / IP address.

RecordingCamera [in]

Camera index on Recording Server.

Username [in]

Recording Server user name.

Password [in]

Recording Server password.

Error [out]

Error while starting the QuickTrack recording, if any.

Result [out, retval]

VARIANT_TRUE if QuickTrack recording starts successfully.

Return Value

The return value is *S_OK*.

ICVVideo3::StopQuickTrackRecording

This method stops on going quick track recording. It has no effect if quick track recording is not in progress.

Syntax (C++)

```
HRESULT StopQuickTrackRecording (
);
```

Parameters

The *StopQuickTrackRecording* method has no parameters.

Return Value

The return value is *S_OK*.

ICVVideo3:: StartThumbnailSearch

This method starts a thumbnail search.

Syntax (C++)

```
HRESULT StartThumbnailSearch (
   [in] DATE Start,
   [in] DATE End,
   [in] INT Width,
   [in] INT Height,
   [in] INT Quality,
   [in] BSTR TimeZoneStandardName,
   [in] UINT Interval,
   [out] INT* ThumbnailCount,
   [out] BSTR* Error,
   [out, retval] VARIANT_BOOL *RetVal
);
```

Parameters

Start [in]

Start time for search operation.

End [in]

End time for the search operation

Width [in] and *Height*[in]

Resolution for result thumbnails. -1 indicates original resolution

Quality [in]

Quality of the thumbnail. The range is zero (worst quality) to 100 (best quality).

TimeZoneStandardName [in]

Desired time zone standard name (see Remarks)

Interval [in]

Interval in seconds for thumbnail search.

ThumbnailCount [in]

Desired thumbnail count.

Error [out]

Error while starting the Thumbnail search.

Result [out, retval]

VARIANT_TRUE if search starts successfully.

Return Value
The return value is *S_OK*.

Remarks
It is strongly recommended to use a language-invariant time zone name, rather than a localized time zone name, for the *TimeZoneStandardName* input value.

ICVVideo3:: EndThumbnailSearch
This method stops an ongoing thumbnail search operation. It has no effect if thumbnail search is not in progress.

Syntax (C++)
```
HRESULT EndThumbnailSearch (
);
```

Parameters
The *EndThumbnailSearch* method has no parameters.

Return Value
The return value is *S_OK*.

ICVVideo3:: SetClientCameraOverlay
This method sets a client overlay based on the overlay type and in the position defined by the overlay position. The camera for this operation is the camera in the current CVVideo instance.

Syntax (C++)
```
HRESULT SetClientCameraOverlay(
[in] enum VideoOverlay OverlayType,
[in] enum VideoOverlayPosition Position,
[in] BSTR OverlayText,
[out] BSTR *Error,
[out, retval] VARIANT_BOOL *Result
);
```

Parameters
OverlayType [in]
Video overlay type. See VideoOverlay enumeration.

Position [in]
Video overlay position. See VideoOverlayPosition enumeration.

OverlayText [in]
Text to output on the overlay. For user overlay types this value can be set to a custom text string with a maximum length of 64. For any other type of overlay this value can be an empty string. If it is not an empty string and the overlay is not of type user, then the string will be ignored.

Error [out]
Error while requesting the Client camera overlay.

Result [out, retval]
VARIANT_TRUE if overlay was set successfully.

Return Value

The return value is *S_OK*.

ICVVideo3:: SetServerCameraOverlay

The *SetServerCameraOverlay* method instructs the currently connected sever to change the overlay parameters for the camera in the current CVVideo instance.

Syntax (C++)

```
HRESULT SetServerCameraOverlay(
   [in]  BSTR OverlayText,
   [in]  VARIANT_BOOL ShowTimeStamp,
   [in]  VARIANT_BOOL ShowCameraName,
   [in]  VARIANT_BOOL ShowStreamProperties,
   [out, retval] VARIANT_BOOL *Result
);
```

Parameters

OverlayText [in]

Overlay text. A null value or an empty string clears the overlay text.

ShowTimeStamp [in]

A Boolean that specifies whether to include timestamps in the overlay.

ShowCameraName [in]

A Boolean that specifies whether to include the camera name in the overlay.

ShowStreamProperties [in]

A Boolean that specifies whether to include camera stream properties to the server in the overlay.

Result [out, retval]

VARIANT_TRUE if overlay call was successful.

Return Value

The return value is *S_OK* if the command was successfully sent to the server, *S_FALSE* if not connected to a server or a problem occurs with the connection.

Remarks

SetServerCameraOverlay is a best-effort method providing no indication of whether or when the server actually executes the command. In CV Server versions prior to 5.x the SetCameraOverlay method was used to set text, timestamp and camera name overlays. That method is deprecated and it is recommended to use the SetServerCameraOverlay method instead. This method also takes a parameter for stream properties. For versions before 5.x this parameter will have no effect on the overlay.

ICVVideo3:: PtzSupports

The read-only PtzSupports property indicates all the PTZ capabilities the current camera's selected PTZ driver supports. The value of this property is a bitwise-OR combination of the PTZ support codes as shown below.

Bit Position	Meaning when Set
0	Can pan left and right

1	Can set the pan speed
2	Can tilt up and down
3	Can set the tilt speed
4	Can zoom in and out
5	Can set the zoom speed
6	Can open and close iris
7	Can set the iris speed
8	Can focus near and far
9	Can set the focus speed
10	Can enable auto-iris
11	Can enable auto-focus
12	Can show and set presets
13	Can soft-reset camera
14	Can select a monitor (applies to serial PTZ protocols)
15	Indicates that this is an IP PTZ protocol
16	Can center on a particular (x,y) co-ordinate
17	Can quick zoom through mouse wheel

Syntax (C++)

```
HRESULT get_PtzSupports(
   [out] INT *Result
);
```

ICVVideo3::GotoHomePreset

The *GotoHomePreset* method instructs the current camera to move (pan, tilt, zoom, focus, and/or iris) to a predefined home preset location.

Syntax (C++)

```
HRESULT GotoHomePreset (
   [in, defaultvalue(-1)] SHORT presetIndex
);
```

Parameters

presetIndex [in]

This is the home preset index. For example, if a camera has 4 home presets configured, this input could be 1,2,3 or 4. The Recording Server will move the camera to the preset index associated with this home preset. If a preset is not provided, the Recording Server will move the camera to the current active home preset.

Return Value

The return value is *S_OK*.

Remarks

GotoHomePreset is associated with the **Connection** context, i.e. it only makes sense to call it when there is a current camera.

GotoHomePreset is a best-effort method providing no indication of whether or when the camera actually executes the command.

ICVVideo3:: SetTranscoders

The *SetTranscoders* method sets the H.264 and MPEG4 decoder option for the camera in the current CVVideo instance.

Syntax (C++)

```
HRESULT SetTranscoders(
      [in] enum Transcoder H264Decoder,
      [in] enum Transcoder Mpeg4Decoder);
```

Parameters

H264Decoder [in]

H.264 decoder type. See Transcoder enum.

Mpeg4Decoder [in]

MPEG4 decoder type. See Transcoder enum.

Return Value

The return value is *S_OK* if the command was successfully sent to the server, *S_FALSE* if not connected to a server or a problem occurs with the connection.

Remarks

SetTranscoders must be called before making live view or playback calls. If this call is not made, the server will default to FFMPEG.

ICVVideo3:: SetGen2AnalogImageSettings

The *SetGen2AnalogImageSettings* method sets the analog image settings for the camera in the current CVVideo instance.

Syntax (C++)

```
HRESULT SetGen2AnalogImageSettings(VARIANT videoProps, VARIANT
videoPropValues, BSTR *error, VARIANT_BOOL *result);
```

Parameters

videoProps [in]

Array of Gen II video properties where each property is of type int and is one of the below values.

```
enum VideoProcProperty : int
{
    Brightness = 0,
    Contrast,
    Hue,
    Saturation,
    Sharpness
}
```

videoPropValues [in]

Array of values of type int for the above properties. The range for sharpness is 0 to 15 and for all the properties, the range is 0 to 255.

error [out]

Error message.

result [out]

Receives a result code indicating the success or failure of the operation

Return Value

The return value is *S_OK* if the command was successfully sent to the server, *S_FALSE* if not connected to a server or a problem occurs with the connection.

Remarks

This call is applicable for only Gen II analog cameras and require camera to be active for the operation.

ICVVideo3:: SetVideoRenderer

The *SetVideoRenderer* method sets the video renderer option for the camera in the current CVVideo instance.

Syntax (C++)

```
HRESULT SetVideoRenderer(
      [in] enum VideoRendererOption option);
```

Parameters

option [in]

Video renderer type. See VideoRendererOption enum.

Return Value

The return value is *S_OK* if the command was successfully sent to the server, *S_FALSE* if not connected to a server or a problem occurs with the connection.

Remarks

SetVideoRenderer must be called before making live view or playback calls. If this call is not made, the server will default to Direct 2D/3D.

ICVVideo3::SetVideoCategories

The SetVideoCategories method sets the video categories of all the clips for playback in the current CVVideo instance.

Syntax (C++)

```
HRESULT SetVideoCategories(
      [in] INT videoCategories);
```

Parameters

`videoCategories` [in]

Two's exponent of VideoCategory enum value of each clip bitwise OR'ed together.

```
enum VideoCategory : int
{
    Scheduled = 0,
    Alarm = 1,
    Motion = 2,
    Exported = 3
};
```

Return Value

The return value is *S_OK* if the command was successfully sent to the server, *S_FALSE* if not connected to a server or a problem occurs with the connection.

Remarks

By default scheduled, motion and alarm video categories are set in the CVVideo instance. To modify this default value, *SetVideoCategories* must be called. For example, to play exported clips, *SetVideoCategories* must be called with the export category set.

ICVVideo3::EnablePresetTour

The EnablePresetTour method can be used to enable or disable preset tour option on a PTZ camera.

Syntax (C++)

```
HRESULT EnablePresetTour([in] VARIANT_BOOL enablePreset, [out] BSTR *error,
[out, retval] VARIANT_BOOL *result);
```

Parameters

`enablePreset` [in]

Boolean indicating whether to enable (*VARIANT_TRUE*) or disable (*VARIANT_FALSE*) preset tour.

`error` [out]

Error message indicating the reason for failure of operation.

`result` [out, retval]

Returns a boolean value indicating the success (*VARIANT_TRUE)* or failure (*VARIANT_FALSE)* of the operation.

Return Value

The return value is *S_OK*.

ICVVideo3::SetMousePtrPtz

The SetMousePtrPtz method can be used to enable or disable mouse based PTZ control.

Syntax (C++)

```
HRESULT SetMousePtrPtz([in] VARIANT_BOOL control);
```

Parameters

`control` [in]

Boolean indicating whether to enable (*VARIANT_TRUE*) or disable (*VARIANT_FALSE*) mouse PTZ control.

Return Value

The return value is *S_OK*.

ICVVideo3::EnableTls

The EnableTls method can be used to enable or disable secure TCP communications to the Recording Server.

Syntax (C++)

```
HRESULT EnableTls([in] VARIANT_BOOL enable)
```

Parameters

`enable` [in]

Boolean indicating whether to enable (*VARIANT_TRUE*) or disable (*VARIANT_FALSE*) secure TCP communications.

Return Value

The return value is *S_OK*

ICVVideo3::StartLocalRecording

This method allows to record video locally on client side.

Syntax (C++)

```
HRESULT StartLocalRecording([in] struct LocalRecordingArgs recordingArgs,
[out] BSTR *Error, [out, retval] VARIANT_BOOL *Result)
```

Parameters

`recordingArgs` [in]

See LocalRecordingArgs structure.

`Error` [out]

Error message indicating the reason for failure of operation.

`Result` [out, retval]

Returns a boolean value indicating the success (*VARIANT_TRUE)* or failure (*VARIANT_FALSE)* of the operation.

Return Value

The return value is *S_OK*.

ICVVideo3::StopLocalRecording

This method stops video recording.

Syntax (C++)

```
HRESULT StopLocalRecording([out] BSTR *Error, [out, retval] VARIANT_BOOL
*Result)
```

Parameters

`Error` [out]

Error message indicating the reason for failure of operation.

`Result` [out, retval]

Returns a boolean value indicating the success (*VARIANT_TRUE)* or failure (*VARIANT_FALSE)* of the operation.

Return Value

The return value is *S_OK*.

ICVVideo4

Method	Description
SetLanguage	Sets current language for user facing strings.

ICVVideo4::SetLanguage

This method sets the current language. The default language is the system's language if supported and if not, English will be used. The *localeName* needs to be in RFC 4646's Language-Region format.

Syntax (C++)

```
HRESULT SetLanguage([in] BSTR localeName)
```

Parameters

`localeName` [in]

String for the locale to be used.

Return Value

The return value is *S_OK*.

ICVVideo5

Property	Description
StreamId	Identifies the stream (one-based index).

ICVVideo5::StreamId

The *StreamId* property represents the stream identifier (a one-based index). This value determines the particular camera stream to connect to on the next connection attempt.

If this property is not set before calling Connect then the Recording Server will automatically select the stream based on the available camera streams and the requested width/height of the stream (if any).

Syntax (C++)

```
HRESULT get_StreamId(
   [out] INT *Result
);
HRESULT put_StreamId(
   [in]  INT newVal
);
```

Parameters

Result [out]

Receives the current value of the property.

newVal [in]

Target stream identifier (one-based index) for the next connection attempt. Can also be set to <= 0 to explicitly enable automatic stream selection.

Return Value

The return value is *S_OK*.

ICVVideo6

Property	Description
Token	The user access token to use when connecting to the server.
ProxyMode	Whether or not proxy mode is enabled. Set to true if the server to connect to is a proxy/gateway. Set to false if the server to connect to is a Recording Server. By default, this property is set to false.
RecordingServerId	The Recording Server GUID to send to the proxy/gateway server if proxy mode is enabled. Required in order to use proxy mode so that the proxy/gateway server knows where to forward server control messages.

Method	Description
SetDecodeThreads	Sets the number threads that will be used when decoding stream

ICVVideo6::Token

The *Token* property determines the user access token that is used on the next connection to a server. Changing this property has no effect on the current connection (if any). The format of this token must be in JSON Web Token (JWT) format (RFC7519). Use this as an alternative to the Username and Password properties.

Syntax (C++)

```
HRESULT get_Token(
   [out] BSTR *Result
);
HRESULT put_Token(
   [in]  BSTR newVal
);
```

Parameters

Result [out]

Receives the current value of the property.

newVal [in]

User access token to use in the next connection to a server.

Return Value

The return value is *S_OK*.

ICVVideo6::ProxyMode

The *ProxyMode* property determines whether or not proxy mode is enabled. Set to true if the server to connect to is a proxy/gateway. Set to false if the server to connect to is a Recording Server. By default, this property is set to false.

If this property is changed and there is an active connection, then the controller will attempt a reconnection. If the reconnection attempt fails then double-check the server address and the value of the Recording Server GUID property (if applicable) or manually reconnect.

Syntax (C++)

```
HRESULT get_ProxyMode(
   [out] VARIANT_BOOL *result
);
HRESULT put_ProxyMode(
   [in]  const VARIANT_BOOL newVal
);
```

Parameters

result [out]

Receives the current value of the property.

newVal [in]

True to enable proxy mode, false to disable proxy mode.

Return Value

The return value is *S_OK*.

ICVVideo6::RecordingServerId

The *RecordingServerId* property specifies the Recording Server GUID to send to the proxy/gateway server if proxy mode is enabled (i.e. if the ProxyMode property is set to true). This value is required in order to use proxy mode so that the proxy/gateway server knows where to forward client control messages.

If this property is changed and there is an active connection, then the controller will attempt a reconnection. If the reconnection attempt fails then double-check the server address and the value of the Recording Server GUID property (if applicable) or manually reconnect.

Syntax (C++)

```
HRESULT get_RecordingServerId(
   [out] GUID *result
);
HRESULT put_RecordingServerId(
   [in]  const GUID newVal
);
```

Parameters

result [out]

Receives the current value of the property.

newVal [in]

The Recording Server GUID to send to the proxy/gateway server if proxy mode is enabled.

Return Value

The return value is *S_OK*.

ICVVideo6::SetDecodeThreads

Sets the number of threads that will be used when decoding streams. Must set before stream opens to have effect.

Syntax (C++)

```
HRESULT SetDecodeThreads (
[in]  int threads
);
```

Parameters

threads [in]

Thread count that will be used.

Return Value

The return value is *S_OK*.

ICVVideo7

Property	Description
InvertPan	When set to TRUE, pan behavior with the mouse behaves opposite from default.
InvertTilt	When set to TRUE, tilt behavior with the mouse behaves opposite from default.

ICVVideo7::InvertPan

The *InvertPan* property determines whether or to invert the pan behavior when using the mouse to pan the camera. Set to true to invert the behavior. Set to false to keep standard behavior. By default, this property is set to false.

Syntax (C++)

```
HRESULT get_InvertPan(
   [out] VARIANT_BOOL *result
);
HRESULT put_InvertPan(
   [in]  const VARIANT_BOOL newVal
);
```

Parameters

result [out]

Receives the current value of the property.

newVal [in]

True to invert pan behavior with the mouse, false to use default behavior.

Return Value

The return value is *S_OK*.

ICVVideo7::InvertTilt

The *InvertTilt* property determines whether or invert the tilt behavior when using the mouse to tilt the camera. Set to true to invert the behavior. Set to false to keep standard behavior. By default, this property is set to false.

Syntax (C++)

```
HRESULT get_InvertTilt(
   [out] VARIANT_BOOL *result
);
HRESULT put_ InvertTilt (
   [in]  const VARIANT_BOOL newVal
);
```

Parameters

result [out]

Receives the current value of the property.

newVal [in]

True to invert tilt behavior with the mouse, false to use default behavior.

Return Value

The return value is *S_OK*.

_ICVVideoEvents

Client application developers should implement this interface if it is desirable to receive asynchronous notifications from a CVVideo control.

_ICVVideoEvents has methods summarized in the next table and described in the text that follows.

Method	Description
OnConnected	Called when a connection to a camera is established.
OnExportProgress	Called to convey progress during export.
OnFailedConnection	Called when a connection attempt ends in failure.
OnLostConnection	Called when an established connection is lost.
OnMouseDblClk	Called when a mouse double click occurs in the control.
OnMouseDown	Called when a mouse button is depressed in the control.
OnMouseMove	Called when a mouse button is moved within the control.
OnMouseUp	Called when a mouse button is released in the control.
OnExportStatus	Called when export video functionality needs to convey error, warning or success regarding the export operation.
OnPlaybackStreamEnd	Called when an end of playback stream is reached.
OnCameraStatusChanged	Called by a CVVideo control when there is a change in camera status
OnPresetNameListChange	Called when the preset list for this camera has been updated.
OnQuickTrackStatus	Called when some indication regarding the ongoing quick track operations needs to be notified.
OnThumbnailSearchResult	Called when resultant thumbnail for a search operation is available.
OnLocalRecordingCompleted	Called when recording is finished.

For all of these methods, the return value is ignored by the CVVideo control.

_ICVVideoEvents::OnConnected

The *OnConnected* method is called by a CVVideo control when it successfully connects to a camera.

Syntax (C++)

```
HRESULT OnConnected(
);
```

Parameters

The *OnConnected* method has no parameters.

_ICVVideoEvents::OnExportProgress

The *OnExportProgress* is called periodically by a CVVideo control while performing an export operation via ExportClip, ExportWizard or SilentExport.

Syntax (C++)

```
HRESULT OnExportProgress(
   [in]  ULONG ExportID,
   [in]  DATE Pos
);
```

Parameters

ExportID [in]

Unused.

Pos [in]

The timestamp of the most recently exported video sample. This value is guaranteed to be greater than the start timestamp of the interval being exported and less than or equal to the end timestamp of that interval, and to increase (though not strictly) across subsequent calls to this method.

_ICVVideoEvents::OnFailedConnection

The *OnFailedConnection* method is called by a CVVideo control when a connection attempt fails.

Syntax (C++)

```
HRESULT OnFailedConnection(
   [in]  LONG Reason
);
```

Parameters

Reason [in]

An internal implementation-defined error code.

_ICVVideoEvents::OnLostConnection

The *OnLostConnection* method is called by a CVVideo control when it loses the current camera connection.

Syntax (C++)

```
HRESULT OnLostConnection(
);
```

Parameters

The *OnLostConnection* method has no parameters.

_ICVVideoEvents::OnMouseDblClk

The *OnMouseDblClk* method is called by a CVVideo control when the user double clicks on the control.

Syntax (C++)

```
HRESULT OnMouseDblClk(
    [in]  MouseButtons Button
);
```

Parameters

Button [in]

Indicates which mouse button was double clicked. See MouseButtons.

_ICVVideoEvents::OnMouseDown

The *OnMouseDown* method is called by a CVVideo control when the user depresses a mouse button while the mouse is over the control.

Syntax (C++)

```
HRESULT OnMouseDown(
    [in]  MouseButtons Button
);
```

Parameters

Button [in]

Indicates which mouse button was pressed. See MouseButtons.

_ICVVideoEvents::OnMouseMove

The OnMouseMove method is called by a CVVideo control when the user moves the mouse over the control.

Syntax (C++)

```
HRESULT OnMouseMove(
    [in]  SHORT X,
    [in]  SHORT Y
);
```

Parameters

X [in]

Mouse X coordinate relative to the left of the control.

Y [in]

Mouse Y coordinate relative to the top edge of the control.

_ICVVideoEvents::OnMouseUp

The *OnMouseUp* method is called by a CVVideo control when the user releases a mouse button while the mouse is over the control.

Syntax (C++)

```
HRESULT OnMouseUp(
    [in]  MouseButtons Button
);
```

Parameters

Button [in]

Indicates which mouse button was released. See MouseButtons.

_ICVVideoEvents:: OnExportStatus

The *OnExportStatus* method is called by a CVVideo control when export video functionality needs to convey error, warning or success regarding the export operation.

Syntax (C++)

```
HRESULT OnExportStatus (
   [in] ULONG exportId,
   [in] enum ExportStatus exportStatus,
   [in] BSTR error
);
```

Parameters

exportId [in]

Id of export operation.

exportStatus [in]

Value of enumeration for export status.

Error [in]

Error description if any.

_ICVVideoEvents:: OnPlaybackStreamEnd

The *OnPlaybackStreamEnd* method is called by a CVVideo control when the end of the playback stream is reached.

Syntax (C++)

```
HRESULT OnPlaybackStreamEnd (
   [in] DATE Position
);
```

Parameters

Position [in]

Date time of the end of playback stream.

_ICVVideoEvents:: OnCameraStatusChanged

The *OnCameraStatusChanged* method is called by a CVVideo control when there is a change in camera status

Syntax (C++)

```
HRESULT OnCameraStatusChanged (
   [in] INT CameraStatus
);
```

Parameters

CameraStatus [in]

Status mask of the camera, see StatusMaskList table.

_ICVVideoEvents:: OnPresetNameListChange

The *OnPresetNameListChange* method is called by a CVVideo control when the preset list for this camera has been updated.

Syntax (C++)

```
HRESULT OnPresetNameListChange (
   [in] VARIANT PresetNameList
);
```

Parameters

PresetNameList [in]

List of preset names (string).

_ICVVideoEvents:: OnQuickTrackStatus

The *OnQuickTrackStatus* method is called by a CVVideo control when some indication regarding the ongoing quick track operations needs to be notified.

Syntax (C++)

```
HRESULT OnQuickTrackStatus (
   [in] VARIANT_BOOL Success,
   [in] BSTR Error
);
```

Parameters

Success [in]

VARIANT_TRUE is Quick Track operation has been successful.

Error [in]

Error description is any.

_ICVVideoEvents:: OnThumbnailSearchResult

The *OnThumbnailSearchResult* method is called by a CVVideo control when resultant thumbnail for a search operation is available

Syntax (C++)

```
HRESULT OnThumbnailSearchResult (
   [in] DATE ThumbnailStart,
   [in] DATE ThumbnailEnd,
   [in] VARIANT Thumbnail,
   [in] VARIANT_BOOL isLast,
   [in] BSTR error
);
```

Parameters

ThumbnailStart [in]

Start time for this thumbnail.

ThumbnailEnd [in]

End time for this thumbnail.

Thumbnail [in]

Actual thumbnail is JPEG format.

isLast [in]

VARIANT_TRUE if this is the last thumbnail in the search results.

error [in]

Error description if any.

_ICVVideoEvents:: OnLocalRecordingCompleted

The *OnLocalRecordingCompleted* method is called by a CVVideo control to notify local recording operation is completed.

Syntax (C++)

```
HRESULT OnLocalRecordingCompleted (
);
```

Parameters

The *OnLocalRecordingCompleted* method has no parameters.

CVCientControl API Enumerations

MouseButtons

The *MouseButtons* enumeration defines the mouse button types.

Syntax (C++)

```
enum MouseButtons {
   CVV_MB_LEFT   = 1<<0,
   CVV_MB_RIGHT  = 1<<1,
};
```

Remarks

CVV_MB_LEFT represents the left mouse button, *CVV_MB_RIGHT* the right mouse button.

ViewOptions

The *ViewOptions* enumeration defines the different viewing types.

Syntax (C++)

```
enum ViewOptions {
   VIEW_STRETCH_TO_FIT = 0,
   VIEW_LETTERBOX,
};
```

Remarks

VIEW_STRETCH_TO_FIT causes the video to stretch to fill the viewing area. *VIEW_LETTERBOX* maintains the source video aspect ratio, introducing horizontal or vertical black bars if necessary.

TimeZoneType

The *TimeZoneType* enumeration represents the time zone of a playback request.

Syntax (C++)

```
enum TimeZoneType {
   TIMEZONE_UNSPECIFIED = 0,
   TIMEZONE_SERVER,
   TIMEZONE_CAMERA,
   TIMEZONE_LOCAL,
   TIMEZONE_UTC,
};
```

Remarks

Each of the above enumerations represents the originating time zone of the start and end times of a playback request. *TIMEZONE_UNSPECIFIED* represents the default time zone which is the server's local time.

ConnectResults

The *ConnectResult* enumeration defines the possible outcomes of a synchronous connection test (see ConnectSync and ConnectSyncResult).

Syntax (C++)

```
enum ConnectResults {
   CR_S_OK             = 0,
   CR_S_INCOMPLETE     = 1,
   CR_F_UNSPECIFIED    = 1000,
   CR_F_UNSUPPORTED    = 1001,
   CR_F_VIDEO          = 1002,
   CR_F_CAMERA         = 1003,
   CR_F_ACCESS         = 1004,
   CR_F_USER           = 1005,
   CR_F_PASSWORD = 1006,
   CR_F_SERVER         = 1007,
   CR_F_LOGIN          = 1008,
   CR_F_PLAYBACK = 1009,
   CR_F_ARGUMENT = 1010,
   CR_F_DSHOW          = 1011,
   CR_F_COMM           = 1012,
   CR_F_WINERR         = 1013,
   CR_F_FILE           = 1014,
   CR_F_NOTCONN  = 1015,
   CR_F_NOTSUPP        = 1016,
   CR_F_LOGINERROR     = 1017,
   CR_F_LDAP           = 1018,
   CR_F_LDAP_SERVER    = 1019,
   CR_F_LDAP_AUTH      = 1020,
   CR_F_ABORT          = 1021,
   CR_F_NOPREVIEW      = 1022,
};
```

Remarks

The result codes in this enumeration are general CompleteView result codes, and not all of them are applicable to synchronous connection tests. Those that are relevant are shown in **maroon** above and are briefly described in the following table.

CR_S_OK	Success.
CR_F_UNSPECIFIED	Unspecified error.
CR_F_ACCESS	Access restriction.
CR_F_USER	Unknown user.
CR_F_PASSWORD	Incorrect password.
CR_F_SERVER	Could not connect to server.
CR_F_LOGINERROR	Login error.
CR_F_LDAP	LDAP error.
CR_F_LDAP_SERVER	Cannot contact the LDAP server.
CR_F_ABORT	Operation aborted.

DewarpLensType

The *DewarpLensType* enumeration represents the vendor of the fisheye lens.

Syntax (C++)

```
enum DewarpLensType
{
	LENS_STANDARD = 0, // Not fisheye
	LENS_IMMERVISION = 1,
	LENS_VIVOTEK = 2,
	LENS_SAMSUNG = 3,
	LENS_SENTRY360 = 4,
	LENS_ACTI = 5,
	LENS_AXIS = 6,
	LENS_ONCAM = 7,
	LENS_HIKVISION = 8,
	LENS_BOSCH = 9,
	LENS_TYPE_END = 10
};
```

ImmerVisionLensProfile

The *ImmerVisionLensProfile* enumeration represents the profile of the Immervision lens.

Syntax (C++)

```
enum ImmerVisionLensProfile
{
	IVLP_NONE = 0,
	IVLP_A0 = 1,		///< A0**V
	IVLP_A1 = 2,		///< A1UST
	IVLP_A8 = 3,		///< A8TRT
	IVLP_BO = 4,		///< BOQQV
	IVLP_B4 = 5,		///< B4QQV
	IVLP_B5 = 6,		///< B5SST
```

```
        IVLP_B6 = 7,            ///< B6SST
        IVLP_B8 = 8,            ///< B8QQT
        IVLP_B7 = 9,            ///< B72YV
        IVLP_B9 = 10,           ///< B9VVT
        IVLP_C1 = 11,           ///< C1ZZV
        IVLP_C3 = 12,           ///< C322V
        IVLP_C4 = 13,           ///< C4VVV
        IVLP_C7 = 14,           ///< C7SST
        IVLP_C8 = 15,           ///< C8WWT
        IVLP_C9 = 16,           ///< C9VVT
        IVLP_C2 = 17,           ///< C2TTV
        IVLP_D1 = 18,           ///< D1SST
        IVLP_D4 = 19,           ///< D4SST
        IVLP_D9 = 20,           ///< D9SSV
        IVLP_END = 21
};
```

DewarpViewType

The *DewarpViewType* enumeration represents the type of the dewarped view.

Syntax (C++)

```
enum DewarpViewType
{
   DVT_NONE = 0,
   DVT_RECT = 1,  // Rectangular or PTZ view
   DVT_QUAD = 2, // Quad view
   DVT_PANODUAL = 3,// Dual Panorama
   DVT_PANOFULL = 4,// Full Panorama
   DVT_PANOCLIP = 5,// Clip Panorama (Only supported by Vivotek 1080p mode)
   DVT_360 = 6,// 360 fisheye view
   DVT_360_3RECT = 7,//Quad view with 360 fisheye view plus 3 rectangular views
   DVT_PANOFULL_2RECT = 8,// Full panorama with 2 rectangular views
   DVT_360_RECT = 9,// Single view with 360 overlay
   DVT_PANONARROW = 10,
   DVT_VERTICALSELFIE = 11, // Vertical selfie mode
};
```

DewarpLensOrientation

The *DewarpLensOrientation* enumeration represents the orientation of fisheye lens.

Syntax (C++)

```
enum DewarpLensOrientation
{
   DLO_NONE = 0,
   DLO_CEILING = 1,
   DLO_WALL = 2,
   DLO_GROUND = 3,
   DLO_END = 4
};
```

ExportStatus

The *ExportStatus* enumeration represents the exported video status.

Syntax (C++)

```
enum ExportStatus
{
   ExportDone = 0,
   ExportWarning,
   ExportError
};
```

CompressionType

The *CompressionType* enumeration represents the type of video compression.

Syntax (C++)

```
enum CompressionType
{
   None = -1,
   Mjpeg = 0,
   Mpeg4 = 1,
   H264 = 2,
   H263 = 3,
   H265 = 4
};
```

VideoOverlay

The *VideoOverlay* enumeration defines the available video overlay types.

Syntax (C++)

```
enum VideoOverlay {
   OverlayNone = -1,
   OverlayCameraName = 0,
   OverlayTimestamp = 1,
   OverlayStreamProperties = 2,
   OverlayUser   = 4,
   OverlayCount = 6
};
```

VideoOverlayPosition

The *VideoOverlayPosition* enumeration defines the available video overlay screen positions.

Syntax (C++)

```
enum VideoOverlayPosition {
   PositionNone  = -1,
   PositionCenter = 0,
```

```
   PositionTopLeft = 1,
   PositionTopRight = 2,
   PositionBottomLeft = 3,
   PositionBottomRight = 4
};
```

Transcoder

The *Transcoder* enumeration defines the available decoder options for H.264 and MPEG4 streams.

Syntax (C++)

```
enum Transcoder {
   TranscoderNone = -1,
   TranscoderIpp = 0,
   TranscoderFfmpeg = 1
};
```

VideoRendererOption

The *VideoRendererOption* enumeration defines the available decoder options for H.264 and MPEG4 streams.

Syntax (C++)

```
enum VideoRendererOption{
      RendererGdi = 0,
      RendererDirect2D = 1
};
```

CVClientControl API Structures

DewarpOptions

The *DewarpOptions* structure represents the options needed by video renderer for dewarping live or recorded video.

Syntax (C++)

```
struct DewarpOptions
{
   enum DewarpLensOrientation lensOrientation;
   enum ImmerVisionLensProfile lensProfile;
   enum DewarpLensType lensType;
   INT centerX;
   INT centerY;
   INT radius;
   VARIANT lensCurveData;
};
```

Parameters

lensOrientation

Orientation of the lens, see DewarpLensOrientation enumeration.

lensProfile

Lens profile specific to Immervision type lenses, see ImmerVisionLensProfile enumeration.

lensType

Dewarp lens type, see DewarpLensType enumeration.

centerX

X coordinate of the center.

centerY

Y coordinate of the center.

radius

Radius of the fisheye camera.

lensCurveData

Lens curve data, byte array.

LocalRecordingArgs

The *LocalRecordingArgs* structure represents the options needed for StartLocalRecording() API to record video.

Syntax (C++)

```
struct LocalRecordingArgs
{
   BSTR userFilePath ;
   DOUBLE userFps ;
   Enum CompressionType userCompressionType ;
   INT userWidth ;
   INT userHeight ;
   UINT64 userDurationToRecord ;
   BSTR userFileName ;
   Enum VideoOverlayPosition userOverlayPosition ;
};
```

Parameters

userFilePath (required)

Destination to store recorded AVI clip.

userFps (optional)

Frame-Rate per second at which to record clip.

userCompressionType (optional)

Compression type of recorded clip. See CompressionType enumeration.

userWidth (optional)

Width of recorded clip.

userHeight (optional)

Height of recorded clip.

userDurationToRecord (optional)

Specific duration to record clip in seconds.

userFilename (optional)

File Name for recorded clip.

userOverlayPosition (optional)

See VideoOverlayPosition enumeration.

Note – When this option is specified by user, it is recommended to disable server-side overlays.

Additional Resources

Visit the Salient website, www.salientsys.com, for additional support and CompleteView training:

- **Manuals & Documentation** (https://support.salientsys.com/hc/en-us/categories/115000292747-Knowledge-Base) – Includes Administrator's Manual, Client User Manuals (including Video, Alarm and Web clients), How To Guides and Tips.

- **Online Tech Support** (https://support.salientsys.com/hc/en-us)– Get quick access to online tech support modules that cover the most frequently asked product questions, such as "Adding and Moving IP Camera Licenses."

- **Training** (https://support.salientsys.com/hc/en-us/categories/115000302988-Training)– we offer both online and classroom training.
    - CompleteView™ Online Certification - Register online for access to interactive training modules covering the Video, Alarm, Mapping and Web clients.
    - CompleteView™ Classroom Certification - Our traditional classroom training is available throughout the United States. Please visit the Salient website for link to online training, training calendar, agenda and registration.

Salient Systems
4616 W. Howard Ln.
Building 1, Suite 100
Austin, TX 78728
512.617.4800
512.617.4801 Fax
www.salientsys.com