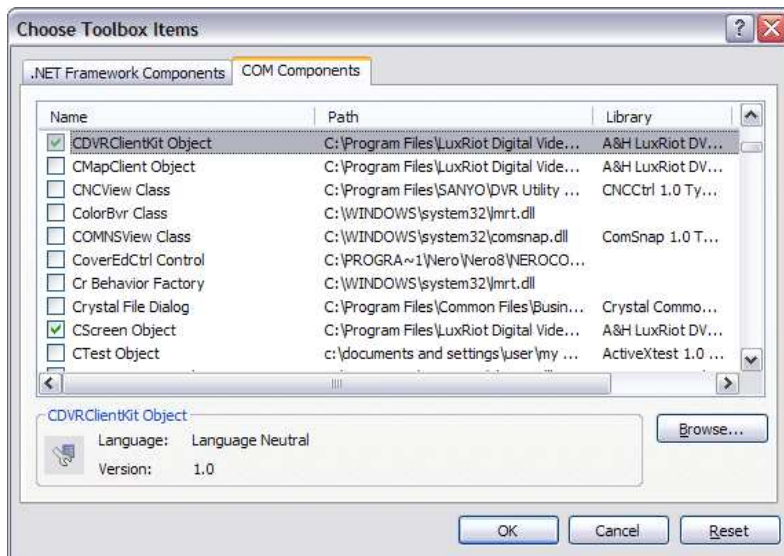


Luxriot Client Kit Manual

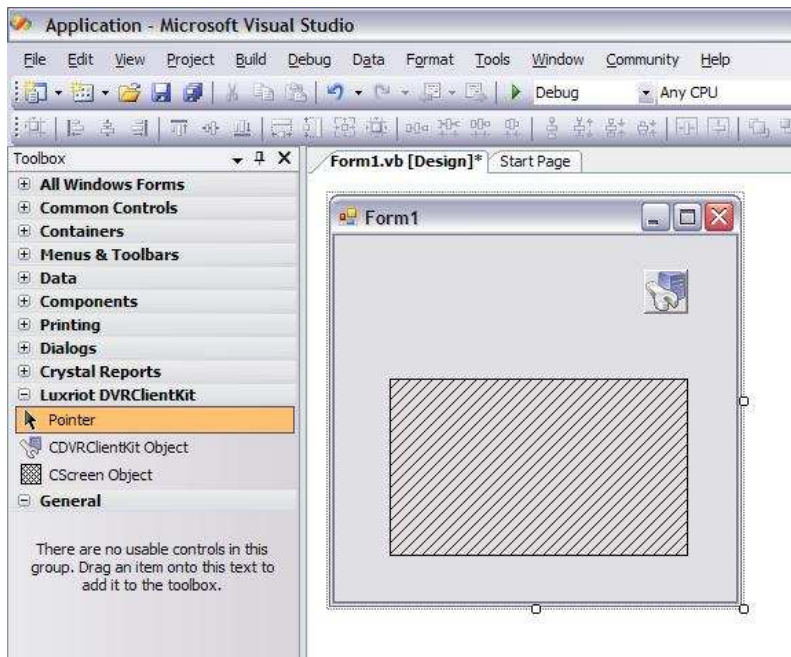
Table of Contents

[Adding components](#)
[Connecting to server](#)
[Handling server events](#)
[Working With Time](#)
[Getting camera list](#)
[Camera configuration](#)
[Inner Camera Parameters](#)
[Camera events](#)
[Camera Ptz Control](#)
[Show live video](#)
[Show archived video](#)
[Runtime](#)
[Interface description](#)

Adding components

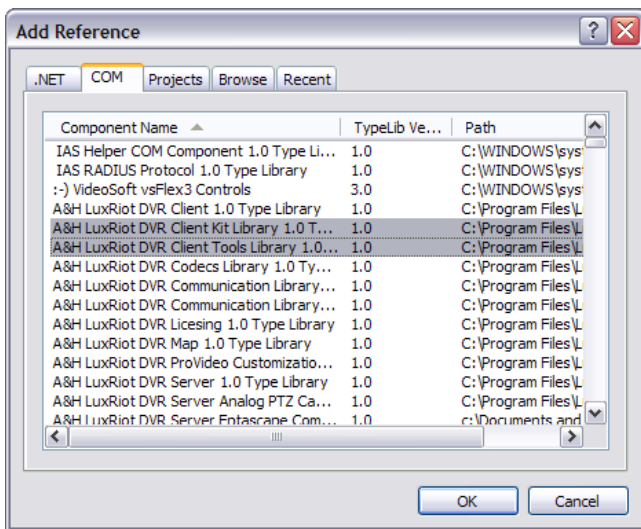


Luxriot Client Kit requires two ActiveX components to be installed: CDVRClientKit and CScreen. To install these components, open menu Tools -> Add/Remove Toolbox Items and choose COM Components tab, where required components (CDVRClientKit Object и CScreen Object) can be added by checking them.



CDVRClientKit is the main component, which implements the most important tasks. CScreen is the component to visualize received video frames from a camera.

Once the components are added into Toolbox, they are ready to be placed on a form to start their use.



Inner objects

In order to use inner API objects it is required to add a reference to COM module «DVR Client Kit Module» in the project settings. In Solution Explorer please right click References -> Add Reference. Choose COM tab there and double click the following items to select them: «A&H LuxRiot DVR Client Tools Library» and «A&H Luxriot DVR Client Kit Library».

Connecting to server

To connect to Luxriot DVR Server it is required to create a server object:

```
Dim Server As DVRClientKitModule.ICkServer
Server = Me.AxDVRClientKit.CreateServer("Custom Title")
```

For connecting to local server use **Server.Host = "localhost"** (or **"127.0.0.1"** without port). Port should be specified only for remote connections, for example **Server.Host = "192.168.1.150:60554"** If connection is local, communication between server and client is done through COM interface, which noticeably speeds up operations. You can also emulate remote connection by connection to local machines ip address:port.

Then logon information is configured with the created object before it can start connection attempts:

```
Server.Host = "192.168.1.150:60554"
Server.UserName = "admin"
Server.Password = ""
```

To connect or disconnect use methods `Connect()` and `Disconnect()` and wait for incoming event `OnServerConnected` ([below_link_here](#)).

```
Server.Connect()
...
Server.Disconnect()
```

Once server object is no longer needed, it may be deleted from current server list:

```
Me.AxDVRClientKit.RemoveServer(Server)
```

Handling server events

Server object events are routed to `DVRClientKit` object and handled on this parent object.

The following events inform about change in server status:

```
OnServerConnecting([in] ICkServer *pServer);
OnServerConnected([in] ICkServer *pServer, [in] LONG nStatus)
OnServerDisconnecting([in] ICkServer *pServer, [in] LONG nConnectionResult)
OnServerDisconnected([in] ICkServer *pServer, [in] LONG nConnectionResult)
```

where `pServer` is the server, which is a source of event, `nStatus` - HRESULT error code querying camera information from server (if any) and `nConnectionResult` - HRESULT error code connecting to remote server (if any).

Example:

```
Private Sub AxDVRClientKit_OnServerConnected
ByVal sender As System.Object,
ByVal e As AxDVRClientKitModule._DVRClientKitEvents_OnServerConnectedEvent
Handles AxDVRClientKit.OnServerConnected

' Parsing event arguments
Dim ServerArg As DVRClientKitModule.ICkServer = e.pServer
Dim Status As Long = e.nStatus
End Sub
```

Working With Time

All time functions take arguments and return values in UTC time. However, IChServer interface has functions to convert UTC time to local time and opposite.

```
IChServer::ConvertUTCtimeToServerTime([in] DATE fUTCtime, [out, retval] *pfServerTime)
```

will convert UTC time to local server time and return it

```
IChServer::ConvertServerTimetoUTCtime([in] DATE fServerTime, [out, retval] *pfUTCtime)
```

will convert local server time to UTC time and return it

Getting camera list

As soon as Client Kit connected to server, it is possible to obtain server's camera list: Example:

```
Dim MediaDevicesArray = Server.GetMediaDevices()  
Dim MediaDevice As DVRClientKitModule.IChMediaDevice  
For Each MediaDevice In MediaDevicesArray  
CameraListComboBox.Items.Add(MediaDevice.Title)  
Next MediaDevice
```

Camera configuration

To configure a camera use methods of interface IChMediaDevice (see description below)

Example:

```
MediaDevice.Title = "New Camera Title"  
MediaDevice.RecordingEnabled = True  
MediaDevice.TimelapseRecordingFrameRate = 1.0
```

It is also possible to show standard camera properties sheet (the same as in Luxriot DVR). For this please call method IChMediaDevice::ShowProperties()

Example:

```
Private Sub PropertiesButton_Click(ByVal sender As System.Object,  
ByVal e As System.EventArgs) Handles PropertiesButton.Click  
MediaDevice.ShowProperties(Me.Handle) End Sub
```

Inner Camera Parameters

You can get additional camera parameters, which can help you identify camera

```
IChMediaDevice::GetComponentIdentifier([out, retval] BSTR *psComponentIdentifier)
```

This function will return device type ID ("Network (IP) Camera", "Media Source" or other)

```
IChMediaDevice::GetHttpLocation([out, retval] BSTR *psLocation)
```

Will return http address of a camera in following format: "http://username@hostname.com:8080"

If camera does not have http address, function will return an error code 0xC1260010 (3240493072), which means camera is not an IP device.

```
IChMediaDevice::GetChannelIdentifier([out, retval] VARIANT *pvIdentifier)
```

Function will return *VARIANT* channel identifier. Return value can be either integer or string.

Camera events

Camera events are handled by handling events of DVRClientKit parent object.

The following events can be received from a camera:

```
OnSample([in] IChMediaDevice* pMediaDevice, [in] IUnknown* pSampleUnknown)
```

Event is called upon receiving frame from camera

```
OnMotion([in] IChMediaDevice *pMediaDevice)
```

camera motion detector triggered motion.

```
OnNotification([in] IChMediaDevice* pChMediaDevice, [in] LONG nCode, [in] VARIANT vParameters)
```

Event is called upon message from camera. Message could be a warning or an error. Example:

```
Private Sub AxDVRClientKit_OnMotion(  
ByVal sender As System.Object,  
ByVal e As AxDVRClientKitModule._DVRClientKitEvents_OnMotionEvent  
) Handles AxDVRClientKit.OnMotion  
  
Dim MediaDevice As DVRClientKitModule.IChMediaDevice = e  
Debug.WriteLine("Motion detected on camera: " + MediaDevice.Title + " (" + MediaDevice.Identifier + ")")  
End Sub
```

Camera Ptz Control

For controlling camera PTZ you first need to get IChPtzControl interface from IChMediaDevice by using Query Interface Example:

```
Dim PtzControl As AxDVRClientKitModule.PtzControl = MediaDevice  
if (Not IsNothing(PtzControl)) Then  
...  
End If
```

Using IChPtzControl interface you can get available PTZ commands list and do PTZ control. You can get list of PTZ commands supported by camera by using following function

```
IChPtzControl::GetCapabilityFlag([out, retval] CK_PTZCONTROLCAPABILITYFLAGS  
*pnCapabilityFlags).CK_PTZCONTROLCAPABILITYFLAGS
```

It can contain one of following flags as well as combination of them

```
typedef enum CK_PTZCONTROLCAPABILITYFLAGS {  
CK_PTZCONTROLCAPABILITYFLAGS_NONE = 0,  
CK_PTZCONTROLCAPABILITYFLAGS_PAN = 1,  
CK_PTZCONTROLCAPABILITYFLAGS_TILT = 2,  
CK_PTZCONTROLCAPABILITYFLAGS_PANTILT = 4,  
CK_PTZCONTROLCAPABILITYFLAGS_RESET = 8,  
CK_PTZCONTROLCAPABILITYFLAGS_ZOOM = 16,  
CK_PTZCONTROLCAPABILITYFLAGS_FOCUS = 32,  
CK_PTZCONTROLCAPABILITYFLAGS_FOCUSAUTO = 64,  
CK_PTZCONTROLCAPABILITYFLAGS_IRIS = 128,  
CK_PTZCONTROLCAPABILITYFLAGS_IRISAUTO = 256,  
} _CK_PTZCONTROLCAPABILITYFLAGS;
```

Depending on capability flags, following PTZ commands might be available

```
typedef enum CK_PTZCOMMAND {  
CK_PTZCONTROLCAPABILITYFLAGS_TILT:  
CK_PTZCOMMAND_UP = 0,  
CK_PTZCOMMAND_DOWN,  
CK_PTZCONTROLCAPABILITYFLAGS_PAN:  
CK_PTZCOMMAND_LEFT,  
CK_PTZCOMMAND_RIGHT,  
CK_PTZCONTROLCAPABILITYFLAGS_PANTILT:  
CK_PTZCOMMAND_UPLIFT,  
CK_PTZCOMMAND_UPRIGHT,  
CK_PTZCOMMAND_DOWNLEFT,  
CK_PTZCOMMAND_DOWNRIGHT,  
CK_PTZCONTROLCAPABILITYFLAGS_ZOOM:  
CK_PTZCOMMAND_ZOOMIN,  
CK_PTZCOMMAND_ZOOMOUT,  
CK_PTZCONTROLCAPABILITYFLAGS_FOCUS:  
CK_PTZCOMMAND_FOCUSIN,  
CK_PTZCOMMAND_FOCUSOUT,  
CK_PTZCONTROLCAPABILITYFLAGS_FOCUSAUTO:  
CK_PTZCOMMAND_FOCUSAUTO,  
CK_PTZCONTROLCAPABILITYFLAGS_IRIS:  
CK_PTZCOMMAND_IRISIN,  
CK_PTZCOMMAND_IRISOUT,  
CK_PTZCONTROLCAPABILITYFLAGS_IRISAUTO:  
CK_PTZCOMMAND_IRISAUTO,  
} _CK_PTZCOMMAND;
```

Additionally, if CK_PTZCONTROLCAPABILITYFLAGS_RESET flag is set, you can use IChPtzControl::Reset() function to restore initial camera position. PTZ control is carried by following function

```
IChPtzControl::StartCommand([in] CK_PTZCOMMAND nPtzCommand)
```

After which you HAVE to call IChPtzControl::StopCommand() function. All IChPtzControl functions can return an error with code 0xC1260011

(3240493073), which means that PTZ for a given device is unavailable. Example:

```
Private Sub PtzForm_Load(ByVal sender As Object, ByVal e As System.EventArgs) Handles MyBase.Load
Dim PtzControl As DVRClientKitModule.ICkPtzControl
PtzControl = SelectedMediaDevice
If (Not IsNothing(PtzControl)) Then
Dim CapabilityFlags As DVRClientKitModule.CK_PTZCONTROLCAPABILITYFLAGS
CapabilityFlags = PtzControl.GetCapabilityFlag()
UpPtzButton.Enabled = CapabilityFlags And CK_PTZCONTROLCAPABILITYFLAGS_TILT
DownPtzButton.Enabled = CapabilityFlags And CK_PTZCONTROLCAPABILITYFLAGS_TILT
...
End If
End Sub
```

```
Private Sub UpPtzButton_MouseDown(ByVal sender As Object,
ByVal e As System.Windows.Forms.MouseEventArgs) Handles PtzUpButton.MouseDown
Dim PtzControl As DVRClientKitModule.ICkPtzControl
PtzControl = SelectedMediaDevice
If(Not IsNothing(PtzControl)) Then
PtzControl.StartCommand(CK_PTZCOMMAND_UP)
End If
End Sub
```

```
Private Sub UpPtzButton_MouseUp(ByVal sender As Object, ByVal e As
System.Windows.Forms.MouseEventArgs) Handles PtzUpButton.MouseUp
Dim PtzControl As DVRClientKitModule.ICkPtzControl
PtzControl = SelectedMediaDevice
If (Not IsNothing(PtzControl)) Then
PtzControl.StopCommand()
End If
End Sub
```

Show live video

To show live video from a camera it is needed to register a window for video playback and then call StartLiveView method. Note: Registration of multiple windows for one camera may significantly reduce playback performance.

Example:

```
Dim SelectedMediaDevice As DVRClientKit.ICkMediaDevice ' Selected media device
' Starting live video
SelectedMediaDevice.RegisterScreen(View.GetOcx()) ' where View is CScreen ActiveX component
SelectedMediaDevice.StartLiveView()

'Stopping live video
SelectedMediaDevice.StopLiveView()
SelectedMediaDevice.UnregisterScreen(View.GetOcx())
```

Warning: Registration of one window with multiple cameras may cause unpredictable execution

Show archived video

To play archive video you need to get ICKStoredStream interface on archive video stream. One ICKMediaDevice can contain multiple video streams. You can receive video stream(s) with following functions

```
ICKMediaDevice::GetLastStoredStream([out, retval] ICKStoredStream **ppStoredStream)
```

Function returning all of recorded video stream

```
ICKMediaDevice::GetStoredStreams([out, retval] VARIANT *pvStoredStreams)
```

Function returning array of archived video streams, related to given camera

```
ICKMediaDevice::GetStoredStreamByTime([in] DATE fTime, [in] VARIANT_BOOL bTakeNearest, [out, retval]
ICKStoredStream **ppStoredStream)
```

Function returning video stream by given time (fTime). If there is no streams for given time and bTakeNearest parameter is set to True, function will return stream closest to given time. To play back archived video it is needed obtain an interfaces to archived video stream, connected with ICKStoredStream interface. This is achieved by calling method GetStoredStream().

Example:

```
Dim StoredStream As DVRClientKitModule.ICkStoredStream = MediaDevice.GetLastStoredStream()
if IsNothing(StoredStream) Then
' Media device isn't contains archive stream
Else
```

```
' Archive ready
End If
```

Once the stream is obtained, similarly to live video, next step is to register CScreen window, which will show video. Then methods of ICkStoredStream interface (see description below) provide control over playback.

Example:

```
Dim FirstTime As Date
Dim LastTime As Date
StoredStream.GetBoundaries(FirstTime, LastTime)
StoredStream.PlaybackTime = FirstTime
StoredStream.RegisterScreen(View.GetOcx())
StoredStream.Play()
' After playback is completed, do not forget to unregister the window.
StoredStream.Stop()
StoredStream.UnregisterScreen(View.GetOcx())
```

Warning: Registration of one window with multiple cameras may cause unpredictable execution

You can also get unique identifier for archive stream with a following function

```
ICkStoredStream::GetIdentifier([out, retval] *psIdentifier).
```

Runtime

For the DVRClientKit to work it is required that Luxriot DVR Client application is installed, or installed are redistributable dll and tlb files, which typically reside in "C:\Program Files\Luxriot Digital Video Recorder Client Kit\Runtime". These files have to be installed along with API and registered in the following order:

```
msvcr71.dll
DVRSupport.dll
DVRLicense.dll
DVRCommunicationInterfaces.tlb
DVRCommunication.dll
DVRCodecs.dll
DVRServer.tlb
DVRServerToolsShared.tlb
DVRServerTools.dll
DVRServerWatchdogTools.dll
DVRClientTools.dll
DVRServerNetworkCameras.dll
DVRServerMediaDevices.dll
ComArT2D.dll
SAA46_32.DLL
DVRServerHiCap.dll
CAT3DI.dll
SQB.dll
DVRServerXeCap.dll
DVRServerAnalogPtzCameras.dll
AV2000SDK.dll
DVRServerNetworkCamerasArecont.dll
DVRClientKit.dll
```

Interface Description

Interface IDvrClientKit

Methods:

HRESULT CreateServer([in] BSTR sCustomTitle, [out, retval] ICkServer **ppServer)

Description:

Method creates a server object and puts it on the global server list, managed by DVRClientKit object. Before releasing server object it is required to call IDvrClientKit::RemoveServer method to remove server from the list.

Parameters:

[in] BSTR sCustomTitle - Custom server name, using only in client application

Returns:

Created ICkServer

HRESULT RemoveServer([in] ICkServer *pServer)

Description:

Removes server from the global server list

Parameters:

[in] ICkServer *pServer - Server to remove

HRESULT GetServers([out, retval] VARIANT* pvServers)

Description:
Returns all created servers array.

HRESULT GetServerByIdentifier([in] BSTR sIdentifier, [out, retval] ICKServer **ppServer)

Description:
Returns created server from the global list by given server's unique identifier. You can get server's identifier by calling function ICKServer::GetIdentifier()
Parameters:
[in] BSTR sIdentifier - Unique server's identifier

HRESULT GetExceptionIdentifier([in] LONG nhResult, [out, retval] BSTR *psDescription)

Description:
Returns description for given error code
Parameters:
[in] LONG nhResult - HRESULT code
Returns:
BSTR - error code description

HRESULT DecodeKeySampleToJpeg([in] IUnknown* pSampleUnkown, [in] VARIANT vParameters, [out] VARIANT* pvData, [out] ULONG* pnDataSize)

Description:
Function which decodes keyframe pSampleUnkown to JPEG format, returning VARIANT array. Data will include JPEG header, so it can be instantly written to disk. If frame is not a keyframe function will return S_FALSE
Parameters:
[in] IUnknown* pSampleUnkown – Sample to decode
[in] VARIANT vParameters — Reserved
[out] VARIANT* pvData – Decoded data
[out] ULONG* pnDataSize – Decoded data size

Interface IDVRClientKitEvents

Methods:

HRESULT OnServerConnecting([in] ICKServer *pServer)

Description:
Raises when server (ICKServer) starting connecting to Luxriot DVR Server
Parameters:
[in] ICKServer *pServer - Server which raise event

HRESULT OnServerConnected([in] ICKServer *pServer, [in] LONG nStatus)

Description:
Raises when server (ICKServer) connected to Luxriot DVR Server
Parameters:
[in] ICKServer *pServer - Server which raise event
[in] LONG nStatus - HRESULT status of connection state, S_OK if no errors.

HRESULT OnServerDisconnecting([in] ICKServer *pServer, [in] LONG nConnectionResult)

Description:
Raises when server (ICKServer) starting disconnecting from Luxriot DVR Server
Parameters:
[in] ICKServer *pServer - Server which raise event
[in] LONG nConnectionResult - HRESULT status of connection state, S_OK if disconnection was graceful.

HRESULT OnServerDisconnected([in] ICKServer *pServer, [in] LONG nConnectionResult)

Description:
Raises when server (ICKServer) disconnected from Luxriot DVR Server
Parameters:
[in] ICKServer *pServer - Server which raise event
[in] LONG nConnectionResult - HRESULT status of connection state, S_OK if disconnection was graceful.

HRESULT OnSample([in] ICKMediaDevice* pMediaDevice, [in] IUnknown* pSampleUnkown)

Description:
Raises when media device (ICKMediaDevice) received video sample from Luxriot DVR Server
Parameters:
[in] ICKMediaDevice* pMediaDevice – Media device which received video sample
[in] IUnknown* pSampleUnkown – Video sample data object. DVRServerTools::ISample interfaces could be queried from pSampleUnkown to get more sample's properties and data.

HRESULT OnMotion([in] ICKMediaDevice *pMediaDevice)

Description:
Raises when motion detected on media device
Parameters:
[in] ICKMediaDevice * pMediaDevice - Media device which raise event

HRESULT OnNotification([in] ICKMediaDevice* pCkMediaDevice, [in] LONG nCode, [in] VARIANT vParameters)

Description:
Raises when media device (IckMediaDevice) received notification from Luxriot DVR Server
Parameters:
[in] ICKMediaDevice* pMediaDevice – Media device which received notification
[in] LONG nCode – Notification HRESULT code
[in] VARIANT vParameters - Reserved

HRESULT OnStoredStreamSample([in] ICKStoredStream *pStoredStream, [in] DATE fSampleTime)

Remarks:
Raises when archive stream draw sample
Parameters:
[in] ICKStoredStream *pStoredStream - Archive stream which raise event
[in] DATE fSampleTime - Time of drawn sample

Interface ICKServer

Methods:

HRESULT GetIdentifier([out, retval] BSTR *psIdentifier)

Returns:
BSTR - Unique identifier of server (for example: "Network (IP) Camera001")

HRESULT Connect()

Description:
Connect to Luxriot DVR Server using properties: Host, UserName, Port and Password

HRESULT Disconnect()

Description:
Disconnect from Luxriot DVR Server

HRESULT GetServerState([out, retval] DVRCLIENTKIT_SERVER_STATE *pServerState)

Returns:
Current server connection state - DVRCLIENTKIT_SERVER_STATE.
Can be one of the following values: DVRCLIENTKIT_SERVER_STATE_DISCONNECTED, DVRCLIENTKIT_SERVER_STATE_CONNECTING, DVRCLIENTKIT_SERVER_STATE_CONNECTED, DVRCLIENTKIT_SERVER_STATE_DISCONNECTEDPAUSE

HRESULT GetMediaDevices([out, retval] VARIANT *pvMediaDevices)

Returns:
VARIANT array of cameras (ICKMediaDevice), configured on server

HRESULT ConvertUTCTimeToServerTime([in] DATE fUTCtime, [out, retval] DATE *pfServerTime)

Description:
Transforms UTC time to local server time.
Parameters:
[in] DATE fUTCtime - UTC time
Returns:
UTC time, transformed to local server time

HRESULT ConvertServerTimeToUTCtime([in] DATE fServerTime, [out, retval] DATE *pfUTCtime)

Description:
Transforms local server time to UTC time
Parameters:
[in] DATE fServerTime - local server time
Returns:
Local server time transformed to UTC time

HRESULT GetServerUTCtime([out, retval] DATE *pfServerUTCtime)

Description:
This function can be used to get server time in UTC format and see if it is set up correctly

Returns:

DATE – servers time in UTC (GMT+0)

Properties:

BSTR UserName - User name to connect

BSTR Password - Password to connect

BSTR Host - Host name or IP address and Port to connect. For example ("192.168.1.150:60554"). If equals to "localhost" then port isn't need.

VARIANT_BOOL AutoReconnect — Attempt to reconnect to remote DVR server automatically in case of disconnection.

BSTR CustomTitle — Server title to be presented in client application (does not affect server settings).

Interface ICkMediaDevice

Methods:

HRESULT GetIdentifier([out, retval] BSTR *psIdentifier)

Description:

Returns unique identifier of the camera.

HRESULT GetServer([out, retval] ICkServer **pServer)

Description:

Returns server object, which the camera belongs to.

HRESULT ShowProperties([in] HANDLE hWnd)

Description:

Shows modal property sheet with camera preferences.

Parameters:

[in] HANDLE hWnd - Owner window handle

HRESULT RegisterScreen([in] IUnknown *pScreen)

Description:

Registers CScreen (ActiveX component for video visualization, see description below) for video playback.

After a call of StartLiveView() method video will start be shown in registered windows

Remark:

Multiple windows registered with one camera may affect playback performance

Parameters:

[in] IUnknown *pScreen — pointer to CScreen object (ActiveX component for video visualization)

HRESULT UnregisterScreen([in] IUnknown *pScreen)

Description:

Unregisters CScreen object to stop showing video through this object

Parameters:

[in] IUnknown *pScreen - pointer to CScreen object to remove from list of receiving video frames for visualization

HRESULT StartLiveView()

Description:

Starts live video playback, video from cameras will be shown through registered (using method RegisterScreen) CScreen objects.

Upon every received frame a IDVRClientKitEvents::OnSample event is called, and If motion is present IDVRClientKitEvents::OnMotion event is called as well. Call to this is equivalent to

ICkMediaDevice::StartStream(DVRCLIENTKIT_MEDIADVICE_STREAMTYPEFLAG_ALL) call.

HRESULT StopLiveView()

Description:

Stops live video playback

HRESULT StartStream([in] DVRCLIENTKIT_MEDIADVICE_STREAMTYPEFLAGS nFlags)

Description:

Starts given stream. It is possible to get motion events using this function, without having to receive video data.

Parameters:

[in] DVRCLIENTKIT_MEDIADVICE_STREAMTYPEFLAGS nFlags – Stream flags to receive events. Can be one or more of the following:

```
typedef enum DVRCLIENTKIT_MEDIADVICE_STREAMTYPEFLAGS
```

```
{
```

```
DVRCLIENTKIT_MEDIADVICE_STREAMTYPEFLAG_NONE = 0,
```

```
DVRCLIENTKIT_MEDIADVICE_STREAMTYPEFLAG_LIVESTREAM = (1 << 0),
```

```
DVRCLIENTKIT_MEDIADVICE_STREAMTYPEFLAG_MOTIONSTREAM = (1 << 1),
```

```
DVRCLIENTKIT_MEDIADVICE_STREAMTYPEFLAG_ALL =
```

```
(DVRCLIENTKIT_MEDIADVICE_STREAMTYPEFLAG_LIVESTREAM |
```

```
DVRCLIENTKIT_MEDIADVICE_STREAMTYPEFLAG_MOTIONSTREAM),
```

```
}_DVRCLIENTKIT_MEDIADVICE_STREAMTYPEFLAGS;
```

HRESULT StopStream([in] DVRCLIENTKIT_MEDIADVICE_STREAMTYPEFLAGS nFlags)

Description:

Stops previously started stream.

Parameters:

[in] DVRCLIENTKIT_MEDIADVICE_STREAMTYPEFLAGS nFlags – Flags of streams you no longer wish to receive. Can contain one or more of the following:

```
typedef enum DVRCLIENTKIT_MEDIADVICE_STREAMTYPEFLAGS
{
    DVRCLIENTKIT_MEDIADVICE_STREAMTYPEFLAG_NONE = 0,
    DVRCLIENTKIT_MEDIADVICE_STREAMTYPEFLAG_LIVESTREAM = (1 << 0),
    DVRCLIENTKIT_MEDIADVICE_STREAMTYPEFLAG_MOTIONSTREAM = (1 << 1),
    DVRCLIENTKIT_MEDIADVICE_STREAMTYPEFLAG_ALL =
    (DVRCLIENTKIT_MEDIADVICE_STREAMTYPEFLAG_LIVESTREAM |
    DVRCLIENTKIT_MEDIADVICE_STREAMTYPEFLAG_MOTIONSTREAM),
}_DVRCLIENTKIT_MEDIADVICE_STREAMTYPEFLAGS;
```

HRESULT GetComponentIdentifier([out, retval] BSTR *psComponentIdentifier)

Description:

Returns device component id. This id can be used to determine reference of a device to given module.

Returns:

BSTR component id, for example "Network (IP) Camera" or "Media Source".

HRESULT GetHttpLocation([out, retval] BSTR *psLocation)

Description:

Returns http address of a given device.

Returns:

BSTR address in form of "http://UserName@HostNameOrIP.com:8080"

HRESULT GetChannelIdentifier([out, retval] VARIANT *pvIdentifier)

Description:

Returns channel id.

Returns:

VARIANT channel id, which can be either a string or a number.

HRESULT GetLastStoredStream([out, retval] ICKStoredStream **ppStoredStream)

Description:

Obtains ICKStoredStream interface to access archived video for the camera

Returns:

ICKStoredStream — interface to access archived video.

RESULT GetStoredStreams([out, retval] VARIANT *pvStoredStreams)

Description:

This function returns all archived streams belonging to device.

Returns:

Archived streams for device in IStoredStream array.

HRESULT GetStoredStreamByTime([in] DATE fTime, [in] VARIANT_BOOL bTakeNearest, [out, retval] ICKStoredStream **ppStoredStream)

Description:

Function will return interface (ICKStoredStream) to archived stream at given time. If there is no stream for given time and bTakeNearest parameter is equal to True, function will return archived stream closest to given time.

Parameters:

[in] DATE fTime - return archived stream at this time.

[in] VARIANT_BOOL bTakeNearest - determines if function should return nearest time if given time is not present in archived stream.

Returns:

ICKStoredStream - stream at given time or time nearest to given

Properties:

BSTR Title - Camera title (for example: "Main office camera(192.168.1.100)")

Recording and motion detection parameters

VARIANT_BOOL MotionDetectorEnabled - Turn on\turn off motion detector

VARIANT_BOOL RecordingEnable - Turn on\turn off recording

VARIANT_BOOL MotionRecordingEnable - Turn on\turn off recording of motion information (motion regions and time)

The following properties are accessible when recording is enabled, otherwise if recording is not configured for the camera they return failure

VARIANT_BOOL TimelapseRecordingEnabled - Enable\Disable timelapse recording

DOUBLE TimelapseRecordingFrameRate - Timelapse recording frame rate (in frames per second)

VARIANT_BOOL MotionControlledRecordingEnabled - Enable\Disable motion controlled recording

DOUBLE MotionControlledRecordingFrameRate - Recording frame rate (in frames per second) when motion is detected. If equals 0.0 then will use maximum available fps
DOUBLE PostMotionRecordingInterval - Time interval (in seconds) to keep recording after motion is detected
DOUBLE MotionControlledRecordingStillFrameRate - Recording frame rate (in frames per second) when no motion is detected. If equals 0.0 then still recording will be disabled

Interface ICKPtzControl

Methods:

HRESULT GetCapabilityFlag([out, retval] CK_PTZCONTROLCAPABILITYFLAGS *pnCapabilityFlags)

Description:

Returns set of flags describing available PTZ commands.

Returns:

CK_PTZCONTROLCAPABILITYFLAGS - flag set. Flags could be one or more of the following:

```
typedef enum CK_PTZCONTROLCAPABILITYFLAGS
{
    CK_PTZCONTROLCAPABILITYFLAGS_NONE = 0,
    CK_PTZCONTROLCAPABILITYFLAGS_PAN = 1,
    CK_PTZCONTROLCAPABILITYFLAGS_TILT = 2,
    CK_PTZCONTROLCAPABILITYFLAGS_PANTILT = 4,
    CK_PTZCONTROLCAPABILITYFLAGS_RESET = 8,
    CK_PTZCONTROLCAPABILITYFLAGS_ZOOM = 16,
    CK_PTZCONTROLCAPABILITYFLAGS_FOCUS = 32,
    CK_PTZCONTROLCAPABILITYFLAGS_FOCUSAUTO = 64,
    CK_PTZCONTROLCAPABILITYFLAGS_IRIS = 128,
    CK_PTZCONTROLCAPABILITYFLAGS_IRISAUTO = 256,
}
```

HRESULT StartCommand([in] CK_PTZCOMMAND nPtzCommand)

Description:

Start execution of given PTZ command.

Parameters:

[in] CK_PTZCOMMAND nPtzCommand - command to execute. Can be one of the following:

```
typedef enum CK_PTZCOMMAND
{
    CK_PTZCOMMAND_UP = 0,
    CK_PTZCOMMAND_DOWN,
    CK_PTZCOMMAND_LEFT,
    CK_PTZCOMMAND_RIGHT,
    CK_PTZCOMMAND_UPLLEFT,
    CK_PTZCOMMAND_UPRIGHT,
    CK_PTZCOMMAND_DOWNLEFT,
    CK_PTZCOMMAND_DOWNRIGHT,
    CK_PTZCOMMAND_ZOOMIN,
    CK_PTZCOMMAND_ZOOMOUT,
    CK_PTZCOMMAND_FOCUSIN,
    CK_PTZCOMMAND_FOCUSOUT,
    CK_PTZCOMMAND_FOCUSAUTO,
    CK_PTZCOMMAND_IRISIN,
    CK_PTZCOMMAND_IRISOUT,
    CK_PTZCOMMAND_IRISAUTO,
}
```

Remarks:

- 1) It is REQUIRED to call StopCommand (see below) after using this function.
- 2) You can only send commands marked as available

HRESULT StopCommand()

Description:

Stops execution of last PTZ command.

HRESULT Reset()

Description:

Returns camera to initial position.

Remarks:

Function can be called only if CK_PTZCONTROLCAPABILITYFLAGS_RESET is returned by GetCapabilityFlags function.

Interface ICKStoredStream

Methods:

HRESULT GetMediaDevice([out, retval] ICKMediaDevice **ppMediaDevice)

Returns:
ICkMediaDevice, which the archive belongs to

HRESULT RegisterScreen([in] IUnknown *pScreen)

Description:
Registers CScreen (ActiveX component for video visualization, see description below) for video playback.
After a call of Play() method video will start be shown in registered windows
Remark:
Multiple windows registered with one camera may affect playback performance
Parameters:
[in] IUnknown *pScreen — pointer to CScreen object (ActiveX component for video visualization)

HRESULT UngeristerScreen([in] IUnknown *pScreen)

Description:
Unregisters CScreen object to stop showing video through this object
Parameters:
[in] IUnknown *pScreen - pointer to CScreen object to remove from list of receiving video frames for visualization

HRESULT GetBoundaries([out] DATE *pfFirstTime, [out] DATE *pfLastTime)

Description:
Returns date and time of first and last archived video frame
Parameters:
[out] DATE *pfFirstTime — date and time of the first archived video frame
[out] DATE *pfLastTime — date and time of the last archived video frame

HRESULT Play()

Description:
Start archived video playback

HRESULT Stop()

Description:
Stop archived video playback

HRESULT GetOneSample()

Description:
Play one video frame and pause

HRESULT GetIdentifier([out, retval] BSTR *psIdentifier)

Description:
Returns unique id of archived stream.
Returns:
BSTR unique id of archived stream.
Properties:
DATE PlaybackTime — current playback date and time, should be withn archive boundaries