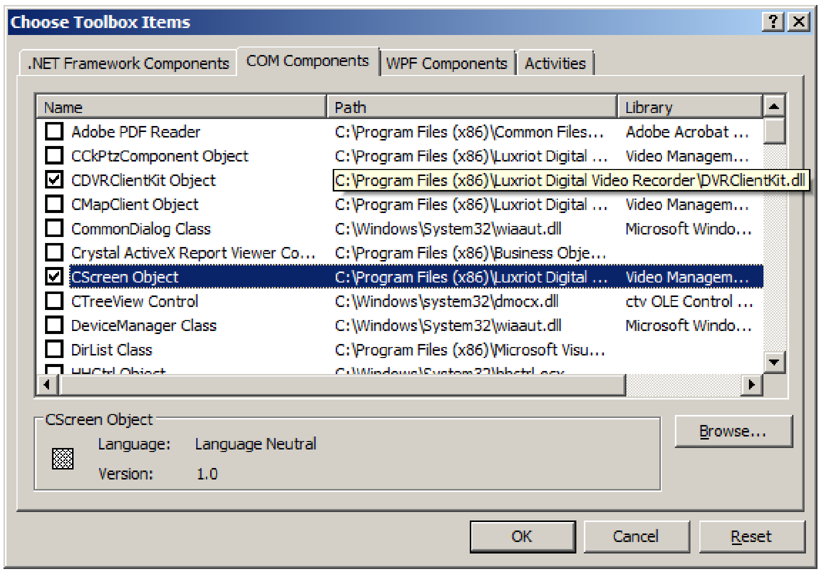
Luxriot ClientKit Manual

Table of Contents

Adding Components

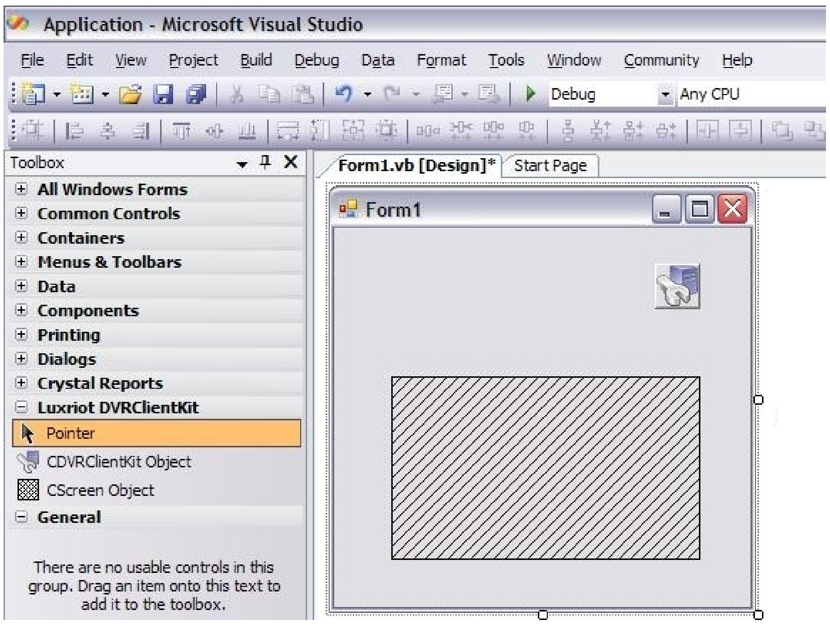

Choose Toolbox Items

.NET Framework Components | COM Components | WPF Components | Activities

Name	Path	Library
☐ Adobe PDF Reader	C:\Program Files (x86)\Common Files...	Adobe Acrobat ...
☐ CCkPtzComponent Object	C:\Program Files (x86)\Luxriot Digital ...	Video Managem...
☑ CDVRClientKit Object	C:\Program Files (x86)\Luxriot Digital Video Recorder\DVRClientKit.dll	
☐ CMapClient Object	C:\Program Files (x86)\Luxriot Digital ...	Video Managem...
☐ CommonDialog Class	C:\Windows\System32\wiaaut.dll	Microsoft Windo...
☐ Crystal ActiveX Report Viewer Co...	C:\Program Files (x86)\Business Obje...	
☑ CScreen Object	C:\Program Files (x86)\Luxriot Digital ...	Video Managem...
☐ CTreeView Control	C:\Windows\system32\dmocx.dll	ctv OLE Control ...
☐ DeviceManager Class	C:\Windows\System32\wiaaut.dll	Microsoft Windo...
☐ DirList Class	C:\Program Files (x86)\Microsoft Visu...	

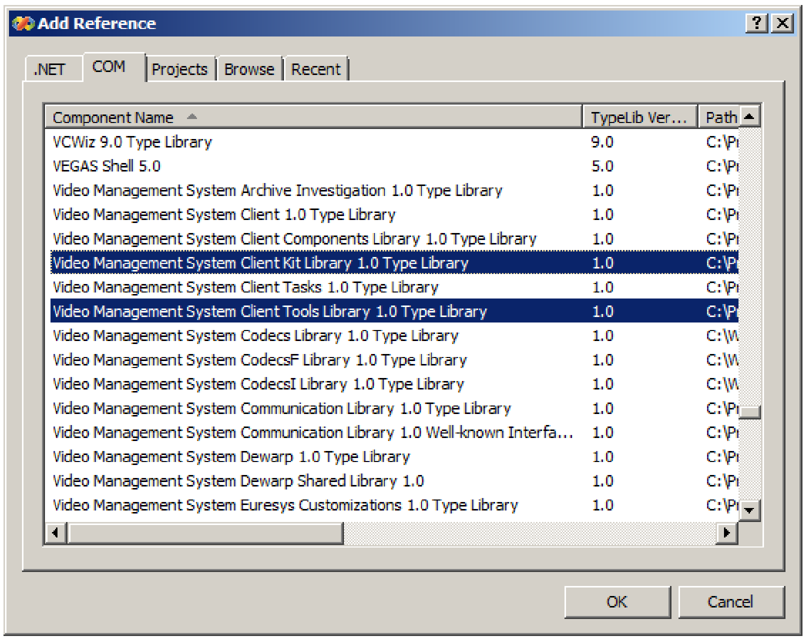
CScreen Object
Language: Language Neutral
Version: 1.0

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.


Add Reference

.NET | COM | Projects | Browse | Recent

Component Name	TypeLib Ver...	Path
VCWiz 9.0 Type Library	9.0	C:\P
VEGAS Shell 5.0	5.0	C:\P
Video Management System Archive Investigation 1.0 Type Library	1.0	C:\P
Video Management System Client 1.0 Type Library	1.0	C:\P
Video Management System Client Components Library 1.0 Type Library	1.0	C:\P
Video Management System Client Kit Library 1.0 Type Library	1.0	C:\P
Video Management System Client Tasks 1.0 Type Library	1.0	C:\P
Video Management System Client Tools Library 1.0 Type Library	1.0	C:\P
Video Management System Codecs Library 1.0 Type Library	1.0	C:\W
Video Management System CodecsF Library 1.0 Type Library	1.0	C:\W
Video Management System CodecsI Library 1.0 Type Library	1.0	C:\W
Video Management System Communication Library 1.0 Type Library	1.0	C:\P
Video Management System Communication Library 1.0 Well-known Interfa...	1.0	C:\P
Video Management System Dewarp 1.0 Type Library	1.0	C:\P
Video Management System Dewarp Shared Library 1.0	1.0	C:\P
Video Management System Euresys Customizations 1.0 Type Library	1.0	C:\P

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 followin items to select them: «Video Management System Client Tools Library» and «Video Management System Client Kit Library».

Connecting To Server

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

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 aslo 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:

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

To connect or disconnect use methods Connect() and Disconnect() and wait for incoming event OnServerConnected.

`VB.NET example`

```
' Create server object
Dim Server As DVRClientKitModule.ICkServer

' Create new server
Server = Me.AxDVRClientKit.CreateServer("Custom Title")

' Set variables describing server location and login information
Server.Host = "192.168.1.150:60554"
Server.UserName = "admin"
Server.Password = ""

' Do the work
Server.Connect()
' ...
Server.Disconnect()

' Clean up
Me.AxDVRClientKit.RemoveServer(Server)
```

...

`C# example`

```
using DVRClientKitModile;

class Program {
  public static void Main ( string[] args ) {
    cKit = new CDVRClientKit();
    ICkServer vmsServer = null;

    vmsServer = cKit.CreateServer("test");

    /* Set variables describing server location and login information */
    vmsServer.Host = "192.168.1.55";
    vmsServer.UserName = "admin";
    vmsServer.UserPassword = "";

    /* add event handlers */
    cKit.OnServerConnecting += cKit_OnServerConnecting;
    cKit.OnServerConnected += cKit_OnServerConnected;

    vmsServer.Connect();
    /* Do the work ... */
    vmsServer.Disconnect();

    /* Clean up */
    cKit.RemoveServer(vmsServer);
  }

  /* Handle events */
  static void cKit_OnServerConnecting ( ICkServer pServer ) {
    Console.WriteLine(String.Format("Connecting.."));
  }

  /* Upon event of an established server connection */
  static void cKit_OnServerConnected ( ICkServer pServer, int nStatus ) {
    Console.WriteLine(String.Format("Server '{0}' connected", pServer.CustomTitle));
  }
}
```

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).

```
VB.NET 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
```

```
C# example:
```

```
static void cKit_OnServerConnected ( ICkServer pServer, int nStatus ) {
 // ...
}
```

Working With Time

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

```
//C++ definition
ICkServer::ConvertUTCTimeToServerTime([in] DATE fUTCTime, [out, retval] *pfServerTime)
```

will convert UTC time to local server time and return it

```
//C++ definition
ICkServer::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:

```
VB.NET example:
```

```
Dim MediaDevicesArray = Server.GetMediaDevices()
Dim MediaDevice As DVRClientKitModule.ICkMediaDevice
```

```
For Each MediaDevice In MediaDevicesArray
    CameraListComboBox.Items.Add(MediaDevice.Title)
Next MediaDevice
```

```
C# example:
```

```
foreach (ICkMediaDevice mDev in (object[])pServer.GetMediaDevices()) {
 /* Skip devices disabled on server */
 if (!mDev.Enabled) continue;
 Console.WriteLine(mDev.Title);
}
```

Camera Configuration

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

```
VB.NET example:
```

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

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

```
VB.NET 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

```
//C++ definition
ICkMediaDevice::GetComponentIdentifier([out, retval] BSTR *psComponentIdentifier)
```

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

```
//C++ definition
ICkMediaDevice::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.

```
//C++ definition
ICkMediaDevice::GetChannelIdentifier([out, retval] VARIANT *pvIdentifier)
```

Function will return *VARIANT* channel identificator. 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:

```
//C++ definition
OnSample([in] ICkMediaDevice* pMediaDevice, [in] IUnknown* pSampleUnknown)
```

Event is called upon receiving frame from camera

```
//C++ definition
OnMotion([in] ICkMediaDevice *pMediaDevice)
```

camera motion detecor triggered motion.

```
//C++ definition
OnNotification([in] ICkMediaDevice* pCkMediaDevice, [in] LONG nCode, [in] VARIANT vParameters)
```

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

```
Private Sub AxDVRClientKit_OnMotion( ByVal sender As System.Object,
                        ByVal e As AxDVRClientKitModule._DVRClientKitEvents_OnMotionEvent )
```

```
                    Handles AxDVRClientKit.OnMotion

    Dim MediaDevice As DVRClientKitModule.ICkMediaDeivce = e
    Debug.WriteLine("Motion detected on camera: "+ MediaDevice.Title + " (" + MediaDevice.Identifier + ")")
End Sub
```

Motion detector settings and events

Analysis depth

```
// C++ definition
MotionDetectorAnalysisDepth([out, retval] LONG* pnDepth);
MotionDetectorAnalysisDepth([in] LONG nDepth);
```

Smoothing level

```
// C++ definition
MotionDetectorSmoothingWidth([out, retval] LONG* pnWidth);
MotionDetectorSmoothingWidth([in] LONG nWidth);
MotionDetectorSmoothingHeight([out, retval] LONG* pnHeight);
MotionDetectorSmoothingHeight([in] LONG nHeight);
```

Detector grid size

```
// C++ definition
MotionDetectorCellRows([out, retval] LONG* pnRows);
MotionDetectorCellRows([in] LONG nRows);
MotionDetectorCellColumns([out, retval] LONG* pnColumns);
MotionDetectorCellColumns([in] LONG nColumns);
```

Sensitivity

```
// C++ definition
MotionDetectorSensitivity([out, retval] DOUBLE* pfSensitivity);
MotionDetectorSensitivity([in] DOUBLE fSensitivity);
```

Exclusion grid

```
// C++ definition
MotionDetectorExclusion([out, retval] VARIANT* pvExclusion);
MotionDetectorExclusion([in] VARIANT vExclusion);
```

Processing frame rate

```
// C++ definition
MotionDetectorMinimalInterframeInterval([out, retval] LONG* pnInterval);
MotionDetectorMinimalInterframeInterval([in] LONG nInterval);
```

Motion source

```
// C++ definition
MotionDetectorSource([out, retval] CKMOTIONSOURCE* pSource);
MotionDetectorSource([in] CKMOTIONSOURCE Source);
```

The following motion events can be received from a camera:

```
// C++ definition
HRESULT OnMotion([in] ICkMediaDevice* pMediaDevice);
HRESULT OnMotion2([in] ICkMediaDevice* pMediaDevice, [in] VARIANT vRegion);
```

pMediaDevice is a device where motion was detected and **vRegion** is a two-dimension VARIANT array of VARIANT_BOOL, which is holding information on where motion was detected.

Function below will print motion detector exclusion grid to console

```
C# example:
```

```
public static void printMDExclusion (ICkMediaDevice mDev) {
  if (mDev == null) return;
  try {
    var exc = (bool[,])mDev.MotionDetectorExclusion;
    int rowCount = exc.GetLength(0);
    int columnCount = exc.GetLength(1);

    /* This variable holds all data to be printed at once
     * since cmd console is very slow */
    string block_output = string.Empty;

    /* Create output delimiter for better visual presentation */
    string delimiter = string.Empty;
    for (int i = 0; i < columnCount; i++)
      delimiter += "==";
    delimiter += "=\n";

    /* Add header and delimiter */
    block_output += String.Format("\tMD Grid:\n\t{0}", delimiter);

    /* Add motion detector exclusion mask in ascii presentation */
    for (int row = 0; row < rowCount; row++) {
      block_output += "\t";
        for (int column = 0; column < columnCount; column++) {
          block_output += String.Format(" {0}", ( exc[row, column] == true ) ? "#" : ".");
        }
      block_output += "\n";
    }

    /* Add tailing delimiter */
    block_output += String.Format("\t{0}\n", delimiter);

    /* Print everything to console */
    Console.Write(block_output);
  } catch (Exception Ex) {
      Console.WriteLine(String.Format("Error: {0}\n", Ex.StackTrace + "\n"));
  }
}
```

Camera PTZ Control

For controlling camera PTZ you first need to get ICkPtzControl interface from ICkMediaDevice by using Query Interface

`VB.NET example:`

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

Using ICkPtzControl 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

```
//C++ definition
ICkPtzControl::GetCapabilityFlag([out, retval] CK_PTZCONTROLCAPABILITYFLAGS *pnCapabilityFlags).CK_PTZCONTROLCAPABILITYFLAGS
```

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

```
// C++ definition
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 capabilty flags, following PTZ commands might be available

```
// C++ definition
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_UPLEFT,
    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 **`ICkPtzControl::Reset()`** function to restore initial camera position. PTZ control is carried by following function

```
//C++ definition
ICkPtzControl::StartCommand([in] CK_PTZCOMMAND nPtzCommand)
```

After which you HAVE to call ICkPtzControl::StopCommand() function. All ICkPtzControl functions can return an error with code 0xC1260011 (3240493073), which means that PTZ for a given device is unavailable.

```
VB.NET 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.

```
VB.NET 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

```
//C++ definition
ICkMediaDevice::GetLastStoredStream([out, retval] ICkStoredStream **ppStoredStream)
```

Function returning all of recorded video stream

```
//C++ definition
ICkMediaDevice::GetStoredStreams([out, retval] VARIANT *pvStoredStreams)
```

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

```
//C++ definition
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().

```
VB.NET 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.

```
VB.NET 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.Pause()
StoredStream.UnregisterScreen(View.GetOcx())
```

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

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

```
//C++ definition
ICkStoredStream::GetIdentifier([out, retval] *psIdentifier)
```

Export archived video

ICkStoredStreamExport interface is used to export archived video to avi files.
Interface is obtained by calling Export() from ICkStoredStream interface.

```
//C++ definition
HRESULT ICkStoredStream::Export([in] _CKEXPORTTYPE ExportType, [out, retval] ICkStoredStreamExport** ppStoredStreamExport);
```

Get identifier

```
//C++ definition
HRESULT ICkStoredStreamExport::Identifier([out, retval] BSTR* psIdentifier);
```

Get stored stream identifier

```
//C++ definition
HRESULT ICkStoredStreamExport::StoredStreamIdentifier([out, retval] BSTR* psIdentifier);
```

Get/set export destination

```
//C++ definition
HRESULT ICkStoredStreamExport::Path([out, retval] BSTR* psPath);
HRESULT ICkStoredStreamExport::Path([in] BSTR sPath);
```

Get/set export since time

```
//C++ definition
HRESULT ICkStoredStreamExport::SinceTime([out, retval] DATE* pfTime);
HRESULT ICkStoredStreamExport::SinceTime([in] DATE fTime);
```

Get/set export till time

```
//C++ definition
HRESULT ICkStoredStreamExport::TillTime([out, retval] DATE* pfTime);
HRESULT ICkStoredStreamExport::TillTime([in] DATE fTime);
```

Following partial example will export first stored stream for 'Entrance2' device upon connected server event.

```
C# example:
```

```
static void cKit_OnServerConnected ( ICkServer pServer, int nStatus ) {

    try {
        ICkStoredStreamExport export = null;

        foreach(ICkMediaDevice mediaDevice in (object[])pServer.GetMediaDevices()) {
            if(mediaDevice.Title != "Entrance2")
                continue;
            foreach(ICkStoredStream storedStream in (object[])mediaDevice.GetStoredStreams()) {

                DateTime sts_Start = new DateTime(),
                            sts_End = new DateTime();

                storedStream.GetBoundaries(out sts_Start, out sts_End);

                /* AVI / WMV settings can be set by casting ICkStoredStreamExport
                   to ICkStoredStreamExportAvi or ICkStoredStreamExportWmv.
                   use CKEXPORTTYPE.CKEXPORTTYPE_AVI for AVI format */
                export = storedStream.Export(CKEXPORTTYPE.CKEXPORTTYPE_WMV);

                export.Path = @"C:\tmp\file.avi";
                export.SinceTime = sts_Start;
                export.TillTime = sts_End;

                /* Configure subtitle color and position */
                export.SubtitleType = CKSUBTITLETYPE.CKSUBTITLETYPE_HARD;
                var subs = (ICkStoredStreamExportSubtitles)_export;
                subs.BackgroundColor = (uint)System.Drawing.ColorTranslator.ToWin32(Color.Black);
                subs.ForegroundColor = (uint)System.Drawing.ColorTranslator.ToWin32(Color.White);
                subs.Position = CKSUBTITLEPOSITION.POSITION_TOPLEFT;
                subs.TextFormat = "$(ClientDateTime)";

                export.Start();
                break;
            }
        }
    } catch { }
}
```

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
```

Two-way audio

Two-way audio is implemented in DVRClientKit in the following way:

- DVRClienKit offers the possibility to retrieve the live audio from a camera by subscribing to the OnAudioSample event
- DVRClienKit offers the possibility to push audio samples to a camera by using the ICkAudioTransmitter interface

The retrieval is similar to the live video stream retrieval - the client application implements an event handler for the **`_DVRClientKitEvents::OnAudioSample([in] ICkMediaDevice* pMediaDevice, [in] ICkSample* pSample)`** event and subscribes to the event. The audio frame is passed as an ICkSample object in the event handler parameters. Through this interface the client application can retrieve the audio buffer and media type (CKMEDIATYPE_PCM - PCM, CKMEDIATYPE_PCMU - PCMU (Mu-Law), CKMEDIATYPE_PCMA - PCMA (A-Law)). It is also possible to cast the ICkSample object to a ICkAudioSample object to retrieve audio specific parameters like rate, channel count and bits per sample.

The audio transmission works by acquiring a ICkAudioTransmitter object from the desired media device by calling **`ICkMediaDevice::GetAudioTransmitter()`** and then pushing data to the camera through this object. This can be done either by sending CCkAudioSample objects to the device via calls to **`ICkMediaDevice::PushSample([in] ICkSample* pClientKitSample)`** that accepts PCMA/PCMU/PCM audio or by pushing raw binary audio data via calls to **`ICkMediaDevice::PushData([in] BYTE* pData, [in] ULONG nDataSize, [in] DATE fTime)`**, but then it must be explicitly formatted as 16 Bit 8000 Hz mono PCM audio. Before sending data the client application must explicitly initialize a transmission session (this helps to preserve the internal resources of the transmitter, such as memory for the sample pool to be allocated only during a session) and close the session when it is no longer needed by calling **`ICkMediaDevice::StartTransmission([in] ULONG nTimeToHold, [in] ULONG nSamplePoolSize)`** and **`ICkMediaDevice::StopTransmission()`**. The parameters for the **`ICkMediaDevice::StartTransmission([in] ULONG nTimeToHold, [in] ULONG nSamplePoolSize)`** call denote the desired maximum time in milliseconds for one audio buffer and the number of samples to preallocate. If the audio samples pushed will be bigger than the buffer length in the allocated pool then an error will occur. Default value 0 can be used in both arguments to use default values (50 samples with 250 milliseconds per sample). If samples are pushed prior to initialization of a transmission session or a new session is initialized while an existing one is running an error will be returned.

The CCkAudioDecoder class is meant to provide audio decoding capabilities between PCMA/PCMU and PCM formats. The ICkAudioDecoder interface implemented by the CCkAudioDecoder class has a **`ICkAudioDecoder::Decode([in] CKMEDIATYPE OutputMediaType, [in] IUnknown*`** `pSampleUnknown, [in] IUnknown* pDecodedSampleUnknown)` method that takes an input and output interfaces of ICkAudioSample (CCkAudioSample class objects implement this interface) and perform decoding from PCMA/PCMU to PCM audio. This can be useful if the audio from the media device is in PCMU/PCMA format and the client application can explicitly handle only PCM.

Interfaces

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 pSampleUnknown 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* pSampleUnknown)

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* pSampleUnknown – Video sample data object. DVRServerTools::ISample interfaces could be queried from pSampleUnknown 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

HRESULT OnAudioSample([in] ICkMediaDevice* pMediaDevice, [in] ICkSample* pSample)

Remarks:
Raises when the media device (ICkMediaDevice) has received a new audio sample from Luxriot DVR Server.
Parameters:
[in] ICkMediaDevice* pMediaDevice – Media device which received the audio sample.
[in] ICkSample* pSample - Audio sample object, from which an ICkAudioSample interface can be acquired to retrieve audio relevant information about the sample.

Interface ICkServer

Methods:

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

Returns:
BSTR - Unique identifier of server (for example: "Network (IP) Camera\001")

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).

HRESULT CreateMediaDevice([in] BSTR sDeviceAddress, [in] ULONG nDevicePort, [in] BSTR sDeviceUsername, [in] BSTR sDevicePassword, [in] BSTR sDeviceTitle, [in] BSTR sDeviceVendor, [in] BSTR sDeviceModel, [out, retval] ICkMediaDevice **ppMediaDevice);

todo

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] HADNLE 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 CSreen 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_MEDIADEVICE_STREAMTYPEFLAG_ALL) call.

HRESULT StopLiveView()

Description:
Stops live video playback

HRESULT StartStream([in] DVRCLIENTKIT_MEDIADEVICE_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_MEDIADEVICE_STREAMTYPEFLAGS nFlags – Stream flags to receive events. Can be one or more of the following:
```
typedef enum DVRCLIENTKIT_MEDIADEVICE_STREAMTYPEFLAGS
{
DVRCLIENTKIT_MEDIADEVICE_STREAMTYPEFLAG_NONE = 0,
DVRCLIENTKIT_MEDIADEVICE_STREAMTYPEFLAG_LIVESTREAM = (1 << 0),
DVRCLIENTKIT_MEDIADEVICE_STREAMTYPEFLAG_MOTIONSTREAM = (1 << 1),
DVRCLIENTKIT_MEDIADEVICE_STREAMTYPEFLAG_ALL =
(DVRCLIENTKIT_MEDIADEVICE_STREAMTYPEFLAG_LIVESTREAM |
DVRCLIENTKIT_MEDIADEVICE_STREAMTYPEFLAG_MOTIONSTREAM),
} _DVRCLIENTKIT_MEDIADEVICE_STREAMTYPEFLAGS;
```

HRESULT StopStream([in] DVRCLIENTKIT_MEDIADEVICE_STREAMTYPEFLAGS nFlags)

Description:
Stops previously started stream.
Parameters:
[in] DVRCLIENTKIT_MEDIADEVICE_STREAMTYPEFLAGS nFlags – Flags of streams you no longer wish to receive. Can contain one or more of the following:
```
typedef enum DVRCLIENTKIT_MEDIADEVICE_STREAMTYPEFLAGS
{
DVRCLIENTKIT_MEDIADEVICE_STREAMTYPEFLAG_NONE = 0,
DVRCLIENTKIT_MEDIADEVICE_STREAMTYPEFLAG_LIVESTREAM = (1 << 0),
DVRCLIENTKIT_MEDIADEVICE_STREAMTYPEFLAG_MOTIONSTREAM = (1 << 1),
DVRCLIENTKIT_MEDIADEVICE_STREAMTYPEFLAG_ALL =
(DVRCLIENTKIT_MEDIADEVICE_STREAMTYPEFLAG_LIVESTREAM |
DVRCLIENTKIT_MEDIADEVICE_STREAMTYPEFLAG_MOTIONSTREAM),
} _DVRCLIENTKIT_MEDIADEVICE_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)

Descirption:
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

HRESULT GetAudioTransmitter([out, retval] ICkAudioTransmitter ppAudioTransmitter)**

Description:
This function returns an audio transmitter object that can be used to send audio to the device. If the device does not support audio input or it is not enabled on the server the transmitter will not be created.
Returns:
ICkAudioTransmitter - object that can be used to push audio data to the device.

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, otherwsie 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_UPLEFT,
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 plaback.
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 CSreen 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 Pause()

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 witihn archive boundaries

HRESULT Export([in] _CKEXPORTTYPE ExportType, [out, retval] ICkStoredStreamExport ppStoredStreamExport)**

Description:
Exports archive video in specified format.

CKEXPORTTYPE - enumeration used to set exported file format:
```
typedef enum CK_PTZCONTROLCAPABILITYFLAGS
{
CKEXPORTTYPE_AVI = 1,
CKEXPORTTYPE_WMV = 3,
}
```

Interface ICkAudioSample

Methods:

HRESULT Initialize([in] CKMEDIATYPE nMediaType, [in] USHORT nChannels, [in] ULONG nRate, [in] USHORT nBitsPerSample)

Description:
This function helps to initialize an audio sample.
Parameters:
[in] CKMEDIATYPE nMediaType — DVRClientKit media type for the audio sample (CKMEDIATYPE_PCM - PCM, CKMEDIATYPE_PCMU - PCMU (Mu-Law) or CKMEDIATYPE_PCMA - PCMA (A-Law)).
[in] USHORT nChannels — Channel count for the audio sample, currently only mono audio is supported.
[in] ULONG nRate - Rate in Hz for the audio sample, currently only 8000 Hz audio is supported.
[in] USHORT nBitsPerSample - Bit count for one audio sample.

Properties:

USHORT Channels - Audio channel count, e.g. 1 for mono.
ULONG Rate - Audio rate in Hz, e.g. 8000 for 8000 Hz PCM.
USHORT BitsPerSample - Size of one audio sample, e.g. 16 for 16 Bit PCM.

Interface ICkAudioDecoder

Methods:

HRESULT Decode([in] CKMEDIATYPE OutputMediaType, [in] IUnknown* pSampleUnknown, [in] IUnknown* pDecodedSampleUnknown)

Description:
Allows decoding 8 Bit 8000 Hz mono PCMU/PCMA to 16 Bit 8000 Hz mono PCM audio sample.
Parameters:
[in] CKMEDIATYPE OutputMediaType — type of the decoded sample, currently only CKMEDIATYPE_PCM is supported.
[in] IUnknown* pSampleUnknown — an audio sample in 8 Bit 8000 Hz mono PCMA/PCMU format, should be a CCkAudioSample object.
[in] IUnknown* pDecodedSampleUnknown - an object that will receive the decoded sample, should be a CCkAudioSample object.

Interface ICkAudioTransmitter

Methods:

HRESULT StartTransmission([in] ULONG nTimeToHold, [in] ULONG nSamplePoolSize)

Description:
Initializes a new audio transmission session. With this call the client can control how much resources are allocated in the transmitter to store samples, this could be important in a scenario where multiple transmitters are created in the same time to reduce memory consumption. After a session has been initialized calls to PushSample and PushData can be made. A session should be stopped when no longer needed to deallocate the internal resources used by the transmitter. The function will fail if a session is already started.
Parameters:
[in] ULONG nTimeToHold — Time in milliseconds that one audio buffer should be able to hold in range [0-2500], if 0 is passed a default value of 250 is used.
[in] ULONG nSamplePoolSize — Number of audio buffers to allocate in range [0-100], if 0 is passed a default value of 50 is used.

HRESULT StopTransmission()

Description:
Call to this function stops the current transmission session and deallocates internal resources, audio data cannot be sent without initializing a new session. Fails if a session has not been started.

HRESULT PushSample([in] ICkSample* pClientKitSample)

Description:
Sends an audio sample to the media device. This function accepts PCM, PCMU or PCMA samples and transcodes PCMA/PCMU to PCM internally. 8000 Hz mono audio is expected. The function will fail if a session is not started or the sample size is larger than the one specified during the initialization of the session or the media type of the sample is not supported.
Parameters:
[in] ICkSample* pClientKitSample — The audio sample to be sent, should be a CCkAudioSample object.

HRESULT PushData([in] BYTE* pData, [in] ULONG nDataSize, [in] DATE fTime)

Description:
Sends raw binary data as an audio sample to the media device. The binary data should be in 16 Bit 8000 Hz mono PCM format. The function will fail if a session is not started or the sample size is larger than the one specified during the initialization of the session.
Parameters:
[in] BYTE* pData — Pointer to the raw audio data.
[in] ULONG nDataSize — Size of the raw audio data in bytes.
[in] DATE fTime - Timestamp for the audio data.