UDVGuard
Software Development Kits (SDK)

Version: 2.03
Language: English
Date: 2011/01/31

Introduction:

UDVGuard.exe is an ActiveX component including OLE Automation interface. For each DoorPhone must be created the new instance of this component.

The destination application can use this component in a visible or invisible mode. The UDVGuard provides access to the DoorPhone for the main application.

The UDVGuard component can execute several functions as stated below through network connecting:

- To receive videodata from built-in camera in the DoorPhone
- Calling with the DoorPhone
- Open the door lock on the DoorPhone
- Light On on the DoorPhone
- Creating records of video and audio data
- Creating snap-shots
- ... etc.

Installation:

As the first step, UDVGuard.exe component needs to register in Windows registers.
It is possible for example from the command line using parameter:

UDVGuard.exe /regserver

Unregistering this component can be done as follows:

UDVGuard.exe /unregserver

Creating instance of UDVGuard:

variable type
Fvar: OleVariant;

create instance:
Fvar:= CreateOleObject('UDVGuard.UDVServer');

free instance:
Fvar:= Unassigned;

Connecting to the DoorPhone:

Connect instance to the DoorPhone:
procedure Fvar.Connect(RemoteIP: string; RemotePort:string);

Disconnect instance:
procedure Fvar.DisConnect;

Get the name of the DoorPhone:
property Fvar.Name: string; //read Get_Name, if instance not connected, return '???'

Visible-invisible mode:

Show window of instance:
property Fvar.Visible: boolean; //read Get_Visible write Set_Visible

Minimize window:
procedure Fvar.MinimizeWindow; //window minimized
procedure Fvar.RestoreWindow; //restored window

Set position on the screen:
property Fvar.Left: integer; //read Get_Left; write Set_Left
property Fvar.Top: integer; //read Get_Top; write Set_Top

Relay on the DoorPhone:

to change status:

property Fvar.Rele1:Boolean; //read Get_Rele1; write Set_Rele1 – Key
property Fvar.Rele2:Boolean; //read Get_Rele2; write Set_Rele2 - Light

when status of any relay has been changed, raise event:
procedure IUDVServerEvents.OnRele(const DestIP: WideString; Rele: Integer; Active: WordBool); dispid 202;
example: see **Events**

Call:

property Fvar.CallStatus:Integer; //read Get_CallStatus; write Set_CallStatus
type TCallStatus = (CALL_OFF = 0, CALL_RING = 1, CALL_ON = 2);

when an incomming call is active, raise event:
procedure IUDVServerEvents.OnIncommingCall(const DestIP: WideString); dispid 201;

when call status is changed, raise event:
procedure OnChangeCallStatus(const DestIP: WideString; Status: Integer); dispid 205;

example: see **Events**

Capturing:

procedure Fvar.CaptureToJPEG(FileName: string);

Videorecording to *.AVI file:

When any call is open and *AVIRecording* is set to "*true*", then *Fvar* instance starts record of video to file. When this call is closed, recording is stopped, avi file created and raise event *OnAVIReady*.

to set destination file path:
property Fvar.AVIPath:WideString; //read Get_AVIPath; write Set_ AVIPath

to start/stop recording
property Fvar.AVIRecording:WordBool; //read Get_AVIRecording; write Set_ AVIRecording

when a new AVI file is ready, raise event:
procedure IUDVServerEvents .OnAVIReady(const DestIP: WideString; const FileName: WideString); dispid 208;

Test functions:

procedure Fvar.SoundTest;

Video transfer:

when a new videosnapshot is ready, raise event:
procedure IUDVServerEvents.OnVideo(const DestIP: WideString); dispid 204;

example: see **Events**

to take this videosnapshot:
property Fvar.VideoStream:OLEVariant; //read Get_VideoStream; write Set_VideoStream

example:

```
V: OleVariant;
L: integer;
P: pointer;
JPEGStream: TMemoryStream;
JPEGImage: TJPEGImage;
Image: TImage;

V:= Fvar.VideoStream;
L:= VarArrayHighBound(V,1) - VarArrayLowBound(V,1) + 1;
P:=VarArrayLock(V);
JPEGStream.Clear;
try
```

```
   JPEGStream.WriteBuffer(P^,L);
   JPEGStream.Seek(0,0);
   JPEGImage.LoadFromStream(JPEGStream);
   Image.Picture.Assign(JPEGImage);
   Image.Refresh;
 finally
  VarArrayUnlock(V);
 end;
```

Debugging:

when anything happens, UDVGuard sends debug log and raises an event:
procedure IUDVServerEvents.OnLog(const DestIP: WideString; Message:WideString); dispid 203;

example: see **Events**

Events

example of processing of events from the UDVGuard:

```
type
 TEventSink = class(TObject, IUnknown, IDispatch)
  private
   { IUnknown }
   function QueryInterface(const IID: TGUID; out Obj): HResult; stdcall;
   function _AddRef: Integer; stdcall;
   function _Release: Integer; stdcall;
   { IDispatch }
   function GetTypeInfoCount(out Count: Integer): HResult; stdcall;
   function GetTypeInfo(Index, LocaleID: Integer; out TypeInfo): HResult; stdcall;
   function GetIDsOfNames(const IID: TGUID; Names: Pointer;
     NameCount, LocaleID: Integer; DispIDs: Pointer): HResult; stdcall;
   function Invoke(DispID: Integer; const IID: TGUID; LocaleID: Integer;
     Flags: Word; var Params; VarResult, ExcepInfo, ArgErr: Pointer): HResult; stdcall;
  public
  end;

{ TEventSink.IUnknown }

function TEventSink._AddRef: Integer;
begin
  // No need to implement, since lifetime is tied to client
  Result := 1;
end;

function TEventSink._Release: Integer;
begin
  // No need to implement, since lifetime is tied to client
  Result := 1;
```

```
end;

function TEventSink.QueryInterface(const IID: TGUID; out Obj): HResult;
begin
  // First look for my own implementation of an interface
  // (I implement IUnknown and IDispatch).
  if GetInterface(IID, Obj) then
    Result := S_OK
  // Next, if they are looking for outgoing interface, recurse to return
  // our IDispatch pointer.
  else if IsEqualIID(IID, IUDVServerEvents) then
    Result := QueryInterface(IDispatch, Obj)
  // For everything else, return an error.
  else
    Result := E_NOINTERFACE;
end;

{ TEventSink.IDispatch }

function TEventSink.GetIDsOfNames(const IID: TGUID; Names: Pointer;
  NameCount, LocaleID: Integer; DispIDs: Pointer): HResult;
begin
  Result := E_NOTIMPL;
end;

function TEventSink.GetTypeInfo(Index, LocaleID: Integer;
  out TypeInfo): HResult;
begin
  Pointer(TypeInfo) := nil;
  Result := E_NOTIMPL;
end;

function TEventSink.GetTypeInfoCount(out Count: Integer): HResult;
begin
  Count := 0;
  Result := S_OK;
end;

function TEventSink.Invoke(DispID: Integer; const IID: TGUID;
  LocaleID: Integer; Flags: Word; var Params; VarResult, ExcepInfo,
  ArgErr: Pointer): HResult;
var
  V1,V2,V3: OleVariant;
begin
  Result := S_OK;
  case DispID of
    201:
      begin
       V1 := OleVariant(TDispParams(Params).rgvarg^[0]);
       Form3.IncommingCall(V1); //V1 = DestIP: string;
      end;
    202:
```

```
    begin
      V1 := OleVariant(TDispParams(Params).rgvarg^[0]);
      V2 := OleVariant(TDispParams(Params).rgvarg^[1]);
      V3 := OleVariant(TDispParams(Params).rgvarg^[2]);
      Form3.ReleStatus(V3,V2,V1); // V3 = DestIP: string; V2 = Rele: integer (1-Key, 2-Light);
                                                  V1 = Active: boolean
    end;
   203:
    begin
      V1 := OleVariant(TDispParams(Params).rgvarg^[0]);
      V2 := OleVariant(TDispParams(Params).rgvarg^[1]);
      Form3.Log(V2,V1); //V2 = DestIP: string; V1 = Message: string
    end;
   204:
    begin
      V1 := OleVariant(TDispParams(Params).rgvarg^[0]);
      Form3.VideoPresent(V1); //V1 = DestIP: string
    end;
   205:
    begin
      V1 := OleVariant(TDispParams(Params).rgvarg^[0]);
      V2 := OleVariant(TDispParams(Params).rgvarg^[1]);
      Form3.ChangeCallStatus(V2,V1);
    end;
  end;
end;
```