CAPI 2.0
Common ISDN API function declarations for VisualBasic

Contents:

1 - CAPI2032.DLL calls:

Here are the main functions from the 32 bit CAPI2.0 DLL file. The file (capi2032.dll) should be in your main windows directory or in the windows system directory. It is manufacturer specific, and is included with the drivers of your ISDN card. Don't use any other DLL!
To start programming, simply put the listed function imports in the declarations section of the main form or module of your Visual Basic project. Remember, these functions work only with Win95/98/NT! For 16 bit Windows, OS/2 or DOS calls, see part 2 of the CAPI specs.

CAPI_INSTALLED:

Public Declare Function **CAPI_INSTALLED** Lib "Capi2032.dll" () As Long

Returns 0x0000 if the CAPI 2.0 is installed.

CAPI_REGISTER:

Public Declare Function **CAPI_REGISTER** Lib "Capi2032.dll" (ByVal *MessageBuffersize* As Long, ByVal *maxLogicalConnection* As Long, ByVal *maxBDataBlocks* As Long, ByVal *maxBDataLen* As Long, ByRef *AppID* As Long) As Long

Registers your application by returning an individual application ID in the variable AppID. Call this function before sending or retrieving any messages.

CAPI_RELEASE:

Public Declare Function **CAPI_RELEASE** Lib "Capi2032.dll" (ByVal *AppID* As Long) As Long

Release the application that has the application ID AppID.

CAPI_GET_PROFILE:

Public Declare Function **CAPI_GET_PROFILE** Lib "Capi2032.dll" (ByRef *CapiProfile* As Any, ByVal *CtrlNr* As Integer) As Long

Gets the supported protocols, manufacturer specific information, number of BChannels etc. If CtrlNr is 0 on function call then the total number of installed controllers/ISDN cards is returned in the CapiProfile struct, which has to be coded according to the CAPI2.0 specs (see end of document).

CAPI_GET_SERIAL_NUMBER:

Public Declare Function **CAPI_GET_SERIAL_NUMBER** Lib "Capi2032.dll" (ByRef *szBuffer* As Any) As Long

Be sure to reserve plenty of space in the szBuffer struct to avoid a General Protection Fault. The struct needs to contain just one string, i.e.

```
Public Type szBuffer
  serialnum As String *255
End Type
```

will do nicely.

CAPI_GET_VERSION:

Public Declare Function **CAPI_GET_VERSION** Lib "Capi2032.dll" (ByRef *CAPIMajor* As Long, ByRef *CAPIMinor* As Long, ByRef *ManufacturerMajor* As Long, ByRef *ManufacturerMinor* As Long) As Long

Returns the version of the CAPI (should be 2.0) and the manufacturer specific release version.

CAPI_SET_SIGNAL:

Public Declare Function **CAPI_SET_SIGNAL** Lib "Capi20.dll" (ByVal *AppID* As Integer, ByRef CallBackFunction, CBF_specific_param As Long) As Long

This function is defined only in the 16 bit CAPI20.DLL. The callback function is of the form CallBackFunction (*AppID* As Integer, *Param* As Long) and as a c/c++ void, i.e. has no return value. The CAPI calls the specified CB function always when a message is available. Not sure if the CBF will work in Visual Basic.

CAPI_WAIT_FOR_SIGNAL:

Public Declare Function **CAPI_WAIT_FOR_SIGNAL** Lib "Capi2032.dll" (ByVal *AppID* As Long) As Long

This function is defined only in the 16 bit CAPI20.DLL. The callback function is of the form CallBackFunction (*AppID* As Integer, *Param* As Long) and as a c/c++ void, i.e. has no return value. The CAPI calls the specified CB function always when a message is available. Not sure if the CBF will work in Visual Basic.

CAPI_PUT_MESSAGE:

Public Declare Function **CAPI_PUT_MESSAGE** Lib "Capi2032.dll" (ByVal *AppID* As Long, *pcapimessage* As Any) As Long

A return value other than 0 indicates an error according to errors defined in class &H11xx.

CAPI_GET_MESSAGE:

Public Declare Function **CAPI_GET_MESSAGE** Lib "Capi2032.dll" (ByVal *AppID* As Long, ByRef *lpCapiBuffer* As Long) As Long

Return value other than 0 or &H1104 ("Message queue empty") indicates an error. After calling, the lpCapiBuffer long value contains the pointer to the message

To get access to the data, use

Call **RtlMoveMemory** (*MessageHeader*, ByVal *lpCapiBuffer*, ByVal Len(*MessageHeader*))

with MessageHeader as the standard message header. After checking MessageHeader.Command and MessageHeader.SubCommand you can copy the rest of the message to the correct structure, t.ex.

Call **RtlMoveMemory** (*AlertConf*, ByVal (*lpCapiBuffer*+Len(*MessageHeader*)), ByVal (*MessageHeader.Length* - Len(*MessageHeader*))

RtlMoveMemory:

Public Declare Sub **RtlMoveMemory** Lib "Kernel32.Dll" (*RecieveStruct* As Any, ByVal *lpCapiBuffer* As Long, ByVal *iMaxLength* As Long)

Public Declare Sub **RtlMoveMemoryPtr** Lib "Kernel32.Dll" Alias "RtlMoveMemory" (ByVal DestPtr As Long, ByVal SourcePtr As Long, ByVal length As Long)

RecieveStruct is the VB Type structure which will receive the data. lpCapiBuffer is the pointer to the data source. The alternative function RtlMoveMemoryPtr uses a pointer for both the target and the source buffer.

Use RtlMoveMemory to move the data from the memory location specified by, for example, the lpCapiBuffer variable of CAPI_GET_MESSAGE to a user defined structure (VB Type), moving iMaxLength bytes.
Tip: on receiving the message, move the first 8 bytes into a standard message header, then read the command and subcommand from it and copy the rest of the data into the structure corresponding to the message, i.e. if command was CAPI_LISTEN and subcommand was CAPI_CONF then move the remaining data into your ListenConf structure.

2 - Basic usage:

For an example on how to use these functions, you can some CAPI G3 fax and phone ActiveX OCX source code from my homepage's ISDN/CAPI section at http://www.hut.fi/~jwagner/CTHP/ . To start with, you should have a look at the subroutine Get_Capi_Message() or Fax_Capi_Message() in the file "CapiCtrl1.ctl". The first lines are the important ones, i.e. calling the CAPI_GET_MESSAGE, copying the header data with RtlMoveMemory and then checking which type of message it is, and the copying the rest of the data into the according Visual Basic structure.

To see what the various messages that are sent and received actually look like, open the "capidef.bas" file. Don't forget to download the complete CAPI2.0 specs (from www.capi.org) if you want to find the full description of all message structures.

The message exchange basically works like this: you sent a *request* message via CAPI_PUT_MESSAGE and on your next CAPI_GET_MESSAGE call the CAPI returns you a *confirmation* message. If you have sent the CAPI a request to monitor something (like incoming calls), you'll at some point get an *indication* message, whenever the event occurs that you told CAPI to monitor. The *indication* message contains some information about the event. You'll have to reply to the message with a *response* message.
That is,

application sends xxx_REQ => CAPI returns xxx_CONF
CAPI sends xxx_IND => application returns xxx_RESP

Download the official CAPI2.0 specs from www.capi.org. Part 1 would be the most important one for beginners. If you don't have a MS Windows operating system, you'll find the right capi20 function declarations for your OS in part 2. A CAPI for Linux is also well on its way, AFAIK it is already part of the ISDN4Linux package (but for now supports only passive ISDN cards).
Ok then good luck and happy programming!

3 - Some basic CAPI 2.0 message structures:

CapiProfile Struct

```
'/* here's the CAPI profile structure to pass to the CAPI_GET_PROFILE function */
Public Type CapiProfile

        NumCtrl As Integer          ' controllers on S0 bus

        NumBChannels As Integer     ' usually returns 2 b-channels
        GlobalOptions As Long       ' bit0=internal contrl. support, bit1=extern. supp.,
                                    ' bit2=handset supp., bit3=DTMF supp.

        B1ProtocolSupp As Long      ' bit0-6: 64kBit HDLC, 64kBit transparent, V.110 async,
                                    ' V.110 sync HDLC, T.30, 64kBit inv HDLC,
                                    ' 56kBit transparent HDLC
        B2ProtocolSupp As Long      ' … see CAPI2.0 specs at www.capi.org
        B3ProtocolSupp As Long      ' … see CAPI2.0 specs at www.capi.org

        i1 As Long                  ' additional information that
        i2 As Long                  ' is manufacturer dependend
        i3 As Long
        i4 As Long
        i5 As Long
        i6 As Long
        manuf1 As Long
        manuf2 As Long
        manuf3 As Long
        manuf4 As Long
        manuf5 As Long

        safetybuffer(64) As Byte    ' in case the CAPI expected more space, to
                                    ' prevent a General Protection Fault
End Type
Public CapiProfile As CapiProfile
```

MessageHeader Struct

```
'/* here's the standard message header to use with the CAPI_GET_MESSAGE */
Public Type MessageHeader
        Length As Integer           ' length of the complete message
        Appid As Integer            ' ID of your application
        Command As Byte             ' command like CAPI_CONNECT
        SubCommand As Byte          ' one of the subcommands like CAPI_IND
        Message_Number as Integer
End Type
Public MessageHeader As MessageHeader
```