General/Introduction

The pad - dll is freeware and serves as a unified driver to connect with StepOver's variety of electronic signature pads (bluePads, blueMPads, LE Pads and +Pads). The dll either accesses the pad via a standard serial com port or a virtual com port. You may use this DLL directly or via the freeware active X interface StepOverSignatureDevice1. We recommand that you use the Active X interface because it is much easyer to handle and has more functionality such as compression and decompression of the biodata.

StepOverSignatureDevice1 ActiveX Version 1.2 Documentation

Name: StepOverSignaturedevice1
StepOverSignatureDevice1 library GUID {B23ECB86-5295-4DA7-BCBC-E564AB5A7334}
Dispatch-Interface for StepOverSignatureDevice-Element GUID {1899784F-C6D5-4D86-869E-21EBC9D1882C}
StepOverSignatureDevice Element GUID {F5C74079-C195-4DA9-B60D-F6F5365042D1}

Datastructures/Types
The methods listed below are declared in Pascal/Delphi but they are OLE and COM conform.
The type „**widestring**“ is compatible to the COM-Type BSTR and represents a dynamically assigned number of 16-BIT Unicode Chars. It's size is up to 2GB.
„**Wordbool**“ is a 2 byte variable who is true then it's <>0.
„**safecall**“ is used to implement „Exeption-Firewalls“ and windows COM-error messaging handling.

Methods

procedure startRead; safecall;
call this procedure to activate the comport and recive the signature. Be sure that you have called setPadSettings first.

procedure stopRead; safecall;
Closes the comport. call this procedure after the you have called startRead and after the user has fineshed to sign.

procedure getPadSettings(var oldSettings, certStr: widestring); safecall;
Opens the configuration window of the driver there the user can choose its pad and comport. Store the old settings of the pad in „oldSettings“. If the user clicks OK you will get the new settings back in the variable oldSettings. If the user clicks cancel you will get the old settings back in oldSettings. If you call the procedure the first time use the followings string in oldSettings: „000000000000000000no pad“ don't leve the variable empty.Read more under „StepOver PadSettings“. The variable certStr contains the certificate of the application which tries to use this procedure. Read more unter „StepOver certificate“.

procedure setPadSettings(padSetting, certStr: widestring); safecall;
Call this procedure before you call StartRead the first time after you have inizialised the active X. PadSetting contains the pad type, comport and pad ID (if needed). Use getPadSettings to get that settings. Read more under „StepOver PadSettings“. The variable certStr contains the certificate of the application which tries to use this procedure. Read more unter „StepOver certificate“.

procedure getBiodataString(var Biodata:widestring); safecall;
This procedure returns the uncompressed biodata in the variable Biodata and clears all biodata in memory. Be sure to call it after stopRead. Read more unter „StepOver Biodata“. You may call getBiodataString with an empty variable. Attention - use this procedure only if you don't want to store the bio data. It is recommanded that you use saveBiodata.

procedure saveBiodata(var Biodata: WideString); safecall;
This procedure returns the compressed biodata in the variable Biodata and clears all biodata in memory. Be sure to call it after stopRead.. You may call getBiodata with an empty variable. If you need the uncompressed bio data later you can call decompressBiodata to decompress them.

procedure decompressBiodata(var Biodata: WideString); safecall;
Call this procedure with the compressed biodata in Biodata, which you got from a file or from the procedure saveBiodata, and it will return the uncompressed biodata in the variable Biodata. Read more unter „StepOver Biodata“.

property width:integer;
property height:integer;
property left:integer;
property top:integer;
Use this properties to define the size and position of the signature field during runtime. But be sure to do that before you call startRead.

procedure setBGPic(const pic, ext: WideString); safecall;
You can use this procedure to set a background picture in the signing area. Be sure to resize the signing area to the size of your BGPicture before. Pic contains the path to the picture. Ext contains the extension of the picture (additional to the one in the file name and without the dot). That means if you set the variable pic to 'C:\test.bmp' write 'bmp' into ext. You can use bmp and tiff files by now.

function savePic(var pic: WideString; const ext: WideString;withBG: WordBool): WordBool; safecall;
Use this function to save the picture of the signature (without the background picture if set) to a file specified in pic. The variables withBG don't work, it's reserved for future use. If you write instr into the variable ext you will get the bitmap back in the variable pic as a string – The picture will not be saved to a file in that case. Important: call this before you call stopRead. StopRead is deleting the picture of the last signature.

procedure setLanguage(Language:widestring); safecall;
You can set the language of the driver by using this procedure. Default is english. You may use the following strings to set the Language:
D
GB
ES
PL
D=German, GB=English, ES=Spanish, PL= Polish.

procedure setColor(color: Integer); safecall;
Using this procedure you can set the color of the rectangle that will be displayed in the ActiveX window after startread. The default color is red.
0=red
1=blue
2=lime
3=white
4=black
5=green

function isPadavAvailable(const settings: WideString): WordBool; safecall;
Checks if the pad given in settings is connected and returns true if it is and false if not. Settings is the string containing the actual pad settings given by the method getPadSettings.

function sensorTouched: WordBool; safecall;
The result of sensorTouched is set to false when you call startRead and becomes true after the user has touched the sensor of his pad (e.g. the userer has started signing).

StepOver Pad - Dll Version 1.2 Documentation

General
Code written in Delphi/Pascal is displayed in green. Code written in C++ is displayed in red.

Units and include files
If you want to access sopad.dll directly from your application you may use our units and include files to do that easily.
For Delphi/Pascal use SOPad.pas
uses
SOPad
For C++ use SOPad.h and SOPad.c
#include "SOPad.c"
#include "SOPad.h"

Datastructures/Types
This section describes the datastructures used to interact with the dll in Pascal/Delphi syntax. However, it should be possible to use any other high level programming language capable of using dynamic link libraries (dlls).

PIDType = array[1..5] of Byte;
typedef unsigned char SOPAD_PIDType[5];
Pad ID Type: An array of five bytes used as the decryption key of biometric data sent by the pad. Also used in StopRead where it is not a decryption key but rather a globally unique identifier of the pad used.

```
TDataFrame = record
   x,
   y : SmallInt;
   z : Byte;
   FrameNumber : integer;
   LineNumber : Byte
Timecode: integer
end;
typedef struct
{
short x;
short y;
unsigned char z;
int FrameNumber;
unsigned char LineNumber;
int timecode;
} SOPAD_TDataFrame;
```

This type is used to store a data frame from the pad. x and y are the coordinates of a cartesian coordinate system having its origin in the upper left corner. z is the pressure data. FrameNumber is the value from a counter in the pad and gets incremented with every frame. LineNumber is a counter which stays the same as long as the pen is on the pad and is incremented whenever a new line is started. In a typical application, one would test if oldLineNumber = newLineNumber to find out if one should use LineTo(...) or MoveTo(...) for visualising the data. Timecode represents the msecs since the signature has started.

#define SOPAD_MAX_STRING_LENGTH 256 (C++ only)
To be kompatible to Delphi/Pascal shortstring.

tFrames= Array of tdataFrame; (Only in Delphi or Pascal)
This is a dynamic array of tdataFrame which is used by getBioData.

```
GMTStampType = record
   years : byte;
   months : byte;
   days : byte;
   hours : byte;
   minutes : byte;
   seconds : byte;
end;
typedef struct
{
unsigned char years;
unsigned char months;
unsigned char days;
unsigned char hours;
unsigned char minutes;
unsigned char seconds;
} SOPAD_GMTStampType;
```

This type is used to store the time stamp (GMT) of the signature and returned by StopRead indicating the time of the signature. (For those pads that have a built in real time clock).

TportList=array[1..256] of shortstring; (Only in Delphi or Pascal)
This type is used by enumPorts to list the available comports.

Methods

BOOL SOPAD_Initialize(); (C++ only)
Call this method to load the dll.

void SOPAD_Uninitialize(); (C++ only)
Call this function to free the dll.

function getPadSettings(oldSettings, certStr: shortstring):shortstring;
void SOPAD_getPadSettings(const char* szOldSettings, const char* szCertStrchar, char szNewSettings[SOPAD_MAX_STRING_LENGTH]);
Opens the configuration window of the driver there the user can choose his pad and comport. Store the old settings of the pad in „oldSettings“. If the user clicks OK you will get the new settings back in result. If the user clicks cancel you will get the old settings back in result. If you call the procedure the first time use the followings string in oldSettings: „000000000000000000no pad“ don't leve the variable empty.Read more under „StepOver PadSettings“. The variable certStr contains the certificate of the application which tries to use this procedure. Read more unter „StepOver certificate“.

procedure setPadSettings(padSetting, certStr: shortstring);
void SOPAD_setPadSettings(const char* szPadSettings, const char* szCertStr);
Call this procedure before you call StartRead the first time after you have inizialised the active X. PadSetting contains the pad type, comport and pad ID (if needed). Use getPadSettings to get that settings. Read more under „StepOver PadSettings“. The variable certStr contains the certificate of the application which tries to use this procedure. Read more unter „StepOver certificate“.
procedure checkCom(Com : Byte; var PadType : shortstring);
void SOPAD_checkCom(unsigned char nComPort, char szPadType[SOPAD_MAX_STRING_LENGTH]);
This procedure checks the (virtual) com port with the number given in Com and returnes the type of pad that is connected to that port as a string. It listens for up to 2 seconds on the serial port before it gives up. Note that this calls SetCom internally and therefore changes the com port that is to be used by the StartRead procedure. PadType is one of the following:', 'LEPadS', 'LEPadBT', 'LEPadUSB', '+PadS', '+PadBT', '+PadUSB', or 'bluePad'. You dan't have to use this procedure if you use getPadSettings and setPadsettings.

procedure setCom(Com: Byte);
void SOPAD_setCom(unsigned char nComPort);
This procedure sets the number of the (virtual) com port that is to be used by the StartRead procedure.
You don't have to use this procedure if you use getPadSettings and setPadsettings.

procedure setPadType(PadType: shortstring);
void SOPAD_setPadType(const char* szPadType);
This procedure sets the type of pad that is being used.
You don't have to use this procedure if you use getPadSettings and setPadsettings.

procedure getPID(var PadID: PIDType; certStr: shortstring);
void SOPAD_getPID(SOPAD_PIDType pPadID, const char* szCertStr);
This procedure blocks until the pad sends it's identifier which is used for encryption and decryption of the data. Those pads which support this feature have a small button that needs to be pressed for several seconds, so the ID is sent. CertString is the certificate of the application which tries to use this procedure. Read more under „StepOver Certificate“. You dan't have to use this procedure if you use getPadSettings and setPadsettings.

procedure setPID(PadID: PIDType; certStr: shortstring);
void SOPAD_setPID(const SOPAD_PIDType pPadID, const char* szCertStr);
This procedure sets the identifier which is used for encryption and decryption of the data. CertString is the certificate of the application which tries to use this procedure. Read more under „StepOver Certificate“. You dan't have to use this procedure if you use getPadSettings and setPadsettings.

procedure startRead;
void SOPAD_startRead();
This procedure opens the (virtual) serial port which was set by SetCom and turns on the power for the pad by setting the DTR line to high.

procedure stopRead(var PadID: PIDType; var TimeStamp : GMTStampType);
void SOPAD_stopRead(SOPAD_PIDType pPadID, SOPAD_GMTStampType* pTimeStamp);
This procedure closes the port to the pad. Also, for those pads that support this feature, it returns a globally unique identifier (which is not the same as the PadID used for decryption) and the time of the signature.

procedure getBioData(var PadID : PIDType; var GMTS : GMTStampType; var nFrames : integer; var Data: tFrames); (Only in Delphi or Pascal)
(C++ users use SOPAD_framesAvailable and SOPAD_getFrame)
This procedure returns a globally unique identifier (which is not the same as the PadID used for decryption) and the time of the signature (for those pads that support this feature). nSamples is the number of frames. Data points to dynamic array filled with TDataFrame's. Call clear biodata after you got the biodata with getBioData. Otherwise you will get the old biodata too if you call getBioData next time.

procedure clearBioData;
void SOPAD_clearBioData();
This procedure clears the biometrical data from the memory. It also frees the memory which GetBioData allocated for its Data, so be sure not to access this any more and to set the dynamic array which you got in the variable Data from getBioData to NIL before calling clearBioData.

function framesAvailable : Boolean;
BOOL SOPAD_framesAvailable();
This function returns true if new frames are available. It doesn't block.

procedure getFrame(var Frame: TdataFrame);
void SOPAD_getFrame(SOPAD_TDataFrame* pFrame);
This procedure returns the next frame in the queue if there's one available, otherwise the parameter remains unchanged.

procedure registerCanvas(canvas : TCanvas; xoffs, yoffs, width, height : Integer);
void SOPAD_registerCanvas(void* pCanvas, int xoffs, int yoffs, int width, int height);
With this procedure one can register a TCanvas on which the signature is drawn in real time using antialiasing for smoother appearance. The upper left corner and the size of the rectangle in which the signature appears can be specified with the xoffs, yoffs, width and height parameters. Note that the dll hooks into the OnChange event of the canvas for improved presentation. To avoid problems, the canvas should only be accessed after calling its Lock method as long as it is register with the dll.

procedure unRegisterCanvas;
void SOPAD_unRegisterCanvas();
This procedure unregisters the currently registered Tcanvas.

Procedure setLang(Lang:shortstring);
void SOPAD_setLang(const char* szLang);
You can set the language of the driver by using this procedure. Default is english. You may use the following strings to set the Language:
D
GB
ES
PL
D=German, GB=English, ES=Spanish, PL= Polish.

procedure enumPorts(var PortList: Tportlist); (Only in Delphi or Pascal)
Handle an variable of the type tportlist (discribed above) to this procedure. EnumPorts writes all available comports into Portlist.
Available means that the comport must exist (including virtual comports) and that the comport is not occupied by an other application.
For example:
You have 3 comports at your computer. Com1 is a real serial port but occupied by your modem driver wich is currently online. Com2 is a real serial port with nothing connected to it. Com7 is a virtual comport with a StepOver USB-Pad on it.
In that case Portlist would be looking like that:
PortList[1]='2';
PortList[2]='7';
PortList[3] to PortList[256]='0';
If there is no available comport on your computer all items in Portlist will be '0'.

Appendix

StepOver PadSettings
the pad settings string used by getpadSettings and setPadSettings contains the settings of the pad. It's looking like that:
000000000000000001blueMPad / bluePad
The first 15 chars contains the pad ID in case of a +Pad or LE Pad. In case of a bluePad or blueMPad the are set to zero. The char 16-18 contains the number of the comPort. At the end of the string is the written name of the pad type. You may store that string in a secure way in your application to use that settings next time.

StepOver certificate
The certificate is needed by setPadSettings and getPadSettings, but only if you use a +Pad or LEPad, otherwise you can handle an empty string to the procedure.
- Why a certificate?
  The +Pads and LE-Pads transmitt their data in an encrypted way to the driver. The unique Pad ID is used to encrypt and decrypt the data. It would be very sensless to transmitt the biometrical data in an encrypted way to the driver if anyone can acces the data there in a decrypted way. Therefore you need to identify your application anytime you read or set the pad ID which is used for decryption.
- There can I get a certificate?
  You can get a certificate from StepOver GmbH. Mailto: Info@stepover.de or call 0700StepOver. Outside Germany call: +49 711 120269-30. Fax:+49 711 120269-31.
- What does a certificate cost?
  It's for free. We do this to secure our pad solution for you and your customer.
- What does StepOver need to get me a certificate?
  We need the name and adress of your company and the name of your application. You don't have to show us sourcecode or technical details.
- How does the certificate work?
  Anytime the user have to transmitt the hardware ID to your application (getPadSettings) a dialog window will be shown that tells the user the name of your application (that is contained in the certificat) to ensure that the user knows to which application he is transmitting its Pad ID. If you use getPadSettings or setPadSettings with a +Pad or LE Pad but without a certificate the user will get a warning window which tells him that a uncertified application tries to use his pad.

StepOver Biodata
The biodata string returned from getBiodataString or decompressBiodata is looking like that:
...x1744y2017z127f827l3t3896x1815y2016z127f828l3t3899x1887y20...
x and y are the coordinates of a cartesian coordinate system having its origin in the upper left corner. z is the pressure data (0-127). F=FrameNumber is the value from a counter in the pad and gets incremented with every frame. L=LineNumber is a counter which stays the same as long as the pen is on the pad and is incremented whenever a new line is started. At the end of the whole biodata string you will find the following:
...t3936p85-0-196-62-25g23:2:43 2.2.2
The chars after p represents the process ID. The process ID is not the same as the pad ID, but bounded to it so that no other StepOver pad may generate the same process ID. The process ID is unique and changes with each signature. You can, for example, use it as identification for a document in your workflow. The process ID is only available at the +Pad an LE Pad. The numbers after the letter g represents the GMT timestamp of the internal clock of the +Pad (Format: (H)H:(M)M:(S)S (d)d.(m)m.(y)y). If you use a pad who is not cabable of the features process ID and or timestamp you will get zeros after p an g.

Technical changes and errors excepted.
Last update 15.11.2003