Developer Roadmap

SigPlus

Table of Contents

Overview

This document is a guide to provide developers with a basic understanding of the necessary functions and commands for successful use and integration of the SigPlus ActiveX software tool. In addition, this guide will strive to help developers gain familiarity with the various software documentation, demos, and how-to guides already available on the Topaz website.

Enabling Signature Capture

Before starting, the Topaz tablet must be properly connected to the developer's computer.

SigPlus can be easily be instantiated at "Design Time" by choosing the SigPlus icon from the tool box, then drawing the SigPlus instance on the form in your IDE.

Anytime SigPlus is created dynamically, rather than at Design Time, it is important that you call the **InitSigPlus()** method. Additionally, setting the TabletInvisible property to True is important if SigPlus is not going to be visible on any form. For example (VB sample code):

```
Dim SigPlus1 As Object
Set SigPlus1 = CreateObject("SIGPLUS.SigPlusCtrl.1")
SigPlus1.InitSigPlus
SigPlus1.TabletInvisible = True
```

With the tablet properly physically connected, SigPlus needs to open the computer's serial or USB port in order to receive the signature information recorded from the pen. Setting the tablet state to 'on' (1) allows SigPlus to open the port and "listen" to the tablet for pen data. This is done through the TabletState property. For example:

```
SigPlus1.TabletState = 1
```

Once the computer's port is opened, as soon as the pen hits the tablet pen data will be transferred over the port and the signature will be visible in real time as the signing process takes place. As the signature is rendered it is also captured in SigPlus.

How Do I Tell When the Signature is Captured?

There is no sure-fire way programmatically for the developer to deduce whether the author has finished signing programmatically. One possible way might consist of polling and checking for the following:

1. First (within your polling event), check if the author has started signing by querying the NumberOfTabletPoints:

```
If SigPlus1.NumberOfTabletPoints = 0 Then
  'signature is not yet being captured
  Exit Sub
End If
```

   A signature is made up of points, captured and accumulated as the pen is drawn across the pad. Zero points means no signature currently exists. Polling to check if the number of points is > 0 will enable the developer to see when the user has started signing by showing when the tablet points are > 0.

2. Tablet points will start increasing with each polling event, indicating the user is still signing:

```
Static intCounter As Integer
If SigPlus1.NumberOfTabletPoints > 0 Then
  If SigPlus1.NumberOfTabletPoints > intCounter Then
          'signature is still being captured, keep polling
          intCounter = SigPlus1.NumberOfTabletPoints
  Else
          'points have stopped accumulating…check again or go on??
  End If
End If
```

When the tablet points stop increasing, it is reasonable to assume that the signature has been captured. The problem with relying solely on tablet points is that if the user drops the pen, or for some reason pauses during signing, etc., the tablet points will remain static until the pen is picked up and the signature is completed, which could take seconds or more. After the “Else” above, you must decide what you want to do as the developer, to comfortably feel that the user is done signing. That might be poll a few more times to verify that the points do not increase again, for example.

The most reliable means to know when signature capture is completed is to code within a sure-fire event. Some popular events used are as simple as an “OK” or “Done” button that the user or operator can click when signature is completed. Depending on the application and tablet model being developed for, these buttons can be programmed to be displayed through the computer the tablet is connected to, or if the tablet has an LCD screen, the buttons can be controlled by the signer on the LCD screen itself, by tapping with the pen.

For more information regarding programming for Topaz LCD equipped signature tablets:
www.topazsystems.com/Software/SigPlusLCD.pdf

For information on the easy to use Topaz Active-X Client Demo for programming LCD tablets:
www.topazsystems.com/Software/download/SigPlusLCD.pdf

Once the signature is captured the tablet state should be returned to 0, which will close the port.

Compression and Encryption Issues

The average size of a signature can range between 6-12 kb (6,000-12,000 characters). After the signature has been captured the developer can set the compression level to greater than 0, (with 0 being default - no compression). As a rule, the compression mode should be set at 0 during signature capture. The code below illustrates how the compression mode can be set. Each successive compression mode compresses the signature at a 2.5 to 1 ratio.

```
SigPlus1.SigCompressionMode = myCompVal
```

Encryption allows the developer to tie a file, or a string/set of strings, to a signature so that the signature can only be successfully returned if the contents of the file or the strings are returned unchanged. The process is rather like creating a key from the data (the file(s) or string data) which is the only way to unlock the signature. The end result is that the signature is tied to that file(s)/data, providing the signature a context.

NOTE: ENCRYPTION MODE MUST BE 0 DURING SIGNATURE CAPTURE.

Signature encryption allows the developer to tie the signature to a set of data, or a file or files. This provides the signature with a context: that the signature belongs with this data or file(s). Therefore, the signature cannot be associated with any other data/file(s) except to those with which it has been encrypted. This is a key component of creating a legally-binding, fully eSign-compliant digital handwritten signature.

Signature encryption is not necessary to capturing and storing the signature, however. If the developer is not interested in eSign law-compliant, legally binding signatures, they certainly do not need to utilize the following encryption methods.

Whether encryption is implemented or not, the signature (when stored in the Topaz SIG file format, or as the ASCII hex string SigString) inherently contains the biometric information useful to a forensic analyst later to determine the signer's identity. Such biometric data as pen stroke order and velocity information, for example, add up to useful data for determining the identity of the signer.

It is the encryption and the biometric data that, together, comprise a complete eSign law-compliant, legally binding digital handwritten signature.

How to Encrypt a Signature

To encrypt/decrypt the signature, use something like the code below. Encryption/Decryption makes use of the AutoKey function comprised of AutoKeyStart, AutoKeyData, and AutoKeyFinish, as well as the EncryptionMode property.

Note that the AutoKeyStart() method is ONLY called if the developer plans on using the AutoKeyData property to pass in strings. If the user passes in a path to a file that SigPlus must go and get, then the developer should NOT call the AutoKeyStart() method.

```
SigPlus1.KeyString = “0000000000000000” //zero out key
SigPlus1.EncryptionMode = 0 //zero out encryption
SigPlus1.TabletState = 0
SigPlus1.AutoKeyStart
SigPlus1.AutoKeyData = "some sample data" //this could be a path to a file, as well
SigPlus1.AutoKeyData = "some more sample data"
SigPlus1.AutoKeyData = "set as much as you like"
SigPlus1.AutoKeyFinish
SigPlus1.SigCompressionMode = myCompVal
SigPlus1.EncryptionMode = myEncVal
‘now you can either extract the signature, or return it…see below
```

The encryption and decryption of signatures are very similar, with the major differences being:

1. If it's a SIG file, whether you use ImportSigFile() or ExportSigFile() at the end of your encryption routine. Import is for bringing back a signature, Export is for getting the signature out, and saved to the location passed in.
2. If it's SigString, then it's which side of the "=" sign the SigString is, for example: SigPlus1.SigString = someSigStringValue : *this puts a signature into SigPlus* someSigStringValue = SigPlus1.SigString : *this copies the signature to the variable…store as you wish*

Essentially, the process for both encryption and decryption can be seen below.

Encrypt

1. Sign
2. AutoKey/Encrypt/Compress
3. Get signature out

Decrypt

1. AutoKey/Encrypt/Compress (same as when encrypted—data or file must match, of course)
2. Get signature out of database / Import from file
3. If the data is the same, and all the modes match, the signature is returned and displayed

To Extract signature from SigPlus after encryption/compression is complete, in either the Topaz .SIG file or as a convenient to use ASCII hex string, use the below calls:

```
'as File
SigPlus1.ExportSigFile "C:\mypath.sig"

'as string
myStrVariable = SigPlus1.SigString
```

To return a saved signature back to SigPlus, use the below code:

```
'as File
SigPlus1.ImportSigFile "C:\mypath.sig"

'as string
SigPlus1.SigString = mySavedSignature
```

LCD Tablet Interactive Functionality

This section of the roadmap deals exclusively with the development of interactive applications for use with the Topaz family of LCD equipped signature capture tablets.

For more information regarding these types of tablets, view Topaz signature pads at:
www.topazsystems.com/products.

For sales and pricing information, please contact Sales at:
sales@topazsystems.com.

View developer's demos at:
www.topazsystems.com/demossourcecode.html.

For further information regarding LCD development:
www.topazsystems.com/Software/download/SigPlusLCD.pdf.

Interactive applications are not a requirement for signature capture, but these customizable programs allow for a more seamless and user-friendly interface for many applications. The primary functionality for programming with the Topaz LCD signature capture tablets has to deal with the use of HotSpots, or areas of the LCD screen which accept pen taps from the user and then run the application as appropriate. Here is a picture of a simple menu displayed on a Topaz SignatureGem LCD 1x5 signature capture tablet. The “Cancel”, “Clear”, and “Done” button are all HotSpots.

There are numerous methods and properties that are used in the programming of an interactive application, view them all in the SigPlus Developer’s Manual at: **www.topazsystems.com/Software/sigplus.pdf**

Writing Signature Images

SigPlus allows developers to convert the native signature file into an image format, if the developer desires. Topaz supports .JPG, .TIF, .BMP, .WMF, and .EMF image file formats. One important factor to be aware of is that once the signature is converted into an image, it will no longer contain the biometric information, can no longer be encrypted, and cannot be tied to a document; thus, it is no longer legally-binding.

SigPlus offers many options to handle the converted signature image, including ways to alter the size, resolution, image zoom, ink thickness, color, etc.

Return a Signature Bitmap Byte Array

To return a bitmap byte array (an array containing the signature bmp), it must be used with ImageFileFormat=0. The benefit of a bitmap array is that you do not have to write the bitmap image to file first:

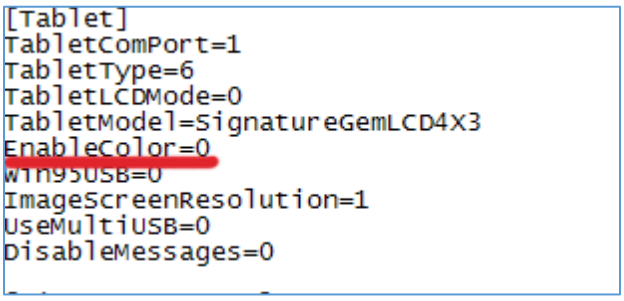
```
Dim Size As Long
Dim ByteValue() As Byte


SigPlus1.ImageFileFormat = 0
SigPlus1.ImageXSize = 1500
SigPlus1.ImageYSize = 500
SigPlus1.ImagePenWidth = 12
SigPlus1.JustifyMode = 5
SigPlus1.BitMapBufferWrite
Size = SigPlus1.BitMapBufferSize ReDim
ByteValue(Size)
ByteValue = SigPlus1.GetBitmapBufferBytes
SigPlus1.BitMapBufferClose
```

Color Images

To write images in color (this ONLY will function when writing out JPGs):

First, open the WIN\SigPlus.ini file. Find the parameter called “EnableColor=0” in the [Tablet] section.



Change this value to “EnableColor=1”. Save the file. This sets SigPlus to create color signatures.

Then, use the means available through your development environment to set the “BackColor” and “ForeColor” of your signature object.

For example, in VB, there is a ForeColor and BackColor property available. To change the colors, please choose a color from the following list, and replace the decimal value of the ForeColor and BackColor with the new color's decimal value.

Black=0
White=16777215
Light Blue=16776960
Dark Blue=16711680
Red=255
Green=65280

Yellow=65535
Light Gray=14671839
Dark Gray=8355711
Pink=12615935
Purple=16711808

For example, for dark blue ink on a yellow background, change the parameters in the SigToolImager.ini as follows:

ForeColor=16711680
BackColor=65535

To determine a specific color's decimal value, please first determine the hex value of your color (the RGB value). For example, perfect blue would be 0000FF (where there is 0 red, 0 green, and 255 blue). Next, flip the value backwards (so instead of 0000FF, it becomes FF0000.) Finally, open the “Calculator” program provided in “Start” → “Programs” → “Accessories”. Click on the “View” menu. Choose the “Scientific calculator”. Choose “Hex” in the radio button selection. Type in your 6-digit hex value, as determined in the above instructions (for our example, we type in 0000FF for perfect blue). Finally, choose “Dec” in the radio button selection, and the Hex value will be converted into the Decimal value necessary for the ForeColor and BackColor parameters in the SigToolImager.ini. As you will see, the Decimal value for perfect blue is 16711680.