Software Developer Guide

SigPlusNET

Version 2.1

Copyright © 2023 Topaz Systems Inc. All rights reserved.

*For Topaz Systems, Inc. trademarks and patents, visit **www.topazsystems.com/legal**.*

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Recent Version Changes

Current Version:

- 2.1.5
  - Updated to .NET Framework 4.7.1. Added sample source for .NET 6 applications.
  - Support added for disabling compression in ini and reading LCDCompressMode and LCDZCompression from Model section of ini.

Previous Versions:

- 2.0.1.0
  - Provides a new base Assembly version for forwards compatibility.
- 2.0.0.50
  - Adds AutoKeyAddANSIData function for compatibility with SigPlusActiveX.
- 2.0.0.46
  - Adds SigWeb interface for better communication with serial devices. Used explicitly in SigWeb.
- 2.0.0.42
  - Contains fixes for remote environments.

General Release Notes

I. Setting the COM Port

- Be sure that tablet state is off when selecting the COM port. The COM port must be selected first, and then tablet state turned on. Also, be sure to set tablet state off before exiting the application.

II. SigPlusNET Component Inheritance

- SigPlusNET inherits all functionality provided by the Control class (Systems.Windows.Forms.Control).

III. Important Notice

- These guidelines or any or all additional documentation or examples do not constitute a warranty about the performance, security, or legal acceptability of SigPlus software in any specific use or implementation. To the extent that SigPlus is used to achieve regulatory or other specific objectives within an industry, you must consult competent experts or regulatory officials together with your own plan to achieve your desired business objectives using the Topaz tools.

Topaz Namespace

SigPlusNet

Provides all the functionality required for customized capture and encryption of electronic handwritten biometric signatures.

SigPlusNET Class

Public class SigPlusNet

Remarks: Provides all the functionality required for customized capture and encryption of electronic handwritten biometric signatures.

Thread Safety: Public static (Shared in Visual Basic) members of this type are safe from multithreaded operations. Instance members are not guaranteed to be thread-safe.

Namespace: Topaz

Assembly: SigPlusNET (in SigPlusNET.dll)

Events for SigPlus NET Control

PenDown

This event will fire when the pen has made contact with the tablet.

For Instance:

```
Topaz.SigPlusNET sigPlusNETObj = new Topaz.SigPlusNET();
sigPlusNETObj.PenDown += new System.EventHandler(this.handleSigPlusNETPenDown);
…
void handleSigPlusNETPenDown(){
        //Pen down event triggered
}
```

PenUp

This event will fire when the pen has been lifted from the tablet.

For Instance:

```
Topaz.SigPlusNET sigPlusNETObj = new Topaz.SigPlusNET();
sigPlusNETObj.PenUp += new System.EventHandler(this.handleSigPlusNETPenUp);
…
void handleSigPlusNETPenUp(){
        //Pen up event triggered
}
```

Methods and Properties for Use with LCD Tablets

I. General Methods:

ClearSigWindow

```
public void ClearSigWindow(
  short Inside
);
```

Remarks: Erases data either inside or outside of sig window based on value of short inside. See SetSigWindow.

Parameters: If Inside equals 0, then signature data is erased (in window). Otherwise, if Inside equals 1, then data outside sig window is erased.

LCDClear

```
public void LCDClear();
```

Remarks: Erases the LCD display easily by calling LCDRefresh to do it.

LCDSetTabletMap

```
public void LCDSetTabletMap(
  int LCDType,
  int LCDXSize,
  int LCDYSize,
  int LCDXStart,
  int LCDYStart,
  int LCDXStop,
  int LCDYStop
);
```

Remarks: This method allows overriding the LCD properties typically defined by the SigPlus.ini file on load of the SigPlus object.

Parameters:
LCDType - Specifies LCD type and format. Currently, unimplemented.
LCDXSize - X Size of LCD display, in pixels
LCDYSize - Y Size of LCD display, in pixels
LCDXStart - X Pos in logical tablet coordinates of LCD
LCDYStart - Y Pos in logical tablet coordinates of LCD
LCDXStart - X Pos in logical tablet coordinates of LCD
LCDYStop - Y Pos in logical tablet coordinates of LCD

SetSigWindow

```
public void SetSigWindow(
  short Coords,
  short NewXPos,
  short NewYPos,
  short NewXSize,
  short NewYSize
);
```

Remarks: This function sets a window in the logical tablet space that restricts the operation of some functions to the specified window. The functions behave as follows: JustifyMode will only operate on points inside of this window. ExportSigFile and WriteImageFile will only operate on points inside the window. SigString only operates on points inside of the window. ClearTablet will only clear in the window. This behavior is enabled by setting the start and stop values to non-zero. The window defaults to (0,0,0,0). The window can be enabled at one spot, re-enabled at another, etc., without disabling in between, and then disabled when the various parts of the tablet data have been separated and stored. To determine the logical values in the control for the installed tablet, see the TabletLogicalXSize and TabletLogicalYSize properties.

Parameters:
Coords - Coordinate system used for this hot spot, 0 = Logical tablet coordinates, 1 = LCD Coordinates.
NewXPos - Location in logical tablet coordinates (upper left - 0,0).
NewYPos - Same
NewXSize - XSize in logical tablet pixels
NewYSize - YSize in logical tablet pixels

II. Graphics Methods:

LCDRefresh

```
public bool LCDRefresh(
  int Mode,
  int XPos,
  int YPos,
  int XSize,
  int YSize
);
```

Remarks: Sends tablet a refresh command with 4 possible modes: 0 - Clear, display is cleared at the specified location. 1 - Complement, complements display at the specified location. 2 - WriteOpaque, transfers contents of the background memory to the LCD display, overwriting the content of the LCD display. 3 - WriteTransparent, transfers contents of the background memory in the tablet to the LCD display and combined in the contents of the LCD display.

Parameters:
Mode-Defined as above (0-3)
XPos-Location in LCD Coordinates (upper left-0,0)
YPos-Same
XSize-X size in LCD pixels
YSize-Y size in LCD pixels

Return Value: True if checksum received and verified, False if no or incorrect checksum received from tablet.

LCDSendGraphic

```
public bool LCDSendGraphic(
  int Dest,
  int Mode,
  int XPos,
  int YPos,
  Bitmap BitmapData
);
```

Remarks: This writes an image to the LCD by taking a .NET Drawing::Bitmap as the source for the image.

Parameters:
Dest- 0=Foreground,1=Background memory in tablet
Mode-0-3 as defined in LCDWriteString
XPos-Location in LCD coordinates (upper left- 0,0)
YPos-Same
BitmapData-Source for rendered image

Return Value: True if successful, False if not.

LCDSetPixelDepth

```
public bool LCDSetPixelDepth(
  short PixDepth
);
```

Remarks:. Color image use only with T-LBK57GC and T-LBK43LC devices. Used to specify color or black and white images when passing an image to paint to the LCD using the LCDSendGraphic() function. Call LCDSetPixelDepth() appropriately prior to painting to the LCD.

Parameters:
PixDepth – 0 = Black and White, 8 = Color

LCDSetWindow

public bool LCDSetWindow(
  int *XPos*,
  int *YPos*,
  int *XSize*,
  int *YSize*
);

Remarks: Sets a signature window that restricts the ink of the SigPlus object to said window on the LCD itself (see SetLCDCaptureMode).

Parameters:
XPos – Location in LCD coordinates (upper left – 0,0)
YPos – Same
XSize – X size in LCD pixels
YSize – Y size in LCD pixels

Return Value: True if checksum received and verified, False if no or incorrect checksum received from tablet.

LCDStringHeight

public int LCDStringHeight(
  Font *DrawFont*,
  string *Str*
);

Remarks: Takes a string and a .NET font and hand back how tall the string is in pixels.

Parameters:
DrawFont - .NET font
Str – String

Return Value: Height of string in pixels

LCDStringWidth

public int LCDStringWidth(
  Font *DrawFont*,
  string *Str*
);

Remarks: Takes a string and a .NET font and hand back how wide the string is in pixels.

Parameters:
DrawFont - .NET font
Str – String

Return Value: Width of sting in pixels

LCDWriteString

public bool LCDWriteString(
  int *Dest*,
  int *Mode*,
  int *XPos*,
  int *YPos*,

```
  Font DrawFont,
  string Str
);
```

Remarks: Used to write the image data to the LCD Display. The data is written at the location specified by the combination of Dest, XPos, and YPos. The Mode determines how the data is written.

Mode 0 - Clear: The Display is cleared at the specified location.
Mode 1 - Complement: The Display is complemented at the specified location.
Mode 2 - WriteOpaque: The contents of the background memory in the tablet are transferred to the LCD display, overwriting the contents of the LCD display.
Mode 3 - WriteTransparent: The contents of the background memory in the tablet are combined with and transferred to the visible LCD memory

Parameters:
Dest-0 = Foreground, 1 = Background memory in tablet
Mode-0, 1, 2, 3 as defined above
XPos-Location in LCD coords to draw at
YPos-Same
DrawFont-Not currently implemented, pass a 0
Str-ASCII hex string value.

Return Value: True if checksum received and verified, False if no or incorrect checksum.

III. Graphics Properties:

GetLCDCaptureMode

```
public int GetLCDCaptureMode();
```

Remarks: Gets the current LCD Capture Mode for the tablet.

Return Value: Mode the LCD is set to capture signatures in: 0 - no LCD commands are sent to the tablet, 1 - sets capture mode to be active with Autoerase in the tablet, 2 - sets the tablet to persistent ink capture without autoerase, 3 - signature ink is displayed inverted on a suitable dark background set using the Graphic functions.

LCDGetCompressionMode

```
public bool LCDGetCompression();
```

Return Value: Returns true if compression mode being used for encoding data sent to the signature pad.

LCDGetLCDSize

```
public uint LCDGetLCDSize();
```

Return Value: Returns the Xsize and YSize in LCD coords of the tablet currently specified in the SigPlus.ini. The value returned is an Unsigned integer (YSize – upper 16 bits, XSize – lower 16 bits)

LCDGetZCompressionMode

```
public bool LCDGetZCompression();
```

Return Value: Returns true if Z compression mode is being used for encoding data sent to the signature pad.

LCDSetCompressionMode

```
public bool LCDSetCompressionMode(
  bool newMode
);
```

Remarks:. True to enable Compression mode to compress the data being sent to the SE, T-LBK57GC, and T-LBK43LC pads.

LCDSetZCompressionMode

```
public bool LCDSetZCompressionMode(
  bool newMode
);
```

Remarks:. True enables Z Compression mode to compress the data being sent to the T-LBK57GC and T-LBK43LC pads. The compression mode must be set as well. This can be achieved using LCDSetCompressionMode.

SetLCDCaptureMode

```
public void SetLCDCaptureMode(
  int CaptureMode
);
```

Remarks: Sets the current LCD Capture Mode for the tablet.

Parameters:
CaptureMode-Mode the LCD is set to capture signatures in,
Mode 0=no LCD commands are sent to the tablet
Mode 1=sets capture mode to be active with Autoerase in the tablet
Mode 2=sets the tablet to persistent ink capture without autoerase
Mode 3=signature ink is displayed inverted on a suitable dark background set using the Graphic functions.

IV. Keypad Methods:

KeyPadAddHotSpot

```
public void KeyPadAddHotSpot(
  short KeyCode,
  short CoordToUse,
  short XPos,
  short YPos,
  short XSize,
  short YSize
);
```

Remarks: Defines in software the location of a tablet HotSpot which is used by the developer to detect user pen taps. The KeyPadAddHotspot() method will require slight variations in px coord location (arguments 3 and 4, varying for about 1 px to 5 px) from its counterpart LCDWriteBitmap(), LCDWriteFile(), or LCDWriteString() method call. For best results, Topaz recommends the following in terms of adding hot spots:

1. Make the hotspot larger than the image/text representing it…this eliminates “hunting and tapping” on the part of the user.
2. Making all hotspots no smaller than 10 px in both the Y and X axis

Parameters:
KeyCode-Integer value defining the HotSpot.
CoordToUse-Coordinate system used for this HotSpot. 0 for logical tablet coordinates, 1 is logical tablet coordinates.
XPos-Location upper left corner of hot spot (upper left- 0,0)
YPos- Same
XSize-X size in pixels.
YSize-Y size in pixels.

KeyPadClearHotSpotList

```
public void KeyPadClearHotSpotList();
```

Remarks: Clears the controls internal list of HotSpots created using KeyPadAddHotSpot.

KeyPadQueryHotSpot

public short KeyPadQueryHotSpot(
  short *KeyCode*
);

Remarks: Queries whether the specified HotSpot has been tapped by the user. Returns a true if the control contains data that is within the definition of the keyCode on the tablet.

Parameters: KeyCode-Mapped Logical Tablet Coordinates.

Return Value: Number of points within the KeyCode definition.

SigPlus NET Control Methods

AutoKeyFinish

public bool AutoKeyFinish ();

Remarks: Completes the auto key generation function.

Return Value: True if Key set in SetAutoKeyANSIData contains only ASCII characters, false if non-ASCII characters are in the Key.

AutoKeyStart

public void AutoKeyStart () ;

Remarks: Initializes the automatic key generation function which derives a key from the data fed to it via SetAutoKeyANSIData (string), when all data is input then AutoKeyFinish () must be called to complete key generation. If the AutoKeyStart method is not called, then the SetAutoKeyANSIData method is used to pass the path to the file, which is then used to encrypt the signature. When AutoKeyStart is called then SetAutoKeyANSIData is used to pass in string literals as data.

ClearTablet

public void ClearTablet();

Remarks: Clears the signature object of ink.

ExportSigFile

public bool ExportSigFile(
  string *FileName*
);

Remarks: Writes out a signature file in the Topaz image-free raw tablet data vector file format (sig extension).

Parameters: FileName – Name of file

Return Value: True if successful, false if not successful

GetKey

public byte[] GetKey();

Return Value: Returns the current binding key, as a byte array.

GetKeyReceipt

public int GetKeyReceipt();

Remarks: Returns a 32 bit value that is uniquely derived from the key, it can be used to verify that a document has not been modified if the Auto key feature was used to generate the key.

Return Value: 32 bit binary receipt.

GetKeyReceiptAscii

public string GetKeyReceiptAscii();

Remarks: Returns the key receipt as an 8 character Ascii hex string.

Return Value: The Ascii string.

GetNumberOfStrokes

public int GetNumberOfStrokes();

Remarks: Returns the total number of strokes in the current signature. Can be used to detect if a signature is present, or not. A signature may have as few as one stroke. Since the Topaz tablet is collecting raw tablet data, it cannot determine and does not assume that the data is a signature. The term "stroke" is used to describe the number of segments in the raw data.

Return Value: Integer value of number of points in the stroke.

GetNumPointsForStroke

public int GetNumPointsForStroke(
  int *StrokeNumber*
);

Remarks: Returns the total number of points in the specified stroke.

Parameters: StrokeNumber – the number of the stroke to inquire about. Ranges from 0 to the NumberOfStrokes minus one.

Return Value: integer value of number of points in the stroke

GetPointXValue

public int GetPointXValue(
  int *StrokeIdx,*
  int PointIdx
);
Remarks: Returns the X coordinate value for the specified point. The value is in LogicalTablet Coordinates.

Parameters:
StrokeIdx – the index of the stroke for the point desired. Ranges from 0 to the NumberOfStrokes minus one.
PointIdx – the index of the point in the stroke. Ranges from 0 to the NumberOfPoints minus one.

Return Value: Integer value of the x coordinate for the point

GetPointYValue

public int GetPointYValue(
  int *StrokeIdx,*
  int PointIdx
);
Remarks: Returns the Y coordinate value for the specified point. The value is in LogicalTablet Coordinates.

Parameters:
StrokeIdx – the index of the stroke for the point desired. Ranges from 0 to the NumberOfStrokes minus one.
PointIdx – the index of the point in the stroke. Ranges from 0 to the NumberOfPoints minus one.

Return Value: Integer value of the y coordinate for the point

GetPointPValue

public int GetPointPValue(
  int *StrokeIdx,*
  int PointIdx
);
Remarks: Returns the pressure value for the specified point. Pressure must be supported by the Topaz device.

Parameters:
StrokeIdx – the index of the stroke for the point desired. Ranges from 0 to the NumberOfStrokes minus one.
PointIdx – the index of the point in the stroke. Ranges from 0 to the NumberOfPoints minus one.

Return Value: Integer value of the pressure for the point

GetPointTValue

public int GetPointTValue(
  int *StrokeIdx,*
  int PointIdx
);
Remarks: Returns the time value for the specified point.

Parameters:
StrokeIdx – the index of the stroke for the point desired. Ranges from 0 to the NumberOfStrokes minus one.
PointIdx – the index of the point in the stroke. Ranges from 0 to the NumberOfPoints minus one.

Return Value: Integer value of the time for the point

GetSaveSigInfo

public bool GetSaveSigInfo();

Return Value: True if SigInfo is enabled, False if disabled.

GetSigImage

public Image GetSigImage();

Remarks: Renders and returns a .NET Image, using Image X and Y Sizes and Image pen width. The returned image can use the SaveAs method in the Image class to save the image into a number of supported formats, such as jpg, tif, bmp, png, etc.

Return Value: A .NET Image.

GetSigReceipt

public int GetSigReceipt ();

Remarks: Returns a 32 bit receipt similar to the key receipt. Forms receipt by using the auto key generation algorithm on the signature file and the result can be used to verify that the signature has not been modified.

Return Value: 32 bit binary receipt.

GetSigReceiptAscii

public string GetSigReceiptAscii ();

Remarks: Same as GetKeyReceiptAscii, but for Sig receipt.

ImportSigFile

public bool ImportSigFile(
  string *FileName*
);

Remarks: Clears the current signature, read in a signature file in the Topaz vector file format, and display it.

Parameters: FileName – Contains the path and filename that is to be read from.

Return Value: True if successful, False if not successful.

NumberOfTabletPoints

public int NumberOfTabletPoints();

Remarks: Returns the total number of points in the current signature, can be used to detect if a signature is present or not.

Return Value: Integer value of number of points in the signature.

SetKey

```
public void SetKey(
  byte[] KeyString
);
```

Remarks: Sets the binding key for storing the signature data

Parameters: KeyString – byte array containing the raw key data. The number of key data bytes used depends on the algorithm used.

SetKeyString

```
public void SetKeyString(
  string KeyString
);
```

Remarks: Sets the Key String into the SigPlus component.

Parameters: KeyString - Hash of the data used to encrypt/decrypt the signature, key internally generated by SigPlus.

SetUseAmbientColors

```
public void SetUseAmbientColors(
  bool UseAmbientColors
);
```

Remarks: Must be enabled to allow the ForeColor and BackColor of the object to be modified.

SigPlusNET Constructor

Initializes a new instance of the SigPlusNET class.

```
public SigPlusNET() ;
```

Sleep

```
public void Sleep(
  uint TimeInMs
);
```

TabletConnectQuery()

```
public bool TabletConnectQuery()
```

Note: TabletConnectQuery() can only be used in a local environment. It cannot be used in a Terminal Server or Citrix environment[MA1].

Return Value: Boolean indicating if signature pad is connected. Uses TabletType value to determine what signature pad connection type to use.

TabletModelNumber()

```
public int TabletModelNumber();
```

Remarks: Please note: SetTabletState(1) must be successfully set before TabletModelNumber() can return a value. TabletModelNumber() returns a value corresponding to a particular Topaz tablet model, used to detect whether the signature pad in question is connected.

Parameters: None.

Return: Model Number. The following list shows the return from TabletModelNumber() and the corresponding Topaz tablet model.

1 = TL(BK)766
8 = TL(BK)755 or TL(BK)750
11 or 12 = TL(BK)462
15 = TL(BK)460
43 = TLBK43LC
57 = TLBK57GC
58 = All Topaz "SE" signature pad models

To differentiate "SE" pads from one another (see 58 above), use TabletSerialNumber(), and the following values correspond to these "SE" pads:

550 = TLBK766SE
551 = TLBK462SE
553 or 557 = TLBK755SE or TLBK750SE

See TabletSerialNumber() below for further details.

TabletSerialNumber()

public long TabletSerialNumber();

Remarks: Please note: SetTabletState(1) must be successfully set before TabletSerialNumber() can return a value. Given the use of these Topaz tablet models:

T-L(BK)462
T-LBK57GC
T-LBK43LC

TabletSerialNumber() returns a unique value corresponding to a particular Topaz tablet. This value can be used to differentiate one signature tablet device from another of the same tablet model type (given the 3 models listed above).

Additionally, given a Topaz 'SE' tablet model (one of the following):

TLBK462SE
TLBK766SE
TLBK755SE
TLBK750SE

TabletSerialNumber() is used as a secondary model number identifier for determining which specific 'SE' tablet is connected. Given the TabletModelNumber() function, a return of 58 indicates that an 'SE' signature pad is connected. At this point, TabletSerialNumber() can now be used to further identify which specific 'SE' tablet is connected. The following TabletSerialNumber() returns correspond to these 'SE' tablet models:

550 = TLBK766SE
551 = TLBK462SE
553 or 557 = TLBK755SE or TLBK750SE

No other Topaz tablet models are able to return a TabletSerialNumber() value for particular use.

Parameters: None.

Return: Serial or further refined Model Number depending upon usage.

WriteImageFile

public bool WriteImageFile(
  string *FileName*
);

Remarks: Scheduled for removal. WriteImageFile() should not be used. Instead, please refer to the GetSigImage() function which returns the signature as a System.Drawing.Image

Return Value: N/A

Parameters: Filename – N/A

SigPlus NET Control Properties

I. General Properties

SetAnnotate

```
public void SetAnnotate(
  string Annotate
);
```

Remarks: Sets Annotation string stored with signature.

Parameters: Annotate – ASCII line character.

GetAnnotate

```
public string GetAnnotate();
```

Return Value: Returns current ASCII Annotation string

SetAutoKeyANSIData[MA2]

```
public void SetAutoKeyANSIData(
  string Key
);
```

Remarks: Adds data to the AutoKey generation function, distilling it down to a set-length key which will ultimately encrypt the signature to the data. Used with AutoKeyStart and AutoKeyFinish methods, but called as a property. SetAutoKeyData and AutoKeyAddData are deprecated and should not be used for new software.

Parameters: Key **-** String containing the data (directly represents document being signed). Should be called only once, so concentrate all of the data into a single variable. Should also be used with SetEncryptionMode and GetSigString or SetSigString.

SetSigCompressionMode

```
public void SetSigCompressionMode(
  int CompressionMode
);
```

Remarks: Sets the current compression mode for signatures.

Parameters: CompressionMode-Mode for compression of signature, where 0 - no compression, 1 - lossless compression with compacted data format, 2-8 - compression ratio of signature stored in in .sig file where 2 - 1KB typ, 4 - 500 byte typ, and 8 - 250 byte typ. Topaz Systems does not recommend compressing beyond setting 1 unless size is more important than signature quality.

GetSigCompressionMode

```
public int GetSigCompressionMode();
```

Remarks: Returns compression mode for signatures.

Return Value: Mode for compression of signature: 0 - no compression, 1 - lossless compression with compacted data format, 2-8 - compression ratio of signature stored in in .sig file where 2=1KB typ, 4 - 500 byte type, and 8=250 byte type. Topaz Systems does not recommend compressing beyond setting 1 unless size is more important than signature quality.

SetEncryptionMode

```
public void SetEncryptionMode(
  int EncryptionMode
);
```

Remarks: Sets EncryptionMode.

Parameters: EncryptionMode-0= no encryption, 1= medium encryption, 2=higher security encryption mode.

GetEncryptionMode

```
public int GetEncryptionMode();
```

Remarks: Returns current EncryptionMode.

Return Value: Numeric value of encryption mode, 0 = no encryption, 1 = medium encryption, 2 = higher security encryption mode.

SetJustifyMode

```
public void SetJustifyMode(
  int JustifyMode
);
```

Remarks: Sets the current justification mode- how the signature is sized and positioned in the signature box as well as when using the WriteImageFile method.

Parameters: JustifyMode - Justification mode, 0 - normal no justification, 1 - justify and zoom signature (upper left corner) 2 - justify and zoom signature (upper right corner) 3 - justify and zoom signature (lower left corner) 4 - justify and zoom signature (lower right corner) 5 - justify and zoom signature (center of control).

GetJustifyMode

```
public int GetJustifyMode();
```

Remarks: Gets the current justification mode – how the signature is sized and positioned in the signature box as well as when using the WriteImageFile method.

Return Value: Justification mode: 0-normal no justification, 1 - justify and zoom signature (upper left corner) 2 - justify and zoom signature (upper right corner), 3 - justify and zoom signature (lower left corner), 4 - justify and zoom signature (lower right corner), 5 - justify and zoom signature (center of control).

SetJustifyX

```
public void SetJustifyX(
  int JustifyX
);
```

Remarks: Sets the buffer size in Logical Tablet Coordinates of "dead space" of left and right edge of SigPlus object if JustifyMode is 1-5. This method functions for both the signature box as well as when using the WriteImageFile method.

Parameters: JustifyX-Justification X buffer size in pixels to be set.

GetJustifyX

```
public int GetJustifyX();
```

Return Value: Justification X buffer size in pixels for display.

SetJustifyY

```
public void SetJustifyY(
  int JustifyY
);
```

Remarks: Sets the buffer size in Logical Tablet Coordinates of "dead space" of top and bottom edge of SigPlus object if JustifyMode is 1-5. This method functions for both the signature box as well as when using the WriteImageFile method.

Parameters: JustifyY-Justification Y buffer size in pixels to be set.

GetJustifyY

```
public int GetJustifyY();
```

Return Value: Justification Y buffer size in pixels for display.

GetKeyString

```
public string GetKeyString();
```

Remarks: Provides hash of the encryption data in ASCII compatible format.

Return Value: Hash of encryption data.

SetSaveSigInfo

```
public void SetSaveSigInfo(
  bool SaveSigInfo
);
```

Remarks: Enables/disables the saving of TimeStamp and Annotate data in the signature.

Parameters: SaveSigInfo – If True then SigInfo will be saved (default), if False then the info will not be saved.

SetSigString

```
public void SetSigString(
  string SigString
);
```

Remarks: Puts signature into the component.

Parameters: SigString – Signature in ASCII format

Return Value: The Ascii string.

GetSigString

```
public string GetSigString();
```

Return Value: SigString as ASCII hex string.

SetTimeStamp

```
public void SetTimeStamp(
  string TimeStamp
);
```

Remarks: Sets the TimeStamp string for the signature.

Parameters: TimeStamp – terminated by ASCII new line character.

GetTimeStamp

public string GetTimeStamp();

Remarks: Gets the current Time Stamp string for the signature.

Return Value: ASCII new line character.

II. Tablet Properties

SetTabletBaudRate

```
public void SetTabletBaudRate(
  int BaudRate
);
```

Remarks: Sets TabletBaudRate, an internal property associated with tablet model.

Parameters: BaudRate – internal tablet property.

GetTabletBaudRate

public int GetTabletBaudRate();

Return Value: Current TabletBaudRate.

SetTabletComPort

```
public void SetTabletComPort(
  int Port
)
```

Remarks: Sets the COM port to use to communicate with a Topaz serial device. The SigPlus.NET component does not lock up a port as is the case with mouse-type drivers. Only set COM port when tablet state is OFF.

Parameters: 1 for on, and 0 (default) for off.

GetTabletComPort

public int GetTabletComPort ();

Return Value: Current COM port setting.

GetTabletComTest

public int GetTabletComTest ();

Remarks: Gets current hardware check mode.

Return Value: Current hardware check mode, 1 True if active, 0 False if not active.

SetTabletComTest

```
public void SetTabletComTest(
  int ComTest
```

Remarks: Sets hardware connection check mode. See code below for example:

```
sigPlusNET1.setTabletComTest(1);
sigPlusNET1.setTabletState(1);

if(sigPlusNET1.getTabletState() == 1)
{
        sigPlusNET1.SetTabletComTest(0); //turn off test mode
}
```

Parameters: 1 for on, and 0 (default) for off.

SetTabletFilterPoints

public void SetTabletFilterPoints()
int *Points*

Remarks: Sets the TabletFilterPoints, an internal property associated with tablet model.

Parameters: Points – Internal tablet property.

GetTabletFilterPoints

public int GetTabletFilterPoints ();

Return Value: Current TabletFilterPoints

SetTabletLogicalXSize

public void SetTabletLogicalXSize()
int *XSize*

Remarks: Sets the range of horizontal values to be used in representing signatures. This is the X-range used for the Topaz vector format, and the internally used format.

Parameters: XSize – Integer value of Tablet logical size. Default is 2150.

GetTabletLogicalXSize

public int GetTabletLogicalXSize();

Return Value: Current horizontal values used in representing signatures in Logical Tablet Coordinates.

SetTabletLogicalYSize

public void SetTabletLogicalYSize()
int Y*Size*

Remarks: Sets the range of vertical values to be used in representing signatures. This is the Y-range used for the Topaz vector format, and the internally used format.

Parameters: YSize – Integer value of Tablet logical size. Default is 1400.

GetTabletLogicalYSize

public int GetTabletLogicalYSize();

Return Value: Current vertical values used in representing signatures in Logical Tablet Coordinates.

SetTabletResolution

public void SetTabletResolution()
int *Resolution*

Remarks: Sets TabletResolution, an internal property associated with tablet model and set by the TabletModel property, based on hardware tablet resolution is 410 dpi, (excluding ClipGem which is 275 dpi) but can be changed at the risk of affecting signature capture.

Parameters: Resolution – internal tablet property.

GetTabletResolution

public int GetTabletResolution();

Return Value: Current TabletResolution.

SetTabletRotation

public void SetTabletRotation()
int *Rotation*

Remarks: Sets the orientation on a 360 degree axis for display of tablet data. The data in the sig representation is stored in the native tablet orientation.

Parameters: Rotation – Display orientation, allowed values are 0, 90, 180, 270.

GetTabletRotation

public int GetTabletRotation();

Remarks: Gets the current orientation on a 360 degree axis for display of tablet data. The data in the sig representation is stored in the native tablet orientation.

Return Value: Current tablet orientation.

SetTabletState

public void SetTabletState(
  int *State*
);

Remarks: Enables tablet to access the COM or USB port to capture signatures or not.

Parameters: State-setting to 1 enables the tablet to capture signatures as above, setting to 0 disables signature capture.

GetTabletState

public int GetTabletState();

Remarks: Indicates capture state of the tablet.

Return Value: Value of 1 enables the component to access the selected COM or USB port and access the tablet for signature capture, 0 disables the tablet for capture.

SetTabletTimingAdvance

public void SetTabletTimingAdvance(
  int *Advance*
);

Remarks: Sets the TabletTimingAdvance, an internal property associated with tablet model.

Parameters: Advance – internal tablet property.

GetTabletTimingAdvance

public int GetTabletTimingAdvance();

Return Value: Current TabletTimingAdvance

SetTabletType

public void SetTabletType(
  int *TabletType*
);

Remarks: Determines if the tablet will accept data from a com port or USB driver.

Parameters:
TabletType-Default is 6
0=Normal mode. When tablet is activated it will accept input from the selected com port.
6=HSB tablet (USB mode for tablets using HID driver)

GetTabletType

public int GetTabletType();

Remarks: Gets TabletType value.

Return Value: Integer value of TabletType. See SetTabletType.

SetTabletXStart

```
public void SetTabletXStart(
  int XStart
);
```

Remarks: Sets the X position in Logical Tablet Coordinates of the upper left-hand corner of the component's signature box.

Parameters: XStart – X coordinate of the upper left corner of the signature box.

GetTabletXStart

```
public int GetTabletXStart () ;
```

Return Value: Current X position in Logical Tablet Coordinates of the upper left-hand corner of the component's signature box.

SetTabletXStop

```
public void SetTabletXStop(
  int XStop
);
```

Remarks: Sets the X position in Logical Tablet Coordinates of the lower right-hand corner of the component's signature box.

Parameters: XStop – X coordinate of the lower right corner of the signature box.

GetTabletXStop

```
public int GetTabletXStop();
```

Return Value: Current X position in Logical Tablet Coordinates of the lower right-hand corner of the component's signature box.

SetTabletYStart

```
public void SetTabletYStart(
  int YStart
);
```

Remarks: Sets the Y position in Logical Tablet Coordinates of the upper left-hand corner of the component's signature box.

Parameters: YStart – Y coordinate for the upper left corner of the signature box.

GetTabletYStart

```
public int GetTabletYStart();
```

Return Value: Current Y position in Logical Tablet Coordinates of the upper left-hand corner of the component's signature box.

SetTabletYStop

```
public void SetTabletYStop(
  int YStop
);
```

Remarks: Sets the Y position in Logical Tablet Coordinates of the lower right-hand corner of the component's signature box.

Parameters: YStop – Y coordinate for the lower right corner of the signature box.

GetTabletYStop

public int GetTabletYStop();

Return Value: Current Y position in Logical Tablet Coordinates of the lower right-hand corner of the component's signature box.

III. Display Properties

SetDisplayAnnotate

public void SetDisplayAnnotate(
  bool *DisplayAnnotate*
);

Remarks: Sets the component to display the Annotation string.

Parameters: DisplayAnnotate – Component to display Annotation string.

GetDisplayAnnotate

public bool GetDisplayAnnotate();

Remarks: Gets the current setting to display Annotation

Return Value: True if Annotation is displayed, false if not displayed

SetDisplayAnnotateData

public void SetDisplayAnnotateData(
  int *XPos*,
  int *YPos*,
  int *Size*
);

SetDisplayAnnotatePosX

public void SetDisplayAnnotatePosX(
  int *XPos*
);

Remarks: Sets the X position in pixels of the start of the Annotation String in the signature box.

Parameters: XPOS – X position for the start of the Annotation String to be set.

GetDisplayAnnotatePosX

public int GetDisplayAnnotatePosX();

Remarks: Gets the current X position for the start of the Annotation String in the signature box.

Return Value: X position for start of the Annotation String, if 0 then text is positioned 5% in from right side of signature box.

SetDisplayAnnotatePosY

public void SetDisplayAnnotatePosY(
  int *YPos*
);

Remarks: Sets the Y position in pixels of the start of the Annotation String in the signature box.

Parameters: YPOS – Y position for the start of the Annotation String to be set.

GetDisplayAnnotatePosY

public int GetDisplayAnnotatePosY();

Remarks: Gets the current Y position for the start of the Annotation String in the signature box.

Return Value: Y position for start of the Annotation String, if 0 then text is positioned 5% in from bottom edge of signature box.

SetDisplayAnnotateSize

public void SetDisplayAnnotateSize(
  int *Size*
);

Remarks: Sets the Y size of the Annotation start of the Time Stamp in the signature box.

Parameters: DisplayAnnotationSize – Y Size of Annotation text in pixels

GetDisplayAnnotateSize

public int GetDisplayAnnotateSize();

Remarks: Gets current Y size in pixels of the Annotation start in the signature box.

Return Value: Text size of the Annotation in pixels, if 0 then the text size is 7.5% of the Y size of the signature box.

SetDisplayPenWidth

public void SetDisplayPenWidth(
  int *PenWidth*
);

Remarks: Sets pen ink width for the displayed signature in pixels.

Parameters: PenWidth – Pen width for the displayed signature in pixels.

GetDisplayPenWidth

public int GetDisplayPenWidth();

Return Value: Current pen ink width for the displayed signature in pixels.

SetDisplayRotate

public void SetDisplayRotate(
  bool *Mode*
);

Remarks: Sets mode allowing signature rotation in the control after capture for Display only, does not save the .sig info rotated.

Parameters: DisplayRotation - Orientation for display of signature after capture.

GetDisplayRotate

public bool GetDisplayRotate();

Return Value: Value of rotation on a 360 degree axis for signature display.

SetDisplayRotateSave

public void SetDisplayRotateSave(
  bool *__unnamed000*
);

Remarks: Sets mode allowing signature rotation and the save of signature in rotated format after capture. Does not save the .sig file rotated. Display rotation only. The rotation value is set by TabletRotation. Note: This is the preferred way of setting TabletMode = add 768 Can be used to rotate and then save in rotated

format, signatures after capture, if the signature was accidentally taken in a rotated orientation during signature capture. Normally, to change the tablet orientation during capture, only the TabletRotation property is used.

Parameters: unnamed000-TRUE DisplayRotateSave mode = active, FALSE DisplayRotateSave mode = inactive

GetDisplayRotateSave

public bool GetDisplayRotateSave();

Remarks: Gets state of DisplayRotateSave. Can be used to rotate and then save in rotated format, signatures after capture, if the signature was accidentally taken in a rotated orientation during signature capture. Normally, so change the tablet orientation during capture, only the TabletRotation property is used.

Return Value: TRUE DisplayRotateSave mode = active, FALSE DisplayRotateSave mode = inactive.

SetDisplayTimeStamp

public void SetDisplayTimeStamp(
  bool *DisplayTimeStamp*
);

Remarks: Sets the component to display the developer provided Time Stamp string.

Parameters: DisplayTimeStamp - Component to be set to display Time Stamp.

GetDisplayTimeStamp

public bool GetDisplayTimeStamp();

Remarks: Gets the current setting for component to display Time Stamp.

SetDisplayTimeStampData

public void SetDisplayTimeStampData(
  int *XPos*,
  int *YPos*,
  int *Size*
);

Remarks: The default is the lower left corner, at -8% of the screen height.

Parameters:
XPos - X position to be set for start of display of Time Stamp.
YPos - Y position to be set for start of display of Time Stamp.
Size - Size of Time Stamp to set.

SetDisplayTimeStampPosX

public void SetDisplayTimeStampPosX(
  int *XPos*
);

Remarks: Sets the X position in pixels of the start of the Time Stamp in the signature box.

Parameters: XPos-X position to be set for start of display of Time Stamp.

GetDisplayTimeStampPosX

public int GetDisplayTimeStampPosX();

Remarks: Gets the current X Position for the start of the Time Stamp String in the signature box.

Return Value: X value in pixels relative to the left edge, if 0 then 5% in from left side of signature box.

SetDisplayTimeStampPosY

```
public void SetDisplayTimeStampPosY(
  int YPos
);
```

Remarks: Sets the Y position in pixels of the start of the Time Stamp in the signature box.

Parameters: YPos-Y position to be set for start of display of Time Stamp.

GetDisplayTimeStampPosY

```
public int GetDisplayTimeStampPosY();
```

Remarks: Gets the current Y Position for the start of the Time Stamp String in the signature box.

Return Value: Y value in pixels relative to the bottom edge, if 0 then 5% in from bottom edge of signature box.

SetDisplayTimeStampSize

```
public void SetDisplayTimeStampSize(
  int Size
);
```

Remarks: Sets the Y size in pixels of the Time Stamp in the signature box.

Parameters: Size – Size of Time Stamp to set.

GetDisplayTimeStampSize

```
public int GetDisplayTimeStampSize();
```

Remarks: Gets the Y size in pixels of the Time Stamp in the signature box.

Return Value: Time Stamp size in pixels, if 0 then text size is 7.5% of Y size of signature box.

SetDisplayWindowRes

```
public void SetDisplayWindowRes(
  bool Mode
);
```

Remarks: Sets mode which renders signatures in lower (screen) resolution for compatibility in printing directly from VB. *MUST BE USED WHEN PRINTING DIRECTLY FROM A VISUAL BASIC FORM.*

Parameters: Mode -TRUE DisplayWindowRes mode = active, FALSE DisplayWindowRes mode = inactive

Return Value: True if Time Stamp is displayed, False if not displayed.

GetDisplayWindowRes

```
public bool GetDisplayWindowRes();
```

Remarks: Gets state of DisplayWindowRes.

Return Value: TRUE DisplayWindowRes mod = active, FALSE DisplayWindowRes mode = inactive.

IV. Image Properties

SetImageAnnotate

```
public void SetImageAnnotate(
  bool ImageAnnotate
);
```

Remarks: Sets the component to display the Annotation string as it applies to the WriteImageFile method.

Parameters: ImageAnnotate-Whether to enable rendering the specified annotation string in the rendered image.

GetImageAnnotate

```
public bool GetImageAnnotate();
```

Remarks: Gets the current setting for the component to display Annotation as it applies to the WriteImageFile method.

Return Value: True if Annotation is displayed, false if not displayed for Image.

SetImageAnnotateData

```
public void SetImageAnnotateData(
  int ImageAnnotatePosX,
  int ImageAnnotatePosY,
  int ImageAnnotateSize
);
```

Remarks: Sets display screen info for Annotate string. The default is the lower right corner, at ~8% of the screen high.

Parameters:
ImageAnnotatePosX - X Location to display Annotate string at in signature display window using Logical Tablet Coordinates.
ImageAnnotatePosY - Y Location to display Annotate string at in signature display window using Logical Tablet Coordinates.
ImageAnnotateSize - Size to display Annotate string, in logical tablet coordinate height.

SetImageAnnotatePosX

```
public void SetImageAnnotatePosX(
  int ImageAnnotatePosX
);
```

Remarks: Sets the X position in pixels of the start of the Annotation String in the signature box as it applies to the WriteImageFile method.

Parameters: ImageAnnotatePosX - X position for the start of the Annotation String to be set for Image.

GetImageAnnotatePosX

```
public int GetImageAnnotatePosX();
```

Remarks: Gets the current X position for the start of the Annotation String in the signature box as it applies to the WriteImageFile method.

Return Value: X position for start of the Annotation string, if 0 then text is positioned 5% in from right side of signature box for Image.

SetImageAnnotatePosY

```
public void SetImageAnnotatePosY (
  int ImageAnnotatePosY
);
```

Remarks: Sets the Y position in pixels of the start of the Annotation String in the signature box as it applies to the WriteImageFile method.

Parameters: ImageAnnotatePosY - Y position for the start of the Annotation String to be set for Image.

GetImageAnnotatePosY

public int GetImageAnnotatePosY();

Remarks: Gets the current Y position for the start of the Annotation String in the signature box as it applies to the WriteImageFile method.

Return Value: Y position for start of the Annotation string, if 0 then text is positioned 5% in from bottom edge of signature box for Image.

SetImageAnnotateSize

public void SetImageAnnotateSize (
  int *ImageAnnotateSize*
);

Remarks: Sets the Y size of the Annotation start in the signature box as it applies to the WriteImageFile method.

Parameters: ImageAnnotateSize - Y size of Annotation text in pixels for Image.

GetImageAnnotateSize

public int GetImageAnnotateSize();

Remarks: Gets the current Y size in pixels of the Annotation start in the signature box as it applies to the WriteImageFile method.

Return Value: Text size of the Annotation in pixels, if 0 then text size is 7.5% of the Y size of the signature box.

SetImageFileFormat

public void SetImageFileFormat(
  int *FileFormat*
);

Remarks: Sets the current format to use for Image files. The default is .BMP. The file extension is not assumed in the WriteImageFile function. Any extension can be specified, but it should match the specified file format. Note that writing an image file is completely different from a .sig file. An image file is just a standard image format of what is seen in the control and cannot be encrypted or decrypted by the SigPlus control. The .sig file format does not store an image, but rather uses a unique method of preserving the original signature data from the tablet. To create a metafile with a transparent background use a trio of instructions to set TabletOpaque = False, then WriteImageFile, the TabletOpaque = True. For all other image files, TabletOpaque must be true when the image file is written.

Parameters:
FileFormat-File format for Image files.
0=Compressed BMP (default) must have .bmp ext.
1=Uncompressed BMP must have .bmp ext.
2=Mono. BMP must have .bmp ext.
3=JPG Q=20 must have .jpg ext.
4=JPG Q=100 must have .jpg ext.
5=Uncompressed TIF must have .tif ext.
6=Compressed TIF must have .tif ext.
7=WMF (windows metafile) must have .wmf ext.
8=EMF (enhanced metafile) must have .emf ext.
9=TIF (1-bit) must have .tif ext.
10=TIF (1-bit inverted) must have .tif ext.

GetImageFileFormat

public int GetImageFileFormat();

Remarks: Returns the current setting of the image file format.

Return Value: Integer value of image file type as listed for the SetImageFileFormat method.

SetImagePenWidth

public void SetImagePenWidth(
  int *ImagePenWidth*
);

Remarks: Sets pen ink width as it applies the WriteImageFile method.

Parameters: ImagePenWidth-Pen ink width for Image.

GetImagePenWidth

public int GetImagePenWidth();

Remarks: Gets current pen ink width as it applies to the WriteImageFile method.

Return Value: Pen ink width for Image.

SetImageTimeStamp

public void SetImageTimeStamp(
  bool *ImageTimeStamp*
);

Remarks: Sets the component to display the Time Stamp as it applies to the WriteImageFile method.

Parameters: ImageTimeStamp – Component to be set to display developer provided Time Stamp.

GetImageTimeStamp

public bool GetImageTimeStamp();

Remarks: Gets the current setting for the component to display Time Stamp as applies to the WriteImageFile method.

Return Value: True if Time Stamp is displayed, False if not displayed for Image.

SetImageTimeStampData

public void SetImageTimeStampData(
  int *ImageTimeStampPosX*,
  int *ImageTimeStampPosY*,
  int *ImageTimeStampSize*
);

Remarks: Sets display screen info for Time and Date stamp. The default is the lower left corner, at ~8% of the screen height.

Parameters:
ImageTimeStampPosX-X, Location to display TimeStamp string at in signature display window using Logical Tablet Coordinates.

ImageTimeStampPosY-Y Location to display TimeStamp string at in signature display window using Logical Tablet Coordinates.

ImageTimeStampSize-Size to display TimeStamp string, in logical tablet coordinates high.

SetImageTimeStampPosX

```
public void SetImageTimeStampPosX(
  int ImageTimeStampPosX
);
```

Remarks: Sets the X position in pixels of the start of the Time Stamp as it applies to the WriteImageFile method.

Parameters: ImageTimeStampPosX – X position to be set for start of Time Stamp for Image.

GetImageTimeStampPosX

```
public int GetImageTimeStampPosX();
```

Remarks: Gets X position of the start of the Time Stamp in the signature box as it applies to the WriteImageFile method.

Return Value: X value in pixels relative to the left edge, if 0 then 5% in from the left side of signature box.

SetImageTimeStampPosY

```
public void SetImageTimeStampPosY(
  int ImageTimeStampPosY
);
```

Remarks: Sets the Y position in pixels of the start of the Time Stamp as it applies to the WriteImageFile method.

Parameters: ImageTimeStampPosY – Y position to be set for start of Time Stamp for Image.

GetImageTimeStampPosY

```
public int GetImageTimeStampPosY();
```

Remarks: Gets the current Y position in pixels for the start of the Time Stamp as it applies to the WriteImageFile method.

Return Value: Y value in pixels relative to the left edge, if 0 then 5% in from the bottom edge of signature box.

SetImageTimeStampSize

```
public void SetImageTimeStampSize(
  int ImageTimeStampSize
);
```

Remarks: Set the Y size in pixels of the Time Stamp in the signature box as it applies to the WriteImageFile method.

Parameters: ImageTimeStampSize – Size of Time Stamp to set for Image

GetImageTimeStampSize

public int GetImageTimeStampSize ();

Remarks: Gets the Y size in pixels of the Time Stamp in the signature box as it applies to the WriteImageFile method.

Return Value: Time Stamp size in pixels, if 0 then text size is 7.5% of Y size of signature box.

SetImageXSize

public void SetImageXSize(
  int *ImageXSize*
);

Remarks: Set the number of X pixels in the image provided by the WriteImageFile method.

Parameters: ImageXSize – Size in X pixels of the Image width.

GetImageXSize

public int GetImageXSize();

Remarks: Gets the current width in X pixels of the image as it applies to the WriteImageFile method.

Return Value: Number of X pixels of image.

SetImageYSize

public void SetImageYSize(
  int *ImageYSize*
);

Remarks: Set the number of Y pixels in the image height provided by the WriteImageFile method.

Parameters: ImageYSize – Size in Y pixels of the Image height.

GetImageYSize

public int GetImageYSize();

Remarks: Gets the current height in Y pixels of the image as it applies to the WriteImageFile method.

Return Value: Number of Y pixels of image height.