Software Developer Guide

SigWeb

Contents

Overview

SigWeb™ is a browser API that allows for rapid integration of signature capture functionality into a web app using Firefox, Chrome, Opera, Edge, and Internet Explorer 11 web browsers under Windows 8 and up. For display of the signature, SigWeb uses an HTML5 <canvas>.

Note: SigWeb is not for use in remote environments such as Citrix XenApp.

The SigWebTablet.js File

The SigWeb Browser API requires the SigWebTablet.js file to function. You will need to add a <script> tag to your page to reference SigWebTablet.js. You will find the latest SigWebTablet.js file at this link: **http://www.sigplusweb.com/SigWebTablet.js**. An example <script> tag can be found below. This example assumes that the .js file is in the same location as the page using SigWeb on the server.

Note: For best results and to allow support, Topaz recommends that you do not alter SigWebTablet.js.

```
<script type="text/javascript" src="SigWebTablet.js"></script>
```

SigWeb Functions

AutoKeyAddData(string KeyData)

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.

Parameters: 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**.

Refer here for example: **www.sigplusweb.com/sigwebtablet_autokeydemo.htm**.

ClearSigWindow(int Inside)

Remarks: Erases data either inside or outside the SigWindow based on the value of Inside.

Parameters: If 0, then signature data is erased from the SigWindow. If 1, then data is removed outside of the SigWindow (for clearing the hot spot buffer).

ClearTablet()

Remarks: Clears the signature object of any ink in the control. To clear ink from the LCD, refer to **LCDRefresh**.

GetDaysUntilCertificateExpires()

Remarks: SigWeb utilizes an SSL certificate to establish secure connections with HTTPS webpages. The SSL certificate expires periodically and must be replaced. Use this function to check the expiration of the SSL certificate bound to the SigWeb service.

Return Value: Amount of days until the SSL certificate bound to the SigWeb service expires. -900 is returned if the SSL certificate cannot be located. -999 is returned if there is an error while attempting to retrieve the SSL certificate.

GetDisplayPenWidth()

Remarks: Returns current ink thickness in the canvas in pixels (default is 5).

Return Value: Current ink thickness for the displayed signature in pixels as int.

GetEncryptionMode()

Remarks: Returns current EncryptionMode (default is 0).

Return Value: Value of encryption mode as integer:
0 = no encryption
1 = weak encryption
2 = higher-security encryption mode
3 = highest-security encryption mode

GetImagePenWidth()

Remarks: Gets current pen ink width for use with **GetSigImageB64** (default is 5).

Return Value: Pen ink width for image as integer.

GetImageXSize()

Remarks: Gets the current width in X pixels for use with **GetSigImageB64**.

Return Value: Number of X pixels of image as integer.

GetImageYSize()

Remarks: Gets the current width in y pixels for use with **GetSigImageB64**.

Return Value: Number of Y pixels of image as integer.

GetJustifyMode()

Remarks: Gets the current justification mode: how the signature is sized and positioned in the signature box.

Return Value: Justification mode as integer:
0 = No justification (default)
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)

GetJustifyX()

Remarks: Gets the buffer size of the right and left side (default is 0).

Return Value: Gets the buffer size in pixels as integer (right and left side).

GetJustifyY()

Remarks: Gets the buffer size of the top and bottom (default is 0).

Return Value: Gets the buffer size in pixels as integer (top and bottom).

GetKeyString()

Remarks: Provides hash of the encryption data in ASCII compatible format (default is "0000000000000000"). Should use **AutoKeyAddData** first.

Return Value: Hash of encryption data as string (32 characters total).

GetLCDCaptureMode()

Remarks: Gets the current **LCDCaptureMode** for the tablet.

Return Value: Mode the LCD is set to capture signatures in as integer:
1 = The tablet is set for 'auto erase' mode (clears LCD in basically 4 seconds).
2 = The tablet is set to persistent ink capture (required for using the LCD to display text/graphics). This keeps the data on the tablet until **LCDRefresh** is called.

GetSigCompressionMode()

Remarks: Returns compression mode for signatures.

Return Value: Mode for compression of signature as integer:
0 = no compression (default)
1 = lossless compression at a 4 to 1 ratio
2-**3** = compression ratio of signature stored in in .sig file where points start getting thrown out of the signature string. Topaz does not recommend compressing beyond setting 1 unless size is more important than signature quality. Generally, modes 2 and 3 are safe to use.

GetSigImageB64(Callback Function)

Remarks: Returns the signature as a PNG image in base64 string format. Parameter is a callback function name with which to pass the image string.

Example: *An example of the call to GetSigImageB64 is below:*
GetSigImageB64(SigImageCallback);

An example of the callback function getting the image string: **function SigImageCallback(str)**
{
//assign base64 string to form field
document.FORM1.sigImageData.value = str;
}

Return Value: base64 PNG image string

Note: Please be sure to add the callback function directly as JavaScript to your webpage in a <script> tag.

GetSigString()

Remarks: Returns the signature as a hexadecimal string. This string has all the biometric data included for verification of the signer's identity; however, you still need to use **AutoKeyAddData** for the signature's intent.

Return Value: SigString as ASCII hex string. Compression and Encryption affect the return itself (see **SetSigCompressionMode** and **SetEncryptionMode** calls).

GetSigWebVersion()

Remarks: Returns the version of the SigWeb service that can be used for determining what SigWeb functionality is supported by the SigWeb Service on the client's computer.

Return Value: string containing the SigWeb version. The string is formatted "major.minor.build.patch". There is a period after each number in the string.

GetTabletBaudRate()

Remarks: Gets the baud rate for serial tablets as integer. This includes B, BSB®, and BBSB models.

Return Value: Current TabletBaudRate as integer. TabletBaudRate only affects B and BSB signature pads.

GetTabletComPort()

Remarks: Gets the COM port for serial tablets as integer. This includes B, BSB, and BBSB models.

Return Value: Current COM port setting as integer. COM port only affects serial and BSB signature pads. Typically, BSB signature pads will be set to COM9.

GetTabletComTest()

Remarks: Gets current hardware check mode, can be used to determine if tablet is connected. See **SetTabletComTest()** for example of usage.

Return Value: Current hardware check mode as integer, true if active, false if not.

GetTabletLogicalXSize()

Remarks: Gets current horizontal size for TabletModel as set in the SigPlus.ini.

Return Value: Current horizontal size used in representing signatures in Logical Tablet Coordinates as integer. These values are generally pre-set and should not be altered.

GetTabletLogicalYSize()

Remarks: Gets current vertical size for TabletModel as set in the SigPlus.ini.

Return Value: Current vertical size used in representing signatures in Logical Tablet Coordinates as integer. These values are generally pre-set and should not be altered.

GetTabletResolution()

Remarks: Gets the TabletResolution, a value for the dpi for signature capture.

Return Value: Current TabletResolution as integer. Usually this is set at 410 dpi.

GetTabletState()

Remarks: Indicates capture state of the tablet.

Return Value: Value of TabletState as integer. **1** enables SigWeb to access the selected COM or HSB port and opens the tablet for signature capture, **0** is off.

GetTabletType()

Remarks: Gets TabletType value.

Return Value: Value of TabletType as integer. See **SetTabletType**.

GetTabletXStart()

Remarks: Gets the position in logical tablet coordinates of the starting X pixel.

Return Value: Current X position in logical tablet coordinates of the upper left-hand corner of the component signature box as integer.

GetTabletXStop()

Remarks: Gets the position in logical tablet coordinates of the final X pixel.

Return Value: Current X position in logical tablet coordinates of the upper right-hand corner of the component signature box as integer.

GetTabletYStart()

Remarks: Gets the position in logical tablet coordinates of the starting Y pixel.

Return Value: Current Y position in logical tablet coordinates of the lower left-hand corner of the component signature box as integer.

GetTabletYStop()

Remarks: Gets the position in logical tablet coordinates of the final Y pixel.

Return Value: Current Y position in logical tablet coordinates of the lower right-hand corner of the component signature box as integer.

KeyPadAddHotSpot(int KeyCode, int CoordToUse, int XPos, int YPos, int XSize, int YSize)

Remarks: Defines the location of a tablet hot spot which is used by the developer to detect user pen taps. Should be used either with pen events (PenUp and/or PenDown) or a timer/loop. **Never place hot spots inside of a SigWindow.** See **SetSigWindow** for more details.

Parameters:
KeyCode = Integer value defining the hot spot index (this should be unique)
CoordToUse = Coordinate system used for this 'hot spot' (generally 1 for LCD coordinate points)
XPos = X position to start
YPos = Y position to start
XSize = X size in LCD coordinates
YSize = Y size in LCD coordinates

KeyPadClearHotSpotList()

Remarks: Clears the controls internal list of hot spots created using **KeyPadAddHotSpot** as shown above. This would include any and all hot spots created at all indices.

KeyPadQueryHotSpot(int KeyCode)

Remarks: Queries whether the specified hot spot has been tapped by the user. Returns a value if the control contains data that is within the hot spot (definition of the KeyCode) on the tablet.

Return Value: Number of points within the hot spot (KeyCode definition).

LCDGetLCDSize()

Remarks: Requests (from the SigPlus.ini file) the size of the LCD in question returned as one value. Part of the value is in the low bytes and the other part of the value is in the high bytes. There is an example below regarding its use:

Return Value: Size of the LCD as a single int value. Given the code example, lcdX should contain the LCD width, and lcdY should contain the LCD height.

Usage:

```
var lcdSize;
var lcdX=0;
var lcdY=0;
lcdSize = LCDGetLCDSize();
lcdX = lcdSize & 0xffff;
lcdY = (lcdSize >> 16) & 0xffff;
```

LCDRefresh(int Mode, int XPos, int YPos, int XSize, int YSize)

Remarks: Sends tablet a refresh command with 3 possible modes:
0 = Clear, LCD display is cleared at the specified location
1 = Complement (meaning invert the pixels at the given location)
2 = WriteOpaque, transfers contents of the background memory to the LCD display, overwriting the content of the LCD display

Note: The LCDRefresh() call works on 8 px boundaries.

Parameters:
Mode = Defined as above (**0** to **2**)
XPos = Start position of the X coordinate
YPos = Start position of the Y coordinate
XSize = X size in LCD pixels
YSize = Y size in LCD pixels

LCDSendGraphicUrl(int Dest, int Mode, int YPos, int XPos, bmp URL)

Remarks: This writes a bitmap image to the LCD from the URL specified.

Dest is as follows:
0 = Write image to the LCD directly (the foreground)
1 = Write image to the background memory of the tablet, can bring it up to foreground using **LCDRefresh**

Mode parameter is as follows:
0 = Clear, LCD display is cleared at the specified location
1 = Complement (meaning invert the pixels at the given location)
2 = WriteOpaque, transfers contents of the background memory to the LCD display, overwriting the content of the LCD display

Parameters:
Dest = Defined above as **0** or **1**
Mode = Defined as above (**0** to **2**)
XPos = Start position of the X coordinate
YPos = Start position of the Y coordinate
URL = path to bitmap you wish to display

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 (associated to the **SetSigWindow** function)

Parameters:
XPos = Start position of the X coordinate
YPos = Start position of the Y coordinate
XSize = X size in LCD pixels
YSize = Y size in LCD pixels.

LCDStringWidth(font Canvas Font, string Text)

Remarks: Takes a string and a canvas font to return how wide the string is in px. This is to determine if what you plan on writing to the LCD is going to fit the x size.

Return Value: Integer representing the LCD coordinates.

LCDWriteString(int Dest, int Mode, int XPos, int YPos, font Canvas Font, int Font Height, string Text)

Remarks: Used to write a string to the LCD.

Dest is as follows:
0 = Write image to the LCD directly (the foreground)
1 = Write image to the background memory of the tablet, can bring it up to foreground using **LCDRefresh**

Mode parameter is as follows:
0 = Clear, LCD display is cleared at the specified location
1 = Complement (meaning invert the pixels at the given location)
2 = WriteOpaque, transfers contents of the background memory to the LCD display, overwriting the content of the LCD display

Parameters:
Dest = Defined above as **0** or **1**
Mode = Defined as above (**0** to **2**)
XPos = Start position of the X coordinate
YPos = Start position of the Y coordinate
Canvas Font = Example: "10pt Arial"
Integer = Height (make this slightly larger than the canvas font size)
String = String to write to LCD

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, the number of points in the signature.

Reset()

Remarks: Closes connection to tablet and clears signature data. For LCD pads, the LCD screen is refreshed/cleared, the LCD and Signature Window are reset, keypad hotspots are cleared, and LCD capture mode is set to 1.

NOTE: Highly recommend using this method on web page dismissal, as it is a single asynchronous command.

SetDisplayPenWidth(int PenWidth)

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

Parameters: Integer representing the pen width.

SetDisplayXSize(int Pixels)

Remarks: Sets the pixel width of the canvas targeted by SigWeb.

Parameters: Integer representing the pixel width of the display canvas.

SetDisplayYSize(int Pixels)

Remarks: Sets the pixel height of the canvas targeted by SigWeb.

Parameters: Integer representing the pixel height of the display canvas.

SetEncryptionMode(int EncryptionMode)

Remarks: Sets the EncryptionMode to use for **AutoKeyAddData**.

Parameters: Integer representing the mode of encryption you want to use.
0 = no encryption
1 = weak encryption
2 = higher-security encryption mode
3 = highest-security encryption mode.

SetImagePenWidth(int ImagePenWidth)

Remarks: Sets current pen ink width for use with **GetSigImageB64** (default is 5).

Parameters: Integer for the ImagePenWidth thickness in pixels (default is 5).

SetImageXSize(int ImageXSize)

Remarks: Set the width of the image in pixels (refer to **GetSigImageB64**).

Parameters: Integer for the ImageXSize width of the Image in pixels.

SetImageYSize(int ImageYSize)

Remarks: Set the height of the image in pixels (refer to **GetSigImageB64**).

Parameters: Integer for the ImageYSize height of the Image in pixels.

SetJustifyMode(int JustifyMode)

Remarks: Sets the current justification mode: how the signature is sized and positioned in the signature box (does not affect the signature data itself)

Parameters: Justification mode as integer:
0 = Normal no justification (default)
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).

The most popular choice is SetJustifyMode(5)

SetJustifyX(int Buffer)

Remarks: Sets the buffer size in LogicalTablet coordinates as a "signature free zone" of left and right edge of the SigPlus object.

Parameters: Integer for the justification X buffer size in pixels to be set.

SetJustifyY(int Buffer)

Remarks: Sets the buffer size in LogicalTablet coordinates as a "signature free zone" of the top and bottom edge of the SigPlus object.

Parameters: Integer for the justification Y buffer size in pixels to be set.

SetKeyString(string KeyString)

Remarks: Sets the KeyString into the SigPlus component. Only used if you don't want to use the original data you used to pass into **AutoKeyAddData**.

Parameters: KeyString, Hash of the data used to encrypt/decrypt the signature, key internally generated by SigPlus (32 characters provided originally by **GetKeyString**).

SetLCDCaptureMode(int CaptureMode)

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

Parameters: Integer representing the capture mode you want to use:
1 = The tablet is set for 'auto erase' mode (clears LCD in basically 4 seconds).
2 = The tablet is set to persistent ink capture (required for using the LCD to display text/graphics). This keeps the data on the tablet until **LCDRefresh** is called.

SetSigCompressionMode(int CompressionMode)

Remarks: Sets the current compression mode for signatures.

Parameters: CompressionMode: Mode for compression of signature
0 = No compression (the default)
1 = Lossless compression at a 4 to 1 ratio
2-**3** = Compression ratio of signature stored in SigString where points start getting thrown out. Topaz Systems does not recommend compressing beyond setting 1 unless size is more important than signature quality; however, generally modes 2 and 3 are safe to use.

SetSigString(string SigString, CanvasContext ctx)

Remarks: Puts signature back into SigWeb and displays it in the canvas. You will need a canvas on the web page to appropriately use this.

Parameters: SigString = String of signature in hexadecimal format. Then you will also need a context to a canvas on the web page. For example, let's assume you have a canvas on your page with an id="cnv". To get the context you would need to use:
var ctx = document.getElementById('cnv').getContext('2d');
and then pass in ctx as your second argument. This will render the signature into that canvas.

SetSigWindow(int Coords, int XPos, int YPos, int XSize, int YSize)

Remarks: This function sets a window in the control that allows ink to render inside of it. This is a means to separate signature area from hot spots (see **KeyPadAddHotSpot** for more details).

Coords: Coordinate system used for this hot spot:
0 = Logical tablet coordinates
1 = LCD coordinates (generally LCD coordinates are used)

Parameters:
Coords (as defined above)
XPos = Starting X position for the SigWindow
YPos = Starting Y position for the SigWindow
XSize = XSize of the window in the coordinate system you chose in Coords
YSize = YSize of the window in the coordinate system you chose in Coords

SetTabletBaudRate(int BaudRate)

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

Parameters: BaudRate is hard coded and generally not required to be set. However, you can refer to the %WINDOWS%\SigPlus.ini file for details regarding the baud rate for each tablet model.

SetTabletComPort(int Port)

Remarks: Sets the COM port using an int. Generally, the value is 1 or 2, but it can be anything. *For BSB pads, it's usually 9.* You can only set it when the **SetTabletState** is currently **0** (off).

Parameters: Whichever COM port into which you have plugged your signature pad.

NOTE: The SigPlus.ini file must have a parameter in place for using a B, BSB, or BBSB signature pad model, and that is: **ServerTabletType=0**. You should place this in the [Tablet] section at the top of your %WINDOWS%/SigPlus.ini file, right under the **TabletType=0** entry.

SetTabletComTest(bool State)

Remarks: Used to check if a Topaz signature pad is connected.

Parameters: You should first make sure the **SetTabletState** call is deactivated. Then you should set the SetTabletComTest State to **true**. Next, you need to set the **SetTabletState** to be active and then check the TabletState using **GetTabletState**. If the signature pad is connected, you will see that the **GetTabletState** returns a **1**, otherwise it will return a **0**. Here is an example:

```
SetTabletComTest(false);
SetTabletState(0, tmr);
SetTabletComTest(true);
if(tmr == null)
{
tmr = SetTabletState(1, ctx, 50);
}
else
{
SetTabletState(0, tmr);
tmr = null;
tmr = SetTabletState(1, ctx, 50);
}
if(GetTabletState() == 0)
{
//Cannot locate signature pad
SetTabletState(0, tmr);
SetTabletComTest(false);
}
else
{
//Located signature pad
SetTabletComTest(false);
}
//you are ready to proceed.
```

SetTabletLogicalXSize(int XSize)

Remarks: Sets the range of horizontal values to be used to represent signatures in TabletLogical coordinates. This has no relation to the displayed, image file, or tablet sizes. This is a set value for a reason, an internally used format. We do not recommend altering it.

Parameters: Integer value of TabletLogical size. Default depends on the tablet in question.

SetTabletLogicalYSize(int YSize)

Remarks: Sets the range of vertical values to be used in representing signatures in TabletLogical coordinates. This has no relation to the displayed, image file, or tablet sizes. This is a set value for a reason, an internally used format. We do not recommend altering it.

Parameters: Integer value of TabletLogical size. Default depends on the tablet in question.

SetTabletResolution(int Resolution)

Remarks: Sets the TabletResolution, an internal property associated with the tablet model and is set automatically by default through the SigPlus.ini file. The most common resolution is 410 dpi, (excluding ClipGem which is 275 dpi) but can be changed at the risk of affecting signature capture, although Topaz does not suggest changing it specifically.

Parameters: Resolution: internal tablet property.

SetTabletState(int State, CanvasContex ctx, milliseconds TimeOfDelay)

Remarks: THE FOLLOWING IS FOR ACTIVATING THE SIGNATURE PAD:

Enables signature capture through the signature pad. The parameter list is defined further below.

NOTE: You must create a 'timer' variable and set it equal to the call to SetTabletState, as in:

```
var tmr;
tmr = SetTabletState(1, ctx, 50);
```

You must use this 'tmr' value later when turning off the signature pad. It is used similar to how setInterval() and clearInterval() work in javascript. To deactivate the signature pad, see the other **SetTabletState** function here.

Parameters:
State = setting to **1** enables the tablet to capture signatures as above.
Canvas Contex = you will also need a context to a canvas on the web page. For example, let's assume you have a canvas on your page with an id="cnv". To get the context you would need to use: **var ctx = document.getElementById('cnv').getContext('2d');** and then pass in **ctx** as your second argument. This will render the signature into that canvas.
TimeOfDelay = this is the 3rd argument, and represents the amount of time (in milliseconds) that the canvas will be updated with the signature data. We suggest **50** but you can actually use whatever you might prefer.

SetTabletState(int State, tmr Timer)

Remarks: THE FOLLOWING IS FOR DEACTIVATING THE SIGNATURE PAD:

Enables signature capture through the signature pad. The parameter list is defined further below.

Parameters:
State = setting **0** disables the tablet to deactivate signature capture.
Timer = this is the original target of the call using **SetTabletState()** to activate the signature pad in the first place. As shown in the original example in the **SetTabletState** call, we set the tmr variable equal to the call to activate the tablet: **var tmr;**
tmr = SetTabletState(1, ctx, 50);
This tmr variable now must be used as the second argument of the call to deactivate signature capture, as in: **SetTabletState(0, tmr);**

SetTabletType(int TabletType)

Remarks: Determines if the tablet will accept data from a COM port, BSB, or HSB® pad, or other method of data input. Usually this is simply done using the %WINDOWS%/SigPlus.ini file, but you can override it as needed.

Parameters: Integer representing B, BSB and BBSB tablets use **0** as the TabletType. -HSB pads use **6** as the TabletType

Note that a BSB or a BBSB signature pad will also require this mode and typically is located on COM9 (see **SetTabletComPort**).

SetTabletXStart(int XStart)

Remarks: Sets the X position in TabletLogical coordinates for the location on the tablet where capture will begin. Topaz does not recommend changing it using this function.

Parameters: Integer, TabletLogical X position where signature capture will start.

SetTabletXStop(int XStop)

Remarks: Sets the X position in TabletLogical coordinates for the location on the tablet where capture will begin. Topaz does not recommend changing it using this function.

Parameters: Integer, TabletLogical X position where signature capture will stop.

SetTabletYStart(int YStart)

Remarks: Sets the Y position in TabletLogical coordinates for the location on the tablet where capture will begin. Topaz does not recommend changing it using this function.

Parameters: Integer, TabletLogical Y position where signature capture will start.

SetTabletYStop(int YStop)

Remarks: Sets the Y position in TabletLogical coordinates for the location on the tablet where capture will stop. Topaz does not recommend changing it using this function.

Parameters: Integer, TabletLogical Y position where signature capture will stop.

TabletModelNumber()

Remarks: Requests from the tablet the 'ModelNumber' of the tablet.

Return Value: This model number is returned as an integer value. Note that only LCD models are able to return a 'TabletModel'. There is a list below regarding which tablets return which value.
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

TabletSerialNumber()

Remarks: Requests from the tablet the 'SerialNumber' of the tablet.

Return Value: This serial number is returned as an integer value. Note that only LCD models are able to return a 'SerialNumber'. This can be a valuable tool for determining which "SE" model you have exactly. Given a model number of 58 from **TabletModelNumber**, you can check the TabletSerialNumber:
550 = TLBK766SE
551 = TLBK462SE
553 or **557** = TLBK755SE or TLBK750SE

If you have multiple LCD1x5 tablets (either T-L(BK)462 or T-L(BK)460 tablets), you can also use the TabletSerialNumber value to uniquely identify each signature pad.

SigWeb Pen Events

PenUp()

Remarks: THE FOLLOWING IS FOR ACTIVATING PENUP:

Enables the PenUp event through the signature pad. The elements are defined below.

NOTE: You must create an 'eventTmr' variable and set it to use setInterval() and well as clearInterval(). For details on these built-in javascript events, please see the page here:
https://www.w3schools.com/jsref/met_win_clearinterval.asp

For a SigWeb example, refer to the following:
var eventTmr;
eventTmr = setInterval(SigWebEvent, 20);

Note that the 'javascript function' argument needs to be 'SigWebEvent' as shown above. After that, you should pass in the number of milliseconds you would like to proceed before the event is called again. At this time, you now need to set up the event to handle when the pen is removed from the signature pad. To do this you need to specify a function to go to, as in this example:

```
onSigPenUp = function ()
{
processPenUp();
};
```

The **onSigPenUp** above is the only portion that is required. From there you set that equal to a function that you can name. For the sake of the demo, we chose to name it **processPenUp()** as shown above. From there you simply need to create your function within the **<script>** tag for your functions, as in:

```
function processPenUp()
{
        alert("Pen Up");
}
```

Use the event function however you might prefer. When you are done, be sure to clear the event with the **clearEvent()** function as defined on the page above.

PenDown()

Remarks: THE FOLLOWING IS FOR ACTIVATING PENDOWN:

Enables the PenDown event through the signature pad. The elements are defined below.

NOTE: You must create an 'eventTmr' variable and set it to use setInterval() and well as clearInterval(). For details on these built-in javascript events, please see the page here:
https://www.w3schools.com/jsref/met_win_clearinterval.asp

For a SigWeb example, refer to the following:

```
var eventTmr;
eventTmr = setInterval( SigWebEvent, 20 );
```

Note that the 'javascript function' argument needs to be 'SigWebEvent' as shown above. After that, you should pass in the number of milliseconds you would like to proceed before the event is called again. At this time, you now need to set up the event to handle when the pen is placed on the signature pad. To do this you need to specify a function to go to, as in this example:

```
onSigPenDown = function ()
{
        processPenDown();
};
```

The **onSigPenDown** above is the only portion that is required. From there you set that equal to a function that you can name. For the sake of the demo, we chose to name it **processPenDown()** as shown above.

From there you simply need to create your function within the **<script>** tag for your functions, as in:

```
function processPenDown()
{
        alert("Pen Down");
}
```

Use the event function however you might prefer. When you are done, be sure to clear the event with the **clearEvent()** function as defined on the page above.

Appendix A – AutoKeyAddData to AutoKeyAddANSIData

AutoKeyAddData encrypted a sigstring with a key that was specially formatted by Topaz. AutoKeyAddANSIData provides more advanced sigstring encryption techniques and is the recommended method to use moving forward. Since many customers have already utilized the AutoKeyAddData method to bind a key to their data, Topaz has provided a guide to determine which function to use to decrypt the signature data. The method requires using both the AutoKeyAddData and AutoKeyAddANSIData functions to attempt to decrypt the sigstring using a provided key. The below steps are only recommended if AutoKeyAddData was used previously in code to encrypt a signature with a key.

The following JavaScript source code shows how to properly decrypt a sigstring if the AutoKeyAddData was used previously:

```
function decryptSigString( key, sigString, encryptionMode, sigCompressionMode )
{
  ClearTablet();
  let isASCII = AutoKeyAddANSIData(key);

  SetEncryptionMode(encryptionMode);
  SetSigCompressionMode(sigCompressionMode);
  SetSigString(sigString);
  if(NumberOfTabletPoints() == 0){ //If the number of tablet points is equal to 0,
                        //then the signature data could not be decrypted
    if(isASCII.toLowerCase() == "false") {//If the key contains non-ASCII values,
                    //then attempt to decrypt using old method

      AutoKeyAddData(key);
      SetEncryptionMode(encryptionMode);
      SetSigCompressionMode(sigCompressionMode);
      SetSigString(sigString);
    }
  }
}
```

Appendix B – SigWeb Page Dismissal

With the release of Chrome 80 and above, web page dismissal handling has become inconsistent across multiple browsers. For example, with the exception of certain temporary measures and work-arounds (documented on the Topaz SigWeb page and elsewhere), Chrome 80+ no longer supports synchronous XML HTTP requests within the page dismissal. Because of this change to Chrome 80 and beyond, and potentially other browsers in the future, an Asynchronous Reset function has been provided in SigWeb 1.6.2+. The purpose of the Reset function is to return the SigWeb pad connection and pad to its default state - clearing and refreshing LCD settings and display, deleting hotspots if applicable, and closing the connection to the signature pad.

It is recommended that the Reset function be placed in the browser page dismissal handler. The Reset function need be used only once to return the signature pad and SigWeb pad connection to its default state either at the completion of the signature capture process and upon page dismissal. See example below:

```
window.onunload = function ()
{
        Reset();
};
```

The previous method for resetting or clearing a T-LBK462-HSB-R (LCD) signature pad:

```
function resetSignaturePad()
{
        LCDRefresh(0, 0, 0, 240, 64);
        LCDSetWindow(0, 0, 240, 64);
        SetSigWindow(1, 0, 0, 240, 64);
        KeyPadClearHotSpotList();
        SetLCDCaptureMode(1);
        SetTabletState(0, tmr);
}

window.onunload = function ()
{
        resetSignaturePad ();
};
```

The previous method for resetting or clearing a T-S460-HSB-R (non-LCD) signature pad:

```
function resetSignaturePad()
{
	ClearTablet ();
	SetTabletState(0, tmr);
}

window.onunload = function ()
{
	resetSignaturePad ();
};
```

New method for resetting or clearing the signature pad:

```
window.onunload = function ()
{
	Reset();
};
```