Software Integration Guide

Topaz SigPlusExtLite SDK

Designed for use in Chrome, Firefox, Opera, and Edge
Browser Extension Frameworks

Version 2.0

Table of Contents

Table of Contents

1.0 – Introduction

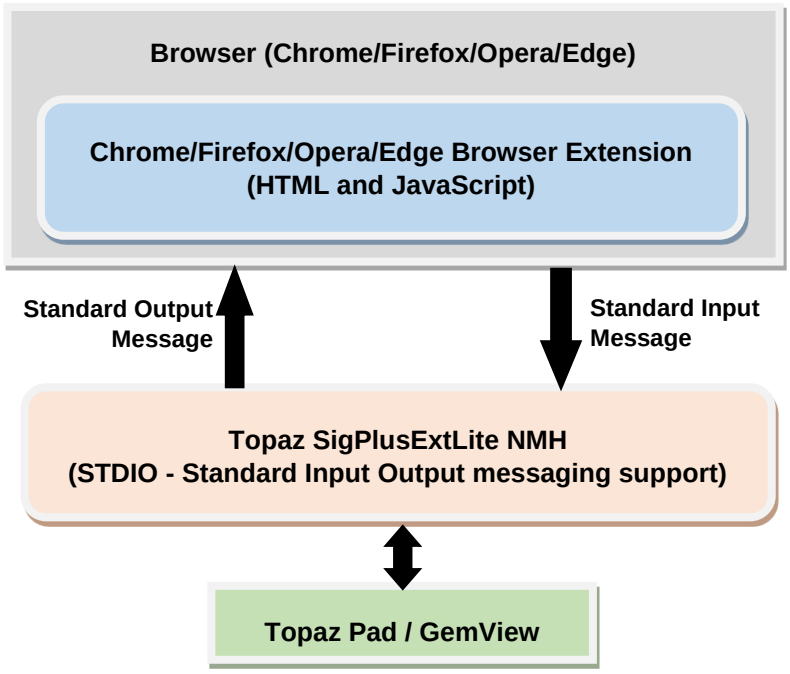
The Topaz SigPlusExtLite SDK allows web applications running in the Chrome, Firefox, Opera, and Edge (Chromium) browsers to capture biometric handwritten signatures using Topaz signature pads.

Note: *Microsoft has moved all Edge development to the Chromium web engine and all references to Edge in this document refer to the Edge (Chromium) browser*.

2.0 – Overview and Architecture

The SigPlusExtLite browser extensions 2.0 provide web pages with the capability to capture signatures using Topaz signature pads and GemView tablet displays connected to client desktops.

The diagram below shows the high-level overview of the solution with critical components involved.



2.1 – Topaz SigPlusExtLite SDK

The Topaz SigPlusExtLite SDK has been developed as a browser extension and a Native Message Handler (NMH). The NMH is a standard C# .Net application. The NMH processes the input text messages from the Chrome, Firefox, Opera, and Edge browsers and executes the requests asynchronously. When a task is completed it sends back the status or output data as an output text message. The NMH hosts all of the user interface functions for capture and display of signatures from devices.

2.2 – Chrome and Firefox Extensions/Webpage

Chrome and Firefox Extensions are HTML, JavaScript, and CSS based code modules that are launched during startup of the browser or launched on demand from web page JavaScript. Both extensions use JavaScript based Chrome Native Messaging APIs to launch and communicate with the Topaz SigPlusExtLite SDK using standard input and output messages.

The Opera and Edge browsers also support the installation and use of Chrome extensions.

3.0 – Key Features

The Topaz SigPlusExtLite SDK provides the following features:

- Set the minimum number of signature points required to qualify a signature as valid
- Initiate signature capture
- Clear the captured signature
- Get information about the connected signature pad
- Export a signature image as PNG/JPG (base64 encoded string)
- Export the signature data in an optionally compressed SigString format.
- Optionally export biometric signature data in encrypted base 64 string format.
- Export signature images with custom text drawn beside the handwritten squiggle.
- Present web-based forms on Topaz GemView devices
- Display idle screen with custom images or blank screen on GemView when not in use
- Ability to customize the signature capture window

4.0 – Operating System and Browser Support

The Topaz SigPlusExtLite SDK can be integrated into web pages running in Chrome, Firefox, Opera and Edge (Chromium-based) browser installed on Windows 7 and newer Windows 32-bit and 64-bit operating systems. The .NET Framework version 4.7.1 should be installed on end user's web client computers.

The following browser versions are supported:

1. Chrome, Version 77 and later
2. Firefox, Version 68 and later
3. Edge, Version 80 and later
4. Opera, Version 68 and later

5.0 – Signature Capture Devices

The SigPlusExtLite SDK supports capturing signatures using electronic signature pads and tablet displays from Topaz Systems.

6.0 – Sample Applications

1. For Topaz signature devices:
   https://sigplusweb.com/sign_chrome_ff_sigplusextlite.html

2. For both GemView and Topaz signature devices:
   https://www.sigplusweb.com/SigPlusExtLite_Demo/sample1/SigPlusExtLiteDemo.html

Note: For information on end user deployment, view Section 12.0 titled "End User Deployment" of this guide.

7.0 – SigPlusExtLite Integration for Signature Capture

7.1 – Launching the Extensions from a Webpage

The Native Messaging APIs allow the Extension to send and receive data from the NMH in JSON format. JSON objects are used to encapsulate both commands and data messages. The format of the JSON message is

{text1: value1, text2: value2}

where ‘text1’ and ‘text2’ are the names of the JSON parameters.

The SigPlusExtLite extension relies on custom HTML events for communication between the web page and the extensions and vice versa.

7.2 – Checking Extension Installation and Version

The following code snippet demonstrates a way to detect if the SigPlusExtLite extension is not installed or disabled in the browser. If not installed it displays an alert.

```
var isInstalled = document.documentElement.getAttribute('SigPlusExtLiteExtension-
installed');
if (!isInstalled) {
    alert("SigPlusExtLite extension is either not installed or disabled. Please install
or enable the extension.");
    return
}
```

Snippet 1 Checking if extension is installed

7.3 – Topaz SigPlusExtLite Version Information

The following code snippet demonstrates registering/raising the custom HTML event "GetVersionInfoEvent" to get the SigPlusExtLite version installed. It also demonstrates how to register and implement the "GetVersionInfoResponse" event for processing the output from the NMH. The NMH will respond with the GetVersionInfoResponse message.

```
var versionInfo = { "metadata": { "version": 1.0, "command": "GetVersionInfo" } };
var versionInfoData = JSON.stringify(versionInfo)
var element = document.createElement("MyExtensionDataElementVersionInfo");
element.setAttribute("msgAttributeVersionInfo", versionInfoData);
element.setAttribute("msg-Attribute-VersionInfo", versionInfoData);
document.documentElement.appendChild(element);
var evt = document.createEvent('Events');
evt.initEvent('GetVersionInfoEvent', true, false);
element.dispatchEvent(evt);
top.document.addEventListener('GetVersionInfoResponse', GetVersionInfoResponse, false);
function GetVersionInfoResponse() {
    var str = event.target.getAttribute("msgAttribute");
    if (str == null || str == "") {
        str = event.target.getAttribute("msg-Attribute");
    }
    var obj = JSON.parse(str);
    // Process the response
}
```

Snippet 2 GetVersionInfoEvent and GetVersionInfoResponse

7.3.1 – INPUT (GetVersionInfoEvent) Message

```
{
      "metadata": {
            "version": 1.0,
            "command": "GetVersionInfo"
      }
}
```

PARAMETER	DESCRIPTION
metadata	A container or group for metadata information.
version	The JSON structure version.
command	The command that describes the JSON structure's functionality.

Table 1 GetVersionInfoEvent message

7.3.2 – OUTPUT (GetVersionInfoResponse) Message

This message is sent back as an event attribute for the event **GetVersionInfoResponse** raised by the extension and handled by the web page.

```
{
        "status" : 1,
        "spelVersion": "2.0.25.0",
        "errorMsg": "message"
}
```

PARAMETER	DESCRIPTION
status	If the value is “1”, version information was retrieved successfully. If value is “0”, the version information could not be retrieved. -1 is returned if SigPlusExtLite is not installed or an older version of SigPlusExtLite is installed.
spelVersion	The version of SigPlusExtLite installed. The value of “spelVersion” attribute will have the version of SPEL if the value of “status” attribute is 1.
errorMsg	Error message returned from SigPlusExtLite SDK in case an error has occurred. The value of “errorMsg” attribute will have the error message if the value of “status” attribute is 0 or -1.

Table 2 GetVersionInfoResponse message

7.4 – Topaz SigPlusActiveX Version Information

The following code snippet demonstrates registering/raising the custom HTML event "GetActiveXVersionInfo" to get the SigPlus ActiveX version installed. It also demonstrates how to register and implement the "GetActiveXVersionInfoResponse" event for processing the output from the NMH. The NMH will respond with the GetActiveXVersionInfoResponse message.

```
var activeXVersionInfo = { "metadata": { "version": 1.0, "command": "
GetActiveXVersionInfo" } };
var activeXVersionInfoData = JSON.stringify(activeXVersionInfo)
var element = document.createElement("MyExtensionDataElementActiveXVersionInfo");
element.setAttribute("msg-Attribute-ActiveXVersionInfo", activeXVersionInfoData);
document.documentElement.appendChild(element);
var evt = document.createEvent('Events');
evt.initEvent('GetActiveXVersionInfoEvent', true, false);
element.dispatchEvent(evt);
top.document.addEventListener('GetActiveXVersionInfoResponse',
GetActiveXVersionInfoResponse, false);
function GetActiveXVersionInfoResponse() {
    var str = event.target.getAttribute("msgAttribute");
    if (str == null || str == "") {
        str = event.target.getAttribute("msg-Attribute");
    }
    var obj = JSON.parse(str);
    // Process the response
}
```

Snippet 3 GetActiveXVersionInfoEvent and GetActiveXVersionInfoResponse

7.4.1 – INPUT (GetActiveXVersionInfoEvent) Message

```
{
	"metadata": {
		"version": 1.0,
		"command": "GetActiveXVersionInfo"
	}
}
```

PARAMETER	DESCRIPTION
metadata	A container or group for metadata information.
version	The JSON structure version.
command	The command that describes the JSON structure's functionality.

Table 3 GetActiveXVersionInfoEvent message

7.4.2 – OUTPUT (GetActiveXVersionInfoResponse) Message

This message is sent back as an event attribute for the event **GetActiveXVersionInfoResponse** raised by the extension and handled by the web page.

```
{
	"status" : 1,
	"activeXVersion": "4.4.1.264",
	"errorMsg": "message"
}
```

PARAMETER	DESCRIPTION
status	If the value is "1", version information was retrieved successfully. -1 is returned if the version information could not be retrieved.
activeXVersion	The version of SigPlus ActiveX installed. The value of "activeXVersion" attribute will have the version of SigPlus ActiveX if the value of "status" attribute is 1.
errorMsg	Error message returned in case an error has occurred. The value of "errorMsg" attribute will have the error description if the value of "status" attribute is -1.

Table 4 GetActiveXVersionInfoResponse message

7.5 – Handling Unsupported Commands

If the SigPlusExtLite extension receives an unrecognized command it raises a custom HTML event named “UnsupportedCommandResponse” and passes the output message as an event attribute. Web pages should register and implement the “UnsupportedCommandResponse” event in the window load event for processing the output from the extension.

```
Window.addEventListener('load',(event)=>{
 top.document.addEventListener('UnsupportedCommandResponse', UnsupportedCommandResponse,
false);
})
function UnsupportedCommandResponse () {
    var str = event.target.getAttribute("msgAttribute");
    var obj = JSON.parse(str);
    // Process the response

}
```

Snippet 4 Sample UnsupportedCommandResponse listener

7.5.1 – OUTPUT (UnsupportedCommandResponse) Message

The SigPlusExtLite NMH returns an output message through the “errorMsg” attribute. It is a string attribute containing an error message for unsupported commands.

```
{
      "errorMsg": "Unsupported command. Cannot process the request."
}
```

PARAMETER	DESCRIPTION
errorMsg	Error message returned from NMH when unsupported command is received.

Table 5 UnsupportedCommandResponse message

7.6 – Topaz Device Detection

The “GetDeviceStatusEvent” needs to be registered to query the NMH for the status of the Topaz device connected.

Once the Topaz/GemView status detection task is completed, the extension raises a custom HTML event named “GetDeviceStatusResponse” and passes the output message as an event attribute. Web pages should register and implement the “GetDeviceStatusResponse” event to process the GetStatusDeviceResponse message.

The following code snippet demonstrates registering/raising the custom HTML event “GetDeviceStatusEvent” to check the device status. It also demonstrates registering and implementing the “GetDeviceStatusResponse” event to process the error message output.

```
var deviceStatus = { "metadata": {"version": 1.0, "command": "GetDeviceStatus"},
                     "deviceStatus": "" };
var deviceStatusData = JSON.stringify(deviceStatus)
var element = document.createElement("MyExtensionDataElementDeviceStatus");
element.setAttribute("msgeAttributeDeviceStatus", deviceStatusData);
document.documentElement.appendChild(element);
var evt = document.createEvent('Events');
evt.initEvent('GetDeviceStatusEvent', true, false);
element.dispatchEvent(evt);

top.document.addEventListener('GetDeviceStatusResponse', GetDeviceStatusResponse,
false);

function GetDeviceStatusResponse() {
    var str = event.target.getAttribute("msgAttribute");
    var obj = JSON.parse(str);
    // Process the response
}
```

Snippet 5 Detect TopazDevice connected

7.6.1 – INPUT (GetDeviceStatusEvent) Message

Following is the sample input message to check if a Topaz signature pad or GemView device is connected.

```
{
	"metadata": {
		"version": 1.0,
		"command": "GetDeviceStatus"
	},
	"deviceStatus": ""
}
```

PARAMETER	DESCRIPTION
metadata	A container or group for metadata information.
version	The JSON structure version.
command	The command that describes the JSON structure's functionality.
deviceStatus	Parameter for the device status. The value should be empty.

Table 6 GetDeviceStatusEvent Message

7.6.2 – OUTPUT (GetDeviceStatusResponse) Message

Following is a sample output GetDeviceStatusResponse message.

```
{
	"deviceStatus": 0,
	"errorMsg": "message"
}
```

PARAMETER	DESCRIPTION
deviceStatus	If value is “0”, no device was detected. If the value is “1”, Topaz signature pad was detected. If the value is “2”, GemView Tablet Display was detected. Negative value signifies an error. -1 is returned if an error occurs in detecting the device. -2 is returned if SigPlusExtLite is not installed or an older version of SigPlusExtLite is installed. -3 is returned if SigPlus drivers are not installed. -4 is returned if an older version of SigPlus drivers are installed.
errorMsg	Error message string returned from NMH if error occurred.

Table 7 GetDeviceStatusResponse message

This message is sent back as an event attribute for the event **GetDeviceStatusResponse** raised by the extension and handled by the web page.

8.0 – SigPlusExtLite GemView Device Integration

SigPlusExtLite 2.0 added the ability to push web forms to the GemView screen to be completed and signed by users, and can show an “Idle Screen” on the GemView to allow displaying messages on GemView when it isn’t being used for signing.

The Idle Screen can also prevent customers controlling the GemView from gaining control of the desktop if that is a concern.

Note: GemView is a second monitor. Desktop targets and task bars should not be displayed on the GemView if it will create a problem in the end user’s signing environment. This can be done easily using Windows configuration and desktop layout. Browsers can also be configured in kiosk mode.

8.1 – Push Browser Tab from Desktop to a GemView Device

A custom HTML event named “PushCurrentTabEvent” needs to be registered to push a browser tab containing a web form for signing to a GemView. Raising this event pushes the tab to the attached GemView. The following code snippet demonstrates registering and raising the custom HTML event:

```
var pushCurrentWindowTab = {"metadata" : {"version" : 1.0, "command" : "PushCurrentTab"},
                            "pushCurrentTab": true };
var pushCurrentWindowTabData = JSON.stringify(pushCurrentWindowTab);
var element = document.createElement("MyExtensionDataElementPushTab");
element.setAttribute("msgAttributePushCurrentTab", pushCurrentWindowTabData);
document.documentElement.appendChild(element);
var evt = document.createEvent('Events');
evt.initEvent('PushCurrentTabEvent', true, false);
element.dispatchEvent(evt);
```

Snippet 6 Raising "PushCurrentTabEvent"

8.1.1 – INPUT (PushCurrentTabEvent) Message

Following is a sample input JSON message to push the browser tab to a GemView device or recall it from the device.

```
{
	"metadata" : {
		"version" : 1.0,
		"command" : "PushCurrentTab"
	},
	"pushCurrentTab" : true
}
```

PARAMETER	DESCRIPTION
metadata	A container or group for metadata information.
version	The JSON structure version.
command	The command that describes the JSON structure's functionality.
pushCurrentTab	If "pushCurrentTab" value is true, the browser tab is pushed to GemView. If the value is false then the browser tab is pushed back to the desktop.

Table 8 PushCurrentTabEvent message

8.2 – Push Browser Tab from a GemView Device to Desktop

To recall the browser tab in GemView from Desktop, create a child window on the main monitor (Desktop) and use the exact same code snippet demonstrated in section but set "pushCurrentTab" to false. Sample shown below.

```
var pushCurrentWindowTab ={ "metadata" : {"version" : 1.0, "command" :
"PushCurrentTab"}, "pushCurrentTab": false };
var pushCurrentWindowTabData = JSON.stringify(pushCurrentWindowTab);
var element = document.createElement("MyExtensionDataElementPushTab");
element.setAttribute("msgAttributePushCurrentTab", pushCurrentWindowTabData);
document.documentElement.appendChild(element);
var evt = document.createEvent('Events');
evt.initEvent('PushCurrentTabEvent', true, false);
element.dispatchEvent(evt);
```

Snippet 7 Recall tab from GemView device

8.3 – Load “Idle Screen” on a GemView Device

To load the “Idle Screen”, a custom HTML event named “LoadStartEvent” needs to be registered. Raising the custom event enables the SigPlusExtLite NMH to display the idle screen on the GemView device as shown below.

```
var loadIdleScreen = {"metadata":{"version": 1.0, "command": "LoadIdleScreen" },
"configurationUrl": null, "duration": 500, "displayLogo":  true };
var imageData = JSON.stringify(loadIdleScreen)
var element = document.createElement("MyExtensionDataElementIdleScreen");
element.setAttribute("msgeAttributeIdleScreen", imageData);
document.documentElement.appendChild(element);
var evt = document.createEvent('Events');
evt.initEvent('LoadStartEvent', true, false);
element.dispatchEvent(evt);
```

Snippet 8 LoadStartEvent to start Idle Screen

8.3.1 – INPUT (LoadStartEvent) Message

Following is a sample input JSON message to load the idle screen.

```
{
	"metadata": {
		"version": 1.0,
		"command": "LoadIdleScreen"
	},
	"configurationUrl": null,
	"duration": 500,
	"displayLogo": true
}
```

PARAMETER	DESCRIPTION
metadata	A container or group for metadata information.
version	The JSON structure version.
command	The command that describes the JSON structure’s functionality.
configurationUrl	If “configurationUrl” is null, it uses local images to load idle screen in GemView. Otherwise uses location specified by referenced configuration xml file.
duration	Home screen slide duration in milliseconds.

displayLogo	If “displayLogo” value is true, then a logo is displayed on the home screen that can be used to dismiss the home screen. If the value is false, then the logo will not be displayed on the home screen.

Table 9 LoadStartEvent message

If the “configurationUrl” parameter is specified as null, then the SigPlusExtLite NMH will display a blank black overlay over the GemView screen. If the black overly is used and “displayLogo” is false the idle screen can be used to prevent a customer using Gemview from gaining control of the desktop. For custom images, a configuration XML containing the links to the images must be provided to the “configurationUrl” as shown below.

```
{
          "metadata": {
                    "version": 1.0,
                    "command": "LoadIdleScreen"
                     },
          "configurationUrl": http://example.com/SigPlusExtLite/Config/config_v2.1.xml",
          "duration": 500,
          "displayLogo": true

};
```

Snippet 9 LoadStartEvent message specifying custom Idle Screen

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
  <!-- the schema version -->
  <schema_version>1.0</schema_version>
  <!-- idle screen (attribute enabled = true/false) -->
  <idle_display enabled="true">
    <images>
        <!-- external links (attribute enabled = true/false) -->
        <external_links enabled="true">
                    <!-- 7" GemView -->
                    <device type="7">
                          <portrait>
                                <!-- list of external links -->
                                <external_link></external_link>
                                <external_link></external_link>
                          </portrait>
                          <landscape>
                                <!-- list of external links -->
                                <external_link></external_link>
                                <external_link></external_link>
                          </landscape>
                    </device>
                    <!-- 10" GemView -->
                    <device type="10">
                          <portrait>
                                <!-- list of external links -->
                                <external_link></external_link>
                          </portrait>
                          <landscape>
                                <!-- list of external links -->
                                <external_link></external_link>
                          </landscape>
                    </device>
                    <!-- 16" GemView -->
                    <device type="16">
                          <portrait>
                                <!-- list of external links -->
                                <external_link></external_link>
                          </portrait>
                          <landscape>
                                <!-- list of external links -->
                                <external_link></external_link>
                                <external_link></external_link>
                          </landscape>
                    </device>
        </external_links>
    </images>
  </idle_display>
</configuration>
```

XML ELEMENT	ATTRIBUTE/VALUE	DESCRIPTION
schema_version	1.0	The XML schema version
idle_display	enabled = true or false	If false, then the idle screen will be disabled. If true then the idle screen is displayed.
external_links	enabled = true or false	If false, then GemView uses local images to load the idle screen else it uses downloaded images to load idle screen.
device	type = 7,10,16	Images for different GemView devices (7”, 10”, 16”).
portrait		Grouping for portrait images
landscape		Grouping for landscape images
external_link		URL of the image to load idle screen in GemView.

Table 10 Configuration XML Schema for Idle Screen files

Samples can be found in our sample code referenced in section 6.0 – Sample Applications.

To improve performance and to avoid repeatedly downloading large image files, SigPlusExtLite first checks the hash generated from these files. If the hash does not match with any of the image files already present, it will assume the image to be a new one and download it and save it to a local repository.

The hash algorithm used to generate the image hash must be SHA256. It is mandatory to create a SHA256 hash and save it as text file (.txt) at the same location as the image. The file names of the image and the text file should be identical as SigPlusExtLite looks for .txt file with the same name as the image file first.

Example: If the image file name is logo.png then the text file with SHA256 hash should be logo.txt

If the text file of the hash is not found, the image will not be downloaded or displayed in the idle screen.

9.0 – Signature Capture and Data Export

Input messages query the NMH, trigger signature capture and specify all required parameters.

Output messages contain device status information and all signature data. They also supply error status information.

9.1 – Signature Capture

The signature capture window can be customized using the custom window attribute for customizing the signing window, signing area, signing toolbar and signing toolbar icons. The following sample shows the JSON input attributes required for signature capture.

```
var message = {
       "metadata": {
              "version": 1.0,
              "command": "SignatureCapture"
       },
       "firstName": "",
       "lastName": "",
       "eMail": "",
       "location": "",
       "imageFormat": 1,
       "imageX": 800,
       "imageY": 100,
       "imageTransparency": false,
       "imageScaling": false,
       "maxUpScalePercent": 0,
       "rawDataFormat": "ENC",
       "minSigPoints": 25,
       "penThickness": "1",
       "penColor": "#000000",
       "encryptionMode": "0",
       "encryptionKey": "EncryptionKey",
       "sigCompressionMode": 1,
       "customWindow": true,
}
```

PARAMETER	DESCRIPTION
metadata	A container or group for metadata information.
version	The JSON structure version.
command	The command that describes the JSON structure's functionality.
firstName	Signature Detail First Name as string. Can be "" if not used
lastName	Signature Detail Last Name as string. Can be "" if not used.
eMail	Signature Detail Email as string. Can be "" if not used.
location	The signing location as string. Can be "" if not used.
imageFormat	Format of the signature image to be exported after signature capture. Send 1 for JPG and 2 for PNG. The default format is PNG.
imageX	Physical width of the signature image to be exported in pixels.
imageY	Physical height of the image to be exported in pixels.
imageTransparency	Signature image background transparency as bool.
imageScaling	Flag indicating whether to scale the image or not as bool.
maxUpScalePercent	Maximum allowed upscale percentage as float (0 to 100).
rawDataFormat	By default, the SDK sends back the signature data as a compressed SigString. In addition, raw data can be exported after signature capture as a string. Use ENC as the value for this field to exporting raw data in the eSign Emcee format as an encrypted base 64 string.
minSigPoints	Minimum number of points required to qualify the signer marks as a valid signature.
penThickness	Thickness of the signature image as integer (1 to 5).
penColor	HTML color code as string.
encryptionMode	Level of encryption for SigString as integer. 0 is Clear text mode; 1 is DES; 2 is higher security encryption mode

encryptionKey	Information/data as string that will be hashed and used as a key within the encryption method
sigCompressionMode	Signature compression mode of the sigstring as integer: 0 to 2. 0 is no compression, 1 is lossless compression and 2 is lossy. Topaz doesn't recommend using 2 unless storage size is critical.
customWindow	Custom Window as boolean. True if a custom window should be displayed; False to display the default signature capture window. The custom window properties must be set beforehand using the structure defined in Section 9.4.

Table 11 SignatureCapture command and attributes

The extension registers a custom HTML event named "SignStartEvent" and web pages raise the custom HTML event "SignStartEvent" to capture a signature.

Once the JSON input message is formatted, it is passed to the browser extensions as an attribute to the "SignStartEvent" raised by the web page.

When the signature capture completes the SigPlusExtLite extension raises a custom HTML event named "SignResponse" and passes the output message as an event attribute. Web pages must register and implement the "SignResponse" event to process the message.

The following code snippet demonstrates raising the custom HTML event "SignStartEvent" to initiate signature capture and also registers the "SignResponse" event.

```
var messageData = JSON.stringify(message);
var element = document.createElement("MyExtensionDataElement");
element.setAttribute("messageAttribute", messageData);
document.documentElement.appendChild(element);
var evt = document.createEvent("Events");
evt.initEvent("SignStartEvent", true, false);

element.dispatchEvent(evt);

top.document.addEventListener('SignResponse', SignResponse, false);
function SignResponse(event) {
    var str = event.target.getAttribute("msgAttribute");
    var obj = JSON.parse(str);
    // Process the response
}
```

Snippet 10 Sample code using SignStartEvent and SignResponse event to capture a signature

The following interface appears for signature capture when the “SignStartEvent” is raised.

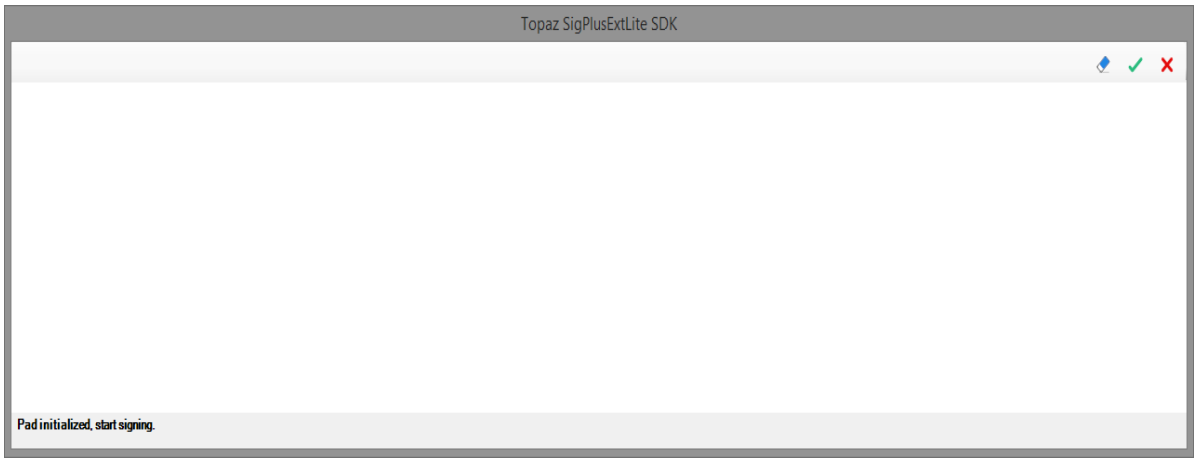


Figure 1 Signature capture windows showing capture toolbar

The toolbar will have options to Clear the Signature, Accept the Signature, and Cancel the signature. If connection with the signature device is successful the “Pad initialized, start signing” message appears in the status bar at the bottom of the window.

The SigPlusExtLite SDK then sends back an output message in the following scenarios after the signature capture is initiated using an input command.

1. Failed to initialize the signature capture dialog due to incorrect input data or any other device or driver related failures.
2. User cancelled signing.
3. User accepted a signature.
4. Signature not captured/accepted.

The SignResponse output message has a ‘status’ parameter to indicate if the signature capture was successful.

Table 12 is a sample output message and the description of its the attributes. This object is sent as an event attribute for the “SignResponse” event.

```
{
        "isSigned": true,
        "imgData": "Base64 formatted image data",
        "rawSigData": "Base64 formatted signature data in encrypted eSign Emcee format",
        "sigstring": "Signature data in compressed, encrypted SigString format",
        "padInfo": "Connected signature pad information",
        "errorMsg": "message"
};
```

PARAMETER	DESCRIPTION
isSigned	Value specifies whether a signature is captured or not as Boolean. Returns true if user accepted a signature and signature capture is successful. A true result is followed by image and raw signature data payloads in the message, otherwise they will be empty strings.
imgData	Signature image as base 64 string data. The format will be either JPG or PNG as specified in the input message. Contains an empty string if the signature is not captured.
rawSigData	Raw Signature data as eSign Emcee encrypted Base 64 String. Contains an empty string if the signature is not captured.
sigstring	Signature data in Topaz SigString format.
padInfo	Contains the information about the signature pad used for signature capture. The value will in the in the format Pad Name + Pad Serial Number.
errorMsg	In case of any failure or user cancelled signing, this field contains the error message. Client applications can use this info to handle the errors.

Table 12 Output message for SignResponse event

9.2 – Custom Signature Image Generation

The Topaz SigPlusExtLite SDK supports drawing custom text beside the signature during signature image generation.

The following sample shows the JSON input attributes required for exporting images that include custom text. Apart from the attributes described in Section 9.1, four extra attributes have to be defined which are described in the table that follows.

```
{
	"metadata": {
		"version": 1.0,
		"command": "SignatureCapture"
		},
	"firstName": "",
	"last Name": "",
	"eMail": "",
	"location": "Lo",
	"imageFormat": 2,
	"imageX": 300,
	"imageY": 150,
	"imageTransparency": true,
	"imageScaling": false,
	"maxUpScalePercent": 0.0,
	"rawDataFormat": "ENC",
	"minSigPoints": 25,
	"penColor": "#000000",
	"encryptionMode": "0",
	"encryptionKey": "By signing, you have acknowlEdged and agreed with the terms and
	conditions set forth by the Agreement.In addition, you are giving your consent to
	proceed with an electronic signature capture process, in accordance with the
	ESIGN Act.",
	"sigCompressionMode": 1,
	"displayCustomTextDetails": true,
	"customTextPercent": 50,
	"customTextLine1": "This is for demo purpose only",
	 "customTextLine2": "This is for demo purpose only”,
	"customWindow": false
};
```

PARAMETER	DESCRIPTION
displayCustomTextDetails	Display custom text details as bool.
customTextPercent	The percentage area that will be occupied in the field (20 to 50).
customTextLine1	Custom text line one as a string.
customTextLine2	Custom text line two as a string.

Table 13 Input message to set custom window text

The following figure shows sample image created with custom text.

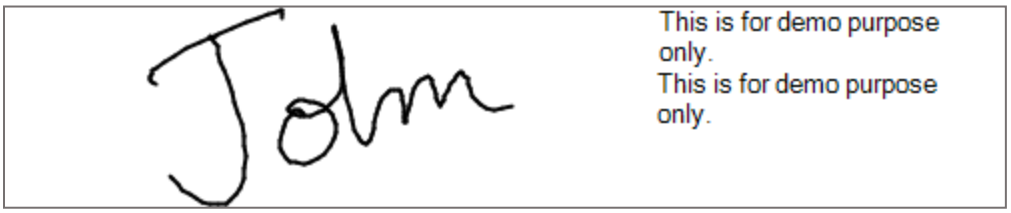


Figure 2 Signature image with custom text

9.3 – Signature Capture Default Window

The default capture window is displayed when the “customWindow” JSON input attribute is set to false in the Signature Capture JSON as shown in the sample below:

```
{..."customWindow": false };
```

The default signature capture window has a fixed height and width for all the GemView devices as well as desktops with Topaz signature pads connected. To change the default size, you can provide a custom height and width described in the sections that follow.

9.4 – Signature Capture Set Custom Window Attributes

The custom signing window message supports customizing attributes such as color, size, location, margins etc. for the various components in a signing window.

Following is a sample input JSON message to create a custom signing window.

```
var customWindowAttributes = {
	"metadata": {
		"version": 1.0,
		"command": "CustomSigningWindow"
	},
	"signingWindowProperties": {
		"signingWindow":{
			"backColor": "#f0f0f0",
			"size": { "width": 785, "height": 340 },
			"location": { "x": 0, "y": 0 },
			"windowState": 0,
			"formBorderStyle": 3,
			“windowTitle”: “Signing Window”
		},
		"signingArea":{
			"backColor": "#ffffff",
			"size": { "width": 785, "height": 240 },
			"location": { "x": 0, "y": 0 },
			"dock": 0
		},
		"signingWindowIconButtonsContainer": {
			"backColor": "#fed7b0",
			"size": 30,
			"padding": {
				"all": -1,
				"left": 0,
				"top": 0,
				"right": 1,
				"bottom": 0
			},
			"dock": 1
		},
		"signingWindowIconButtons": {
			"backColor": "#4b5320",
			"size": 24,
			"margin": {
				"all":-1,
				"left": 0,
				"top": 1,
				"right": 5,
				"bottom": 2
			}
		}
	}
}
```

Snippet 11 Sample CustomSigningWindow command input message setting customWindowAttributes

Creating a custom window requires raising the HTML event “CustomWindowEvent” to save custom window attributes into the local path. The page must also register the “CustomWindowResponse” event to process the output from the NMH. The following shows raising the CustomWindowEvent using the customWindowsAttributes defined in Snippet 11 Sample CustomSigningWindow command input message.

```
var customWindowData = JSON.stringify(customWindowAttributes);
var element = document.createElement("MyExtensionDataElementCustomWindow");
element.setAttribute("msgAttributeCustomWindow", customWindowData);
element.setAttribute("msg-Attribute-CustomWindow", customWindowData);
document.documentElement.appendChild(element);
var evt = document.createEvent("Events");
evt.initEvent("CustomWindowEvent", true, false);

element.dispatchEvent(evt);
top.document.addEventListener('CustomWindowResponse', CustomWindowResponse, false);


/*custom window response*/
function CustomWindowResponse(event) {
   var str = event.target.getAttribute("msgAttribute");
   if (str == null || str == "") {
       str = event.target.getAttribute("msg-Attribute");
   }
   var obj = JSON.parse(str);
   // Process the response
}
```

Snippet 12 Sample of CustomWindowEvent saving customWindowAttributes

The SigPlusExtLite NMH returns an output message through the “status” and “errorMsg” parameters. See sample output JSON message below:

```
{
      "status": 0,
      "errorMsg": "message"
}
```

PARAMETER	DESCRIPTION
status	If status is ‘0’ validation failed. If status is ‘1’ custom window attributes were validated and saved successfully. -1 is returned if a generic error occurs. -2 is returned if SigPlusExtLite is not installed or a version of SigPlusExtLite older than version 2 is installed.
errorMsg	String containing an error message.

Table 14 Sample output message from CustomWindowResponse to CustomWindowEvent

9.4.1 – Custom Signing Window

To enable the signature capture custom window, set the “customWindow” JSON input attribute to true as below.

```
{..."customWindow": true };
```

Signature capture custom window definition categories are:

1. Custom Signing Window
2. Custom Signing Area
3. Custom Icon Buttons Container (Toolbar)
4. Custom Icon Buttons

The Topaz SigPlusExtLite SDK supports custom signing windows for both GemView and Topaz signature pads. Signing window back color, size, location, window state and form border style attributes can all be customized.

The signing window has restrictions on minimum and maximum values for width and height. For Topaz signature pads the minimum width and height is 785 and 340 pixels respectively, while the maximum width and height of GemView depends on the display size. The table below shows the minimum and maximum values (width and height) for different GemView devices.

GEMVIEW TYPE	MINIMUM SIZE		MAXIMUM SIZE		GEMVIEW MODE
	WIDTH	HEIGHT	WIDTH	HEIGHT	
7 inch	785	340	1024	560	Landscape
	600	340	600	984	Portrait
10 inch	785	340	1280	760	Landscape
	785	340	800	1240	Portrait
16 inch	785	340	1366	728	Landscape
	768	340	768	1326	Portrait

Table 15 GemView signing windows size restrictions

To customize the signing window, construct the JSON string and define the various attributes as shown in the sample below.

```
"signingWindow":{
      "backColor": "#f0f0f0",
      "size": { "width": 785, "height": 340 },
      "location": { "x": 0, "y": 0},
      "windowState": 0,
      "formBorderStyle": 3
},
```

PARAMETER	DESCRIPTION	Default
backColor	HTML color code as hexadecimal string.	"#F0F0F0"
size	Size is defined by width and height attributes. Size can only be defined when the “windowState” attribute is set to “Normal”.	
width	Width as integer.	785
height	Height as integer.	340
location	Location is defined by X and Y coordinates. Location can only be defined when the “windowState” attribute is set to “Normal”.	
x	X coordinate of location as integer.	0
y	Y coordinate of location as integer.	0
windowState	Window State as integer. 0 is Normal State; 2 is Maximized State	0
formBorderStyle	Form Border Style as integer. 0 is None; 1 is Fixed Single; 2 is Fixed 3D; 3 is Fixed Dialog; 4 is Sizable	3
windowTitle	Text displayed as the title of the window	“Topaz SigPlusExtLite”

Table 16 signingWindow attributes for CustomSigningWindow command

9.4.2 – Custom Signing Area

The signing area has restrictions on minimum and maximum values for width and height. For Topaz signature pads, minimum width and height is 785 and 240 respectively while the GemView maximum width and height depends on the monitor size. The table below shows the minimum and maximum values (width and height) for different GemView devices.

GEMVIEW TYPE		MINIMUM SIZE		MAXIMUM SIZE		GEMVIEW MODE
		WIDTH	HEIGHT	WIDTH	HEIGHT	
7 inch		785	240	1024	448	Landscape
		600	240	600	787	Portrait
10 inch		785	240	1280	608	Landscape
		785	240	800	992	Portrait
16 inch		785	340	1366	582	Landscape
		768	340	768	1060	Portrait

Table 17 GemView signing area size restrictions

To customize the signing area, construct the JSON string and define the various attributes as shown in the sample below. The table that follows describes each of the attributes of the signing area.

```
"signingArea":{
       "backColor": "#ffffff",
       "size": { "width": 785, "height": 240 },
       "location": { "x": 0, "y": 0 },
       "dock": 0
},
```

PARAMETER	DESCRIPTION	Default
backColor	HTML color code as hexadecimal string.	"#FFFFFF"
size	Size is defined by width and height attributes. When the “dock” attribute is set to “None”, both width and height are applied. For left and right docked signing area only width is applied and for top and bottom docked signing area only height is applied.	
width	Width as integer.	785
height	Height as integer.	240
location	Location is defined by X and Y coordinates. Location can only be defined when the “dock” attribute is set to “None”.	
x	X coordinate of location as integer.	0
y	Y coordinate of location as integer.	0
dock	Dock as integer. 0 is None ;1 is Top;2 is Bottom; 3 is Left; 4 is Right; 5 is Fill. Defaults to none with X and Y attributes as 0.	0

Table 18 signingArea attributes for CustomSigningWindow command

9.4.3 – Custom Icon Buttons Container (Toolbar)

The custom signing toolbar can also be customized. The back color, size, padding and dock properties of the toolbar can be customized.

The signing toolbar has restrictions on minimum and maximum values for size. The size attribute is dependent on the dock attribute. For top and bottom docked toolbars size relates to height and width will be fixed to width of the signing window. For left and right docked toolbars, the size relates to width and the height of the toolbar will be fixed to the signing window height. The default dock of the toolbar is top.

For desktops with Topaz signature pads connected, minimum size is 30 while the maximum size depends on the monitor size.

The table below shows the minimum and maximum sizes for different GemView devices.

GEMVIEW TYPE	MINIMUM SIZE	MAXIMUM SIZE	GEMVIEW MODE
7 inch	30	112	Landscape
	30	196	Portrait
10 inch	30	152	Landscape
	30	248	Portrait
16 inch	30	145	Landscape
	30	265	Portrait

Table 19 GemView signing toolbar size restrictions

To customize the signing toolbar, construct the JSON string and define the various attributes as shown in the sample input JSON string below. The table that follows describes each of the attributes of custom toolbar.

```
"signingWindowIconButtonsContainer": {
        "backColor": "#fed7b0",
        "size": 30,
        "padding": {
                "all": -1,
                "left": 0,
                "top": 0,
                "right": 1,
                "bottom": 0
        },
        "dock": 1
},
```

PARAMETER	DESCRIPTION	Default
backColor	HTML color code as hexadecimal string.	“#F0F0F0”
size	Size as integer. Width is applied for left and right docked toolbars and height is applied for top and bottom docked toolbars.	30
padding	Padding is defined by all, left, top, right and bottom attributes.	
all	All as integer. If -1, individual values of left, right, top and bottom margins will be considered; If a positive value is provided, this value is applied to left, right, top and bottom paddings and the individual values will be ignored.	-1
left	Left pad as integer.	0
top	Top pad as integer.	0
right	Right pad as integer.	0
bottom	Bottom pad as integer.	0
dock	Dock as integer. 1 is Top;2 is Bottom; 3 is Left; 4 is Right.	1

Table 20 Custom toolbar attributes for CustomSigningWindow command

9.4.4 – Custom Icon Buttons

The Topaz SigPlusExtLite SDK supports custom toolbar icon buttons for both GemView and Topaz Signature pads. The back color, size and margin properties of the toolbar icon buttons can be customized.

The signing toolbar icon buttons has restrictions on minimum and maximum values for size. For desktops with Topaz signature pads connected, minimum size is 24 while the maximum size depends on the monitor size.

The table below shows the minimum and maximum sizes for different GemView devices.

GEMVIEW TYPE	MINIMUM SIZE	MAXIMUM SIZE	GEMVIEW MODE
7 inch	24	112	Landscape
	24	196	Portrait
10 inch	24	152	Landscape
	24	248	Portrait
16 inch	24	145	Landscape
	24	265	Portrait

Table 21 GemView toolbar icon size restrictions

To customize the toolbar icon buttons, construct the JSON string and define the various attributes as shown in the sample input JSON string below. The table that follows describes each attribute of the custom toolbar icon buttons.

```
"signingWindowIconButtons": {
      "backColor": "#4b5320",
      "size": 24,
      "margin": {
            "all":-1,
            "left": 0,
            "top": 1,
            "right": 5,
            "bottom": 2
      }
}
```

PARAMETER	DESCRIPTION	Default
backColor	HTML color code as string.	“#F0F0F0”
size	Size as integer. The value is applied to both height and width to maintain the aspect ratio.	24
margin	Margin is defined by all, left, top, right and bottom attributes.	
all	All as integer. If -1, individual values of left, right, top and bottom margins will be considered; If a positive value is provided, this value is applied to left, right, top and bottom margins and the individual values will be ignored.	-1
left	Left as integer.	0
top	Top as integer.	1
right	Right as integer.	10
bottom	Bottom as integer.	2

Table 22 Custom toobar icon attributes for CustomSignWindow command

9.5 – Signature Capture Get Custom Window Attributes

To retrieve the SigPlusExtLite custom window attributes the HTML event “GetCustomWindowAttributesEvent” needs to be registered. An input message should be sent to the NMH as an event attribute.

When the SigPlusExtLite get custom window attributes task completes the SigPlusExtLite extension raises an HTML event named “GetCustomWindowAttributesResponse” and passes the output message as an event attribute. Web pages should register and implement the “GetCustomWindowAttributesResponse” event to process the output from the extension. The following code snippet demonstrates raising the event “GetCustomWindowAttributesEvent” to get custom window attributes and registering the “GetCustomWindowAttributesResponse” event to process it.

```
var customWindowAttributes = {
     "metadata": {
         "version": 1.0,
         "command": "GetCustomWindowAttributes"
     }
 };
 var customWindowAttributesData = JSON.stringify(customWindowAttributes)
 var element = document.createElement("MyExtensionDataElementGetCustomWindow");
 element.setAttribute("msgAttributeGetCustomWindow", customWindowAttributesData);
 element.setAttribute("msg-Attribute-GetCustomWindow", customWindowAttributesData);
 document.documentElement.appendChild(element);
 var evt = document.createEvent('Events');
 evt.initEvent('GetCustomWindowAttributesEvent', true, false);
 element.dispatchEvent(evt);

 top.document.addEventListener('GetCustomWindowAttributesResponse',
 GetCustomWindowAttributesResponse, false);
 // get the custom window attributes response
 function GetCustomWindowAttributesResponse(event) {
     var str = event.target.getAttribute("msgAttribute");
     if (str == null || str == "") {
        str = event.target.getAttribute("msg-Attribute");
     }
     var obj = JSON.parse(str);
     // Process the response
 }
```

Snippet 13 Raising GetCustomWindowAttributes event and registering GetCustomWindowAttributesResponse

9.5.1 – INPUT (GetCustomWindowAttributesEvent) Message

The custom window attributes message gets the values of the custom window attributes that were saved when custom signing window was created. The input JSON message to get the version information is shown below.

```
{
	"metadata": {
		"version": 1.0,
		"command": "GetCustomWindowAttributes"
	}
}
```

PARAMETER	DESCRIPTION
metadata	A container or group for metadata information.
version	The JSON structure version.
command	The command that describes the JSON structure's functionality.

Table 23 GetCustomWindowAttributes message

9.5.2 – OUTPUT (GetCustomWindowAttributesResponse) Message

The SigPlusExtLite SDK sends back an output message through the "status", "signingWindowProperties" and "errorMsg" attributes. "signingWindowProperties" is a JSON structure that contains the signing window attributes. For further information on the CustomWindowAttributes structure please refer to Section 9.4 – Signature Capture Set Custom Window.

Table 24 is a sample output JSON message.

```
{
        "status" : 1,
        "signingWindowProperties": { ... },
        "errorMsg": "message"
}
```

Table 24 GetCustomWindowsAttributesResponse message

PARAMETER	DESCRIPTION
status	If the value is “1”, signing window attributes were retrieved successfully. If value is “0”, the signing window attributes could not be retrieved because the attributes were not present/saved or in case of a generic error and the default signing window attributes will be used. -1 is returned if SigPlusExtLite is not installed or an older version of SigPlusExtLite is installed.
signingWindowProperties	A JSON structure that contains the signing window attributes or properties defined in Section 9.4. The “signingWindowProperties” values are valid if the value of “status” attribute is 1.
errorMsg	Error message returned from SigPlusExtLite SDK in case of error. The value of “errorMsg” attribute will have the error message if the value of “status” attribute is less than or equal to 0.

10.0 – Cross Origin Iframe Setup

10.1 – Overview

Modern browsers limit the ability to communicate between webpages of different origins for security reasons. Websites that contain iframes to different websites contain cross origin iframes. Communication between the website that contains the iframe and the website referenced in the iframe is limited. Websites that utilize cross origin iframes with SigPlusExtLite must add additional code to allow SigPlusExtLite to send the collected signature data to the base webpage that the iframe(s) references. The webpage that references a cross origin iframe with SigPlusExtLite code must set the attribute “SigPlusExtLite-iFrameId” to the cross origin iframe id. For example:

```
document.documentElement.setAttribute('SigPlusExtLite-iFrameId',
"crossOriginIFrameID");
```

The SigPlusExtLite extension will use this attribute to automatically send the SigPlusExtLite signature data to the referenced webpage. The referenced webpage requires special code for properly identifying, retrieving, and processing the signature data. On the webpage that the iframe links to, the webpage must listen for and process the data sent to the window’s “message’ event. For example:

```
window.addEventListener("message", processMessage, false);
processMessage(){
  //process the SigPlusExtLite data
}
```

10.2 – Integration

On the webpage that hosts the iframe with “src” linked to a cross origin webpage, the attribute “SigPlusExtLite-iFrameId” must be set to the id of the cross origin iframe that contains the code for interacting with SigPlusExtLite. The following code should be added substituting “crossOriginIFrameID” for the id of your cross origin iframe:

```
document.documentElement.setAttribute('SigPlusExtLite-iFrameId', "
crossOriginIFrameID ");
…
<iframe id="crossOriginIFrameID"
src="https://website_of_different_origin.com/docusign/SigPlusExtLite_Cross_Origin.html">
```

Once the signature data is redirected to the webpage referenced in the src of the iframe, the referenced webpage should identify whether the data is from the extension and then use the signature data to complete the signing process.

Example:

If the sample website https://www.SampleDomainTwo.com references https://www.SampleDomainOne.com via an iframe, then https://www.SampleDomainTwo.com must add code to set an attribute in order for SigPlusExtLite to send signature data from one origin to the next.

The code would look like the following:

Code on https://www.SampleDomainTwo.com:

```
document.documentElement.setAttribute('SigPlusExtLite-iFrameId',
"crossOriginIFrameIDSampleDomainOne"); //set the “SigPlusExtLite-iFrameId” to the
iframe id of the iframe to send data to
…
<iframe id="crossOriginIFrameIDSampleDomainOne"
src=”https://www.SampleDomainOne.com/Sample_SigPlusExtLite_Cross_Origin.html"
height="825px" width="1000px">
```

Code on https://www.SampleDomainOne.com:

```
/**Add the below JS code to allow a Cross Origin iframe to trigger the SigPlusExtLite
response functions*/
window.addEventListener("message", processMessage, false);
function processMessage(event) {
    /*if the response from sigplusextlite can only handle the response*/
                if(event.data.message_id == "Topaz_SigPlusExtLite")
                {
                                /*response from post message*/
                                response = event.data.data;
                                //…handle the signature data
                                //…for more information on handling the signature
                                //…data, please reference the following demo:
                                //…
https://www.sigplusweb.com/CrossOriginIFrameDemos/SampleOne/SigPlusExtLiteDemo_CORS.h
tml
                }
};
```

For sample code to use for processing the signature data sent cross origin, please reference the following demo:
https://www.sigplusweb.com/CrossOriginIFrameDemos/SampleOne/SigPlusExtLiteDemo_CORS.html

The latest code for processing the signature data sent cross origin can also be obtained from the “Topaz SigPlusExtLite Background Extension” content script. Below are the steps for getting the code from the content script for Chrome, Firefox, and Edge Chromium:

Chrome Topaz SigPlusExtLite Background Extension Content Script

1. Confirm that the Chrome SigPlusExtLite extension is installed in the Chrome browser
2. Right click a webpage in Chrome browser and select “Inspect”
3. In the developer tools window that appears, select the “Sources” tab
4. In the “Sources” tab, select the arrow in the top left to change from “Page” to “Content scripts”
5. Under “Content scripts”, expand the “Topaz SigPlusExtLite Background Extension” section and select the javascript file
6. In the source, look for the code: “window.addEventListener("message", …);”

Firefox Topaz SigPlusExtLite Background Extension Content Script

1. Confirm that the Firefox SigPlusExtLite extension is installed in the Firefox browser
2. Right click a webpage in Chrome browser and select “Inspect”
3. In the developer tools window that appears, select the “Debugger” tab
4. Under “Sources”, expand the “Topaz SigPlusExtLite Extension” section
5. Select the javascript file under the “Topaz SigPlusExtLite Extension” section
6. In the source, look for the code: “window.addEventListener("message", …);”

Edge Chromium Topaz SigPlusExtLite Background Extension Content Script

1. Confirm that the Chrome SigPlusExtLite extension is installed in the Edge Chromium browser
2. Right click a webpage in Edge Chromium browser and select “Inspect”
3. In the developer tools window that appears, select the “Sources” tab
4. In the “Sources” tab, select the arrow in the top left to change from “Page” to “Content scripts”
5. Under “Content scripts”, expand the “Topaz SigPlusExtLite Background Extension” section and select the javascript file
6. In the source, look for the code: “window.addEventListener("message", …);”

Note: If there are three or more cross origin iframes (e.g. www.sampleOne.com links to an iframe for www.sampleTwo.com which links to an iframe for www.sampleThree.com which contains the SigPlusExtLite source code), then the following additional code will need to be added in each intermediate webpage (in this case, the webpage hosted in www.sampleTwo.com) to pass the data to the required destination:

```
window.addEventListener("message", processMessage, false);

function processMessage(event) {
  if(event.data.message_id == "Topaz_SigPlusExtLite"){
  var iframe = document.getElementById(“crossOriginIFrameSampleThree”);
                    if (iframe != null) {
                                /*get the source from the iframe */
                                const url = new URL(iframe.src);
                                /*get the origin from the url */
                                const origin = url.origin;
                                /*post the data to the specified iframe page*/
                                iframe.contentWindow.postMessage(event.data, origin);
                    }
  }
}
…
<iframe id="crossOriginIFrameSampleThree"
src="www.sampleThree.com/…/SigPlusExtLite_Cross_Origin.html"> //sampleTwo.com’s cross origin iframe to sampleThree’s webpage
```

11.0 – Accessibility

SigPlusExtLite provides hot keys and mouse over events to accept the signature, clear the signature, or cancel the signing process. The hot keys for accessibility are shown below.

Hot Key	DESCRIPTION
ALT + A	Accept signature
ALT + X	Cancel signing process
ALT + C	Clear signature

SigPlusExtLite is also speech-reader compatible. On load of the signature capture window, a user can navigate between controls on the window by selecting the tab key. Each control will specify its functionality. The speech-reader will read aloud the following text for each control that is tabbed over.

Control	Text	Tab Order
Accept Button	Accept button. Press ALT + A to accept the signature.	2
Cancel Button	Cancel button. Press ALT + X to cancel the signature.	3
Clear Button	Clear button. Press ALT + C to clear the signature.	1

12.0 – End User Deployment

Follow the steps outlined in the SigPlusExtLite User Installation Guide at:
https://www.topazsystems.com/Software/SigPlusExtLite_UserInstall.pdf